NECESSARY LIBERAL PRECONDITIONS:
A PROOF SYSTEM

Mster's Thesis in Informatics

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

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

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 postcondition, 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 $\mathbb{R}\infty$, 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].

TODO: Rewrite; add chapter contents.

2

PRELIMINARIES

2.1 GUARDED COMMAND LANGUAGE

We use Dijkstra's (non-deterministic) guarded command language (GCL) [2] to conceptualize a computer program and to include non-determinism. For better understanding, we use an equivalent [1] form of nGCL that is similar to modern pseudo-code:

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

2.2 WEAKEST PRECONDITION

We define the weakest precondition transformer structurally in lambda-calculus style[2] as follows:

C	**wp.C.F**
skip	F
$\mathit{diverge}$	false
$x := e$	$F[x/e]$
$C_1; C_2$	$wp.C_1.(wp.C_2.F)$
$\mathit{if}\ (\varphi)\ \{C_1\}\ \mathit{else}\ \{C_2\}$	$(\varphi \wedge wp.C_1.F) \vee (\neg\varphi \wedge wp.C_2.F)$
$\{C_1\}\square\{C_2\}$	$wlp.C_1.F \vee wlp.C_2.F$
$\mathit{while}\ (\varphi)\ \{C'\}$	$\mathit{lfp}\ X.(\neg\varphi \wedge F) \vee (\varphi \wedge wp.C'.X)$

Table 1: The Weakest Precondition Transformer

$F[x/e]$ is F where every occurrence of x is syntactically replaced by e. $\mathit{lfp}X.f$ is the least fixed point of function f with variable X. Let $\Phi(X) := \neg\varphi \wedge F) \vee (\varphi \wedge wp.C'.X$ the characteristic function.

To justify this definition, we must first clarify the intended semantics/meaning of the wp-transformer. Let $[\![C]\!]$ denote the execution of program C, $[\![C]\!].\sigma$ denote the set of final states that can occur after the execution of C. (A state is a function

1 Specifically, $\mathit{if}\ (\varphi)\ \{C_1\}\ \mathit{else}\ \{C_2\}$ is equivalent to $\mathit{if}\ \varphi \to C_1\ []\ \neg\varphi \to C_2\ \mathit{fi}$; $\{C_1\}\square\{C_2\}$ is equivalent to $\mathit{if}\ \mathit{true} \to C_1\ []\ \mathit{true} \to C_2\ \mathit{fi}$.

2 For example, $wp.C.F$ can be seen as $wp(C, F)$ in "typical" style, where wp is treated as a function that has two parameters. The advantage of lambda-calculus style is scalability, we can simply extend the aforementioned function like $wp.C.F.\sigma$ where σ means the initial state. Here wp is treated as a function that has three parameters, if we were to write it in the "typical" style. It is then questionable whether we changed the semantics of wp.

that maps a program variable to a value. The set of states is denoted by $\Sigma = \{\sigma \mid \sigma : Vars \to Vals\}$.) If C is deterministic, $[\![C]\!].\sigma$ is the set of a single state, either a final state σ' or $\bot$ if the execution does not terminate. If C is non-deterministic, $[\![C]\!].\sigma$ can be a set with multiple elements. It is only sensible if we define with can rather than will/must, especially with non-deterministic programs, since their execution by definition will guarantee only little about final states, hence we would end up with mostly empty sets, which is not meaningful in our case.

TODO: Justify all the definitions except while.

TODO: Explain least point iteration from bottom.

2.3 DEFINING LOOPS

In Dijkstra's original paper[2], he defined *wp* for while-loops based on its (intended) semantics.

Let

$$WHILE = while(\varphi)\{C'\} \qquad IF = if\ (\varphi)\{C'; WHILE\}\ else\ \{skip\}$$

Rewriting Dijkstra's definition in a form conforming to our style, he defines

$$H_0(F) = (F \wedge \neg\psi) \qquad H_k(F) = (wp.IF.(H_{k-1}(F)) \vee H_0(F))$$

Intuitively, we can understand $H_k(F)$ as the weakest precondition such that the program terminates in a final state satisfying F after at most k iterations.

Then by definition:

$$wp.WHILE.F = (\exists k \geqslant 0 : H_k(F)) \tag{1}$$

Our definition is equivalent to this definition. Coincidentally, $H_k(F)$ is the k—th iteration from bottom $\bot$ to calculate the least fixed point of the characteristic function: $\Phi^k(\bot)$. Thus by finding the least fixed point, we've found a k that satisfies (1).

2.4 WEAKEST LIBERAL PRECONDITION

We define the weakest liberal precondition transformer in Table 2.

C	**wlp.C.F**
skip	F
diverge	true
$x := e$	$F[x/e]$
$C_1; C_2$	$wp.C_1.(wp.C_2.F)$
if (φ) $\{C_1\}$ else $\{C_2\}$	$(\varphi \wedge wp.C_1.F) \vee (\neg\varphi \wedge wp.C_2.F)$
$\{C_1\}\Box\{C_2\}$	$wlp.C_1.F \wedge wlp.C_2.F$
while (φ) $\{C'\}$	gfp $X.(\neg\varphi \wedge F) \vee (\varphi \wedge wp.C'.X)$

Table 2: The Weakest Liberal Precondition Transformer

3

A PROOF SYSTEM

3.1 A PROOF SYSTEM

In this section we study the necessary liberal precondition:

$$wlp.C.F \implies G$$

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 April 21, 2023 (`classicthesis` version 0.1).