Automated Program Analysis: Revisiting Precondition Inference through Constraint Acquisition

Grégoire Menguy[1], Sébastien Bardin[1], Nadjib Lazaar[2], and Arnaud Gotlieb[3]

[1]Université Paris-Saclay, CEA, List, Palaiseau, France
[2]LIRMM, University of Montpellier, CNRS, Montpellier, France
[3]Simula Research Laboratory, Oslo, Norway

Reference: Menguy, G., Bardin, S., Lazaar, N., and Gotlieb, A. (2022). Automated program analysis: Revisiting precondition inference through constraint acquisition. In Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-ECAI 2022), Vienna, Austria.

Link: `https://www.ijcai.org/proceedings/2022/260`

keywords: constraint acquisition, active learning, weakest precondition, software engineering

We assure that this work has not already been presented to a KR audience in a major forum.

Abstract

The following results have been published at the International Joint Conference on Artificial Intelligence (IJCAI) 2022.

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.

1 Introduction

Program annotations under the form of function pre/postconditions (Hoare 1969; Floyd 1993; Dijkstra 1968) are crucial for the development of correct-by-construction systems (Meyer 1988; Burdy et al. 2005), program refactoring (Ernst et al. 2001). They can benefit both a human or an automated program analyzer, typically in software verification – where they enable scalable (modular) analysis (Kirchner et al. 2015; Godefroid, Lahiri, and Rubio-González 2011). Unfortunately, annotations are rarely available and must be retrofit by hand into the code, limiting their utility – 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 (Cousot et al. 2013; Ernst et al. 2001; Padhi, Sharma, and Millstein 2016; Astorga et al. 2018; Gehr, Dimitrov, and Vechev 2015). 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 (Hoare 1969; Floyd 1993; Dijkstra 1968; Cousot et al. 2013) 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 (Ernst et al. 2001; Padhi, Sharma, and Millstein 2016; Gehr, Dimitrov, and Vechev 2015), overcome static analysis limitations on complex codes and have drawn attention from the software engineering community (Zhang et al. 2014). Yet, they heavily depend on the quality of the underlying test cases, which are often simply generated at random, given by the users (Ernst et al. 2001) (passive learning), or automatically generated during the learning process – but without any clear coupling between sampling and learning (Padhi, Sharma, and Millstein 2016; Gehr, Dimitrov, and Vechev 2015) – and so, show no clear guarantee on the inference process.

Constraint Acquisition. Constraint programming (CP) (Rossi, Van Beek, and Walsh 2006) 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 (Bessiere et al. 2017). 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 (Bessiere et al. 2013). 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.

Goal and contributions. 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. 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 (see Table 1), 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 are summarized in Table 2. They show that PRECA significantly outperforms prior precondition learners, be it black-boxes or white-boxes – which came as a surprise. For implicit postconditions, 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 infers 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.

2 Relevance to KR

Constraint acquisition (Bessiere et al. 2017) is a machine learning framework that aims to infer a user's concept as a set of formally-defined constraints. In passive mode, the user gives classified examples himself, while in active mode, acquisition generates them automatically and asks the user to classify them. Constraint acquisition enjoys clear guarantees (termination, soundness, completeness) and returns a symbolic representation of the user concept. Thus, such results are easily interpretable and can be fed to other symbolic reasoning frameworks like constraint solvers. Hence, we believe that constraint acquisition integrates perfectly into the scope of KR.

On top of it, this paper proposes the first application of *constraint acquisition* to program analysis. We show how to adapt and extend constraint acquisition to infer function preconditions. It removes the main limitation of constraint acquisition: the dependency on a human user, restricting the manageable number of queries. We replaced such a user with an automatic oracle, removing this limitation. Our proposal, PRECA, benefits from the theoretical foundations of constraint acquisition. It offers stronger guarantees than the state-of-the-art precondition inference methods and outperforms them in practice. In that sense, this paper bridges constraint acquisition to the active fields of code verification and software engineering.

Grammar		
P	$:=$	$C \Rightarrow A \mid A \mid \neg A$
C	$:=$	$C \wedge C \mid A \mid \neg A$
A	$:=$	$valid(p_j) \mid alias(p_j, p_l) \mid deref(p_j, g)$
	$\mid$	$i_j = 0 \mid i_j < 0 \mid i_j \leq 0 \mid i_j = i_l \mid i_j < i_l \mid i_j \leq i_l$
Semantics of constraint over pointers		
$valid(p_j)$	$\equiv$	$p_j \neq NULL$
$alias(p_j, p_l)$	$\equiv$	$p_j = p_l$
$deref(p_j, g)$	$\equiv$	$p_j = \&g$ where $\&g$ is the address of g

p_j (resp. i_j) are pointers (resp. integers) and g is a global variable.

Table 1: Grammar of constraint language Γ

	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**
↳ BK	15/50	8/44	38/50	16/44	45/50	18/44	46/50	18/44
↳ Preproc.	19/50	9/44	36/50	16/44	45/50	18/44	46/50	18/44
↳ $\emptyset$	13/50	7/44	35/50	15/44	45/50	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 preconditions without (resp. with) a post-condition. We study 3 variations of Daikon and PIE: (i) the 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 feed it with PRECA examples. Finally, we study PRECA with its background knowledge and preprocess (highlighted), with background knowledge only (BK), with preprocessing only (Preproc.), without any of them ($\emptyset$) and in passive mode with 100 random queries (Random). P-Gen being a static method, we consider only its original form.

Table 2: Results depending on the time budget

References

Astorga, A.; Srisakaokul, S.; Xiao, X.; and Xie, T. 2018. Preinfer: Automatic inference of preconditions via symbolic analysis. In *DSN*. IEEE.

Bessiere, C.; Coletta, R.; Hebrard, E.; Katsirelos, G.; Lazaar, N.; Narodytska, N.; Quimper, C.-G.; and Walsh, T. 2013. Constraint acquisition via partial queries. In *IJCAI*.

Bessiere, C.; Koriche, F.; Lazaar, N.; and O'Sullivan, B. 2017. Constraint acquisition. *Artificial Intelligence*.

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

Cousot, P.; Cousot, R.; Fähndrich, M.; and Logozzo, F. 2013. Automatic inference of necessary preconditions. In *VMCAI'13*. Springer.

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

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

Floyd, R. W. 1993. Assigning meanings to programs. In *Program Verification*. Springer.

Gehr, T.; Dimitrov, D.; and Vechev, M. 2015. Learning commutativity specifications. In *CAV'15*.

Godefroid, P.; Lahiri, S. K.; and Rubio-González, C. 2011. Statically validating must summaries for incremental compositional dynamic test generation. In *SAS*. Springer.

Hoare, C. A. R. 1969. An axiomatic basis for computer programming. *CACM*.

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

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

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

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

Zhang, L.; Yang, G.; Rungta, N.; Person, S.; and Khurshid, S. 2014. Feedback-driven dynamic invariant discovery. In *ISSTA*. ACM.