Automated Program Analysis: Revisiting Precondition Inference through Constraint Acquisition

Grégoire Menguy
CEA, List, France
gregoire.menguy@cea.fr

In collaboration with Sébastien Bardin, Nadjib Lazaar and Arnaud Gotlieb

***Abstract*—The following results are published at the International Joint Conference on Artificial Intelligence (IJCAI) 2022. The preprint can be found here:** **https://tinyurl.com/2p8x3x45.**

Program annotations under the form of function pre/postconditions are crucial for many software engineering and program verification applications. Unfortunately, such annotations are rarely available and must be retrofit by hand. In this paper, we explore how Constraint Acquisition (CA), a learning framework from Constraint Programming, can be leveraged to automatically infer program preconditions in a *black-box manner*, from input-output observations. We propose PRECA, the first ever framework based on active constraint acquisition dedicated to infer memory-related preconditions. PRECA overpasses prior techniques based on program analysis and formal methods, offering well-identified guarantees and returning more precise results in practice.

I. INTRODUCTION

Program annotations under the form of function pre/postconditions [1], [2], [3] are crucial for the development of correct-by-construction systems [4], [5], program refactoring [6]. They can benefit both a human or an automated program analyzer, typically in software verification – where they enable scalable (modular) analysis [7], [8] – or in symbolic execution – where it help to scale to big code [9]. Unfortunately, annotations are rarely available and must be retrofit by hand into the code, limiting their interest – especially for black-box third-party components.

Problem. Efforts have been devoted to *automatically infer* preconditions from the code, and contract inference is now a hot topic in Program Analysis and Formal Methods [10], [6], [11], [12], [13]. Since this problem is undecidable (as most program analysis problems), the goal is to design principled methods with good practical results. Yet, the state-of-the-art is still not satisfactory. While *white-box approaches* leveraging standard static analysis [1], [2], [3], [10] can be helpful, they quickly suffer from precision or scalability issues, have a hard time dealing with complex programming features (loops, recursion, dynamic memory) and cannot cope with black-box components. On the other hand *black-box methods*, leveraging test cases to dynamically infer (likely) function contracts [6], [11], [13], overcome static analysis limitations on complex codes and have drawn attention from the software engineering community [14]. Yet, they heavily depend on the quality of the underlying test cases, which are often simply generated at random, given by the users [6] (passive learning), or automatically generated during the learning process – but without any clear coupling between sampling and learning [11], [13] – and so, show no clear guarantee on the inference process.

Constraint Acquisition. Constraint programming (CP) [15] has made considerable progress over the last forty years, becoming a powerful paradigm for modelling and solving combinatorial problems. However, modelling a problem as a constraint network still remains a challenging task that requires expertise in the field. Several constraint acquisition (CA) systems have been introduced to support the uptake of constraint technology by non-experts. Especially, rooted in version space learning, CONACQ is presented in its passive and active versions [16]. Based on solutions and non-solutions labelled by the user (acting as an oracle), the system learns a set of constraints that correctly classifies all examples given so far. This is an active field of research, with many proposed extensions, for example allowing partial queries [17]. However, even though CONACQ enjoys strong theoretical foundations, such CA systems are hard to put in practice, as they require to submit *thousands of queries to a user*. In automated program analysis, the huge number of queries is not a problem as long as a program plays the oracle.

II. GOAL AND MOTIVATION

In this paper, we explore the potential of Constraint Acquisition for *black-box precondition inference*. To the best of our knowledge, this is the *first* application of CA to program analysis and our overall results show its potential there.

We focus on *memory-related preconditions* – e.g., predicates stating on which inputs a function can be executed without leading to a memory violation – in a *black-box manner*. Let us consider the prototype[1] of function `find_first_of` in

[1]While PRECA does not need the source code, it is given in Listing 2 for pedagogy purpose

Listing 1 (from Frama-C [7] test suite). We aim to infer which values of `a`, `m`, `b` and `n` are accepted without relying on the source code – still we can execute the code over chosen input and observe results.

```
void find_first_of(int* a,int m,int *b,int n)
```

Listing 1: Function prototype

From white-box to black-box. White-box analysis (such as P-Gen [18]) uses the program source code to infer preconditions. Yet several practical scenarios are impractical for white-box methods. First, having the *whole* source code is often unrealistic (many projects embed third-party components). Second, in practice program analyzers focus on a single programming language, but many projects use combinations of them (e.g., inline assembly in C code). Third, despite huge progress in the past decades, white-box program analysis still suffers on large or complex codes (unbounded loops, recursion, dynamic allocation, etc.) possibly leading to serious scalability or precision issues. Fourth, obfuscation is common in certain ecosystems to make reverse engineering harder and thwart white-box analysis. In all these scenarios, black-box methods are the sole option. Yet, as generalization is involved, black-box methods can compute *incorrect* preconditions (i.e., formula actually being not preconditions).

