NECESSARY LIBERAL PRECONDITIONS: A PROOF SYSTEM

Mster's Thesis in Informatics

ANRAN WANG
School of Computation, Information and Technology - Informatics
Technical University of Munich

Anran Wang: *Necessary Liberal Preconditions: A Proof System*

NECESSARY LIBERAL PRECONDITIONS: A PROOF SYSTEM
NOTWENDIGE LIBERALE VORBEDINGUNGEN: EIN BEWEISSYSTEM

Mster's Thesis in Informatics

ANRAN WANG, B.SC.
School of Computation, Information and Technology - Informatics
Technical University of Munich

Examiner:	Prof. Jan Křetínský
Supervisor:	Prof. Benjamin Lucien Kaminski, Lena Verscht, M.Sc.
Submission date:	15. September 2023

Anran Wang: *Necessary Liberal Preconditions: A Proof System*

DECLARATION

Ich versichere, dass ich diese Masterarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15. September 2023

Anran Wang

ABSTRACT

Short summary of the contents in English. . . a great guide by Kent Beck how to write good abstracts can be found here:

`https://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html`

ZUSAMMENFASSUNG

Kurze Zusammenfassung des Inhaltes in deutscher Sprache. . .

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

Part I

PART 1

Some text about this part.

1 BACKGROUND

With computer programs melded into almost every aspect of human life, we often not only require their efficiency, but also their correctness. We would like to know for sure that our programs deliver the desired results, and do not run forever. This is called *total correctness*. To know "for sure", we could verify programs using formal methods that have been developing for decades. One of such methods are *Hoare Triples* [4] proposed by Tony Hoare in 1969. A Hoare Triple contains three parts: a precondition, a program, and a postcondition. They are written as such: $\{F\}C\{G\}$.[1] It states that if the system is in a state that satisfies the precondition, then the state after the execution of the program will satisfy the postcondition, provided that the program terminates. This is called *partial correctness*.

Originally, Hoare Triples only deals with deterministic programs in a top-to-down manner, but nondeterminism can be added in a sensible way [2]. Here, Dijkstra presented the *weakest precondition* transformer (wp): starting with a post-condition, it works backwards and "guesses" what the precondition can be. wp is concerned with total correctness and is related to Hoare Triples by an implication: [2]

$$\forall G.\ G \implies wp.C.F : \{G\}C\{F\}$$

This connection not only tells us that

- given $wp.C.F$, any predicate G that implies it can be the precondition of a valid Hoare Triple: $\{G\}C\{F\}$;

it also shows when Hoare Triple will guarantee total correctness:

- given a valid Hoare Triple $\{G\}C\{F\}$, if its precondition G implies $wp.C.F$, then $\{G\}C\{F\}$ is valid for total correctness.

Sometimes, however, we deem nontermination a good behaviour, and proving partial correctness suffices. The *weakest liberal precondition* transformer [3] can be used in such occasions: if the system is in a state satisfying $wlp(C, F)$, then either F is reachable after the termination of C, or C does not terminate. wlp directly relates to Hoare Triples via an implication:

$$\forall G.\ G \implies wlp(C, F) : \{G\}C\{F\}$$

G is then called the *sufficient liberal precondition*, and finding it means we can prove the absense of errors in the program (if it terminates). In contrast, the

1 Originally it was written as $F\{C\}G$, but now it is often written with brackets around conditions instead of the program.

2 Here $wp.C.F$ is a function written in lambda-calculus style, it can be seen now as a function $wp(C, F)$. This form of writing proves to be simple and elegant in the upcoming sections.

necessary liberal precondition G (where $wlp(C, F) \implies G$) tells us that the system will not satisfy the postcondition F, once G is violated. Cousot et al. studied the matter from a practical perspective [1], they proposed inference tools and experimented in industrial codes. In this thesis, we aim to research this matter further with a more theoretical view. We would like to propose a proof system and prove its soundness and completeness similar as in [5], but using Dijkstra's guarded command language (GCL) [2].

Instead of the usual qualitative reasoning using logical predicates, we would like to study in a quantitive setting using functions that represent quantities such as expectations of program variables. While predicates map program states to true or false, functions map program states to $R\infty$, real numbers extended with (negative) infinity. In this setting, not only are infinities clear indication for nontermination, the transformers can also express more such as flow of quantitive information [6].

2 PRELIMINARIES

2.1 GUARDED COMMAND LANGUAGE

We use Dijkstra's (non-deterministic) guarded command language (nGCL) [2] to conceptualize a computer program and to include non-determinism. For better understanding, we use an equivalent form [6] of nGCL that is similar to modern pseudo-code:

$$C \;::=\; x := e \;\mid\; C; C \;\mid\; \{C\}\square\{C\} \;\mid\; \mathit{if}\ (\varphi)\ \{C\}\ \mathit{else}\ \{C\} \;\mid\; \mathit{while}\ (\varphi)\ \{C\}$$

2.2 HOARE TRIPLE

A Hoare Triple is a triple $\{G\}C\{F\}$ where G and F are logical predicates, and C is a program. G is called a *precondition*, F a *postcondition*. This triple reads, "if G is satisfied before the execution of C, then if C terminates, F is satisfied after the execution of C".

