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

TODO: Make first letter big?

TODO: Rewrite; add chapter contents.

2 PRELIMINARIES

2.1 THE ROOT OF ALL GOOD: HOARE TRIPLES

In 1969, C.A.R. Hoare published his famous article *An Axiomatic Basis for Computer Programming* [3] to explore the logic of computer programs use axioms and inference rules to prove the properties of programs. This system is known referred as Hoare Triples. He introduced sufficient preconditions that will guarantee correct results but does not rule out non-termination. A selection of the axioms and rules are shown in Table 1. [1][2]

Axiom of Assignment	$F[x/e]\ \{x := e\}\ F$
Rules of Consequence	If $G\ \{C\}\ F$ and $F \Rightarrow P$ then $G\ \{C\}\ P$
	If $G\ \{C\}\ F$ and $P \Rightarrow G$ then $P\ \{C\}\ F$
Rule of Composition	If $G\ \{C_1\}\ F_1$ and $F_1\ \{C_2\}\ F$ then $G\ \{C_1; C_2\}\ F$
Rule of Iteration	If $F \wedge (G\ \{C\}\ F)$ then $F\ \{\text{while } B \text{ do } C\}\ \neg B \wedge F$

Table 1: Valid Hoare Triples

$\{F[x/e]\}$ is obtained by substituting occurrences of x by e.

Semantically, a Hoare