Black-box passive learning is not enough. Black-box methods should exercise the function under analysis on a representative set of test cases to infer relevant preconditions. A solution is to assume that users can provide such tests and leverage passive learning. Yet, this is often unrealistic – especially when the source code is not available. Moreover, random testing is rarely satisfactory, e.g., with 100 random test cases, both Daikon [6] and PIE [11] infer here an *incorrect* precondition for `find_first_of`.

Active learning. Gehr et al. [13] performs active learning, generating test cases automatically. Such approaches are more actionable and less sensitive to user bias. Still, methods developed so far lack theoretical guarantees. Indeed, they cannot ensure that all useful test cases have been considered. Gehr et al. method infers in $\approx 700s$ an *incorrect* precondition for `find_first_of`, generating 177 test cases.

PRECA insights. Our method performs black-box *precondition inference* through active constraint acquisition [16]. Unlike previous active approaches, PRECA mixes the sampling and learning phases which enables to show good theoretical properties. Indeed, when a test case is generated, PRECA directly observes how the function behaves on it and updates its search space accordingly. As such, given a set of constraints B called the bias, PRECA will generate all test cases to ensure convergence modulo B. Thus, if all queries can be exactly classified and if B is expressive enough, PRECA returns the *weakest precondition*. Regarding our example, it infers the (correct) *weakest precondition* $(m > 0 \Rightarrow valid(a)) \wedge (m > 0 \wedge n > 0 \Rightarrow valid(b))$, where $valid(p) \equiv (p \neq NULL)$, in 172s, with 45 test cases.

Alg.	Active?	Success.	#Test cases	Time
Daikon	no	no	100	0.6s
PIE	no	no	100	11s
Gehr et al.	yes	no	177	700s
P-Gen	(white-box)	(do not apply)	-	-
PRECA	**yes**	**yes**	**45**	**172s**

TABLE I: `find_first_of` results, no source code

III. CONTRIBUTIONS

Our main contributions are the following:

- We propose PRECA, the first ever (CONACQ-like) framework based on active constraint acquisition and dedicated to infer preconditions. We show that PRECA enjoys much better theoretical correctness properties than prior black-box approaches. Indeed, if our learning language is expressive enough, PRECA is *guaranteed* to infer the *weakest precondition*;
- We describe a specialization of PRECA to the important case of memory-related preconditions. Especially, we propose a dedicated constraint language (including memory constraints) for the problem at hand, as well as domain-based strategies to make the approach more efficient in practice;
- We experimentally evaluate the benefits of our technique on several benchmark functions. The results (see Table II) show that PRECA significantly outperforms prior precondition learners, be it black-boxes or white-boxes – which came as a surprise. For implicit postcondition, with a 1h time budget, PRECA handles 92% of our dataset against 52% and 74% respectively for black- and whitebox methods. For explicit postcondition, PRECA infer 41% of the dataset against 23% and 34% for other black- and white-box methods. Overall, PRECA with 5s budget per sample performs better than prior approaches with 1h per sample.

Overall, it turns out that seeing the precondition inference problem as a Constraint Acquisition task is beneficial, leading to good theoretical properties and beating prior techniques.

	1s		5s		5 mins		1h	
	$\#WP_\top$	$\#WP_Q$	$\#WP_\top$	$\#WP_Q$	$\#WP_\top$	$\#WP_Q$	$\#WP_\top$	$\#WP_Q$
Daikon	1.4/50	0.4/44	1.6/50	0.4/44	1.6/50	0.4/44	1.6/50	0.4/44
↳ PRECA	2/50	1/44	2/50	1/44	2/50	1/44	2/50	1/44
↳ Both	3.3/50	0/44	5.7/50	0/44	5.7/50	0/44	5.7/50	0/44
PIE	16.4/50	4.7/44	16.4/50	4.7/44	17.7/50	4.7/44	17.7/50	5.3/44
↳ PRECA	5/50	3/44	5/50	3/44	5/50	3/44	5/50	3/44
↳ Both	25.3/50	11.3/44	25.4/50	11.3/44	26.4/50	11.3/44	28.4/50	11.3/44
Gehr et al.	8.0/50	5.0/44	16.8/50	8.1/44	26.1/50	10.1/44	26.1/50	10.3/44
↳ PRECA	37/50	15/44	43/50	17/44	46/50	18/44	46/50	18/44
PRECA	29/50	11/44	**38/50**	**16/44**	**46/50**	**18/44**	**46/50**	**18/44**
↳ Random	29.9/50	12.1/44	29.9/50	12.1/44	30.0/50	12.1/44	30.0/50	12.1/44
P-Gen	**34/50**	**13/44**	37/50	15/44	37/50	15/44	37/50	15/44