2.3 QUANTITY

Let Σ be the set of all program states, then the set of quantities is defined by

$$\mathbb{A} = \{f \mid f : \Sigma \to \mathbb{R}^{\pm\infty}\}$$

With the ordering

$$f \preceq g \text{ iff } \forall \sigma \in \Sigma : f(\sigma) \leqslant g(\sigma)$$

as well as join $\curlywedge$ and meet $\curlyvee$

$$f \curlywedge g = \lambda\sigma.min\{f(\sigma), g(\sigma)\}; \quad f \curlyvee g = \lambda\sigma.max\{f(\sigma), g(\sigma)\}$$

we can define a complete lattice $\langle \mathbb{A}, \preceq \rangle$. Note that all subsets in a complete lattice have supremum and infimum.

Informally, $f \preceq g$ is analogous to $F \implies G$ for predicates, $\curlywedge$ ($\curlyvee$) is analogous to $\wedge$ ($\vee$).

2.4 WEAKEST (LIBERAL) PREQUANTITY

Let $[\![C]\!](\sigma)$ be the set of all **reachable** states by program C from initial state σ, f a *postquantity*. Let $\bigcurlyvee S$ denote the supremum of set S, then:

$$wp.C.f.\sigma = \bigcurlyvee_{\tau \in [\![C]\!](\sigma)} f(\tau)$$

implication	**defines**
$G \implies wp(C, F)$	total correctness
$G \implies wlp(C, F)$	partial correctness
$wp(C, F) \implies G$	partial incorrectness
$wlp(C, F) \implies G$	???

Table 1: implications using wp and wlp[6]

As the name suggests, $wp.C.f.\sigma$ gives the "weakest", i.e. supremum, prequantity given post quantity f. Given the program, postquantity, initial state, the wp transformer gives the "biggest" element of all postquantities at all reachable final states. Since $f \preceq g$ means g is weaker than f, the biggest quantity is the weakest. Informally, fixing an initial state, whether a prequantity is valid for the weakest prequantity depends on whether the postquantity is valid for at least one of the final states.

The weakest liberal prequantity transformer is defined with the infimum:

$$wlp.C.f.\sigma = \bigcurlywedge_{\tau \in [\![C]\!](\sigma)} f(\tau)$$

2.5 NECESSARY AND SUFFICIENT CONDITIONS

We can now use some implications to define correctness or incorrectness as shown in Table 1.

We are interested in the last implication labeled with ???, where G is called the *necessary liberal precondition*, in a sense that

- all preconditions that lead to non-termination of C imply G,
- a preconditions that do not imply G leads to a postcondition that does not imply F.

Our goal is to study this implication and its connection to correctness/incorrectness.

3 GOAL: A PROOF SYSTEM

3.1 GOAL: A PROOF SYSTEM

This thesis aims to study the necessary liberal precondition(quantity) and its relation to Hoare Triple. In their paper, Cousot et al. studied the necessary preconditions and proposed static analysis rules with inference tools. We aim to research the same subject with a more theoretical approach in the flavor of [5]. Instead of defining big-step semantics, we use GCL and hopefully to come up with a proof system, and by doing so with GCL syntax, we hope to prove the completeness and soundness of the proof system using structural induction. Finally, we will look for interesting use cases demonstrating the necessary liberal precondition reasoning techniques and how they differ from other systems.

Part II

PART 2

Some text about this part.

Part III

APPENDIX

BIBLIOGRAPHY

[1] Patrick Cousot, Radhia Cousot, Manuel Fähndrich, and Francesco Logozzo. "Automatic inference of necessary preconditions." In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer. 2013, pp. 128–148.

[2] Edsger W Dijkstra. "Guarded commands, nondeterminacy and formal derivation of programs." In: *Communications of the ACM* 18.8 (1975), pp. 453–457.

[3] Edsger W. Dijkstra and Carel S. Scholten. "On substitution and replacement." In: *Predicate Calculus and Program Semantics*. New York, NY: Springer New York, 1990. ISBN: 978-1-4612-3228-5. URL: `https://doi.org/10.1007/978-1-4612-3228-5_2`.

[4] Charles Antony Richard Hoare. "An axiomatic basis for computer programming." In: *Communications of the ACM* 12.10 (1969), pp. 576–580.

[5] Edsko de Vries and Vasileios Koutavas. "Reverse hoare logic." In: *International Conference on Software Engineering and Formal Methods*. Springer. 2011, pp. 155–171.

[6] Linpeng Zhang and Benjamin Kaminski. "Quantitative Strongest Post." In: *arXiv preprint arXiv:2202.06765* (2022).

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of March 14, 2023 (`classicthesis` version 0.1).