$\#WP_\top$ (resp. $\#WP_Q$) is the number of inferred weakest precondition without (resp. with) a postcondition. We study 3 variations of Daikon and PIE: (i) original one (highlighted) on 100 random examples; (ii) on PRECA examples; (iii) on both random and PRECA examples. We study the original active Gehr et al. method (highlighted) and we feed it with PRECA examples. Finally, we study PRECA in active (highlighted) and in passive mode with 100 random queries (Random). P-Gen being a static method, we consider only its original form.

TABLE II: Results depending on the time budget

IV. Conclusion

We propose the first application of Constraint Acquisition to the Precondition Inference problem, a major issue in Program Analysis and Formal Methods. We show how to instantiate the standard framework to the program analysis case, yielding the first black-box active precondition inference method with clear guarantees. Moreover, our experiments for memory-oriented preconditions show that PreCA significantly outperforms prior works, demonstrating the interest of CA here.

```
int find(int* t, int n, int val) {
    for (int i = 0; i < n; i++)
        if (t[i] == val) return i;
    return n;
}

int find_first_of(int* a, int m, int* b, int n) {
    for (int i = 0; i < m; i++)
        if (find(b, n, a[i]) < n) return i;
    return m;
}
```

Listing 2: Implementation of find_first_of

References

[1] C. A. R. Hoare, "An axiomatic basis for computer programming," *CACM*, 1969.

[2] R. W. Floyd, "Assigning meanings to programs," in *Program Verification*. Springer, 1993.

[3] E. W. Dijkstra, "A constructive approach to the problem of program correctness," *BIT Numerical Mathematics*, 1968.

[4] B. Meyer, "Eiffel: A language and environment for software engineering," *JSS*, 1988.

[5] L. Burdy, Y. Cheon, D. R. Cok, M. D. Ernst, J. R. Kiniry, G. T. Leavens, K. R. M. Leino, and E. Poll, "An overview of jml tools and applications," *STTT*, 2005.

[6] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," *TSE*, 2001.

[7] F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski, "Frama-c: A software analysis perspective," *Formal Aspects of Computing*, 2015.

[8] P. Godefroid, S. K. Lahiri, and C. Rubio-González, "Statically validating must summaries for incremental compositional dynamic test generation," in *SAS*. Springer, 2011.

[9] M. Christakis, P. Müller, and V. Wüstholz, "Guiding dynamic symbolic execution toward unverified program executions," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 144–155.

[10] P. Cousot, R. Cousot, M. Fähndrich, and F. Logozzo, "Automatic inference of necessary preconditions," in *VMCAI'13*. Springer, 2013.

[11] S. Padhi, R. Sharma, and T. Millstein, "Data-driven precondition inference with learned features," *ACM SIGPLAN Notices*, 2016.

[12] A. Astorga, S. Srisakaokul, X. Xiao, and T. Xie, "Preinfer: Automatic inference of preconditions via symbolic analysis," in *DSN*. IEEE, 2018.

[13] T. Gehr, D. Dimitrov, and M. Vechev, "Learning commutativity specifications," in *CAV'15*, 2015.

[14] L. Zhang, G. Yang, N. Rungta, S. Person, and S. Khurshid, "Feedback-driven dynamic invariant discovery," in *ISSTA*. ACM, 2014.

[15] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[16] C. Bessiere, F. Koriche, N. Lazaar, and B. O'Sullivan, "Constraint acquisition," *Artificial Intelligence*, 2017.

[17] C. Bessiere, R. Coletta, E. Hebrard, G. Katsirelos, N. Lazaar, N. Narodytska, C.-G. Quimper, and T. Walsh, "Constraint acquisition via partial queries," in *IJCAI*, 2013.

[18] M. N. Seghir and D. Kroening, "Counterexample-guided precondition inference," in *ESOP*. Springer, 2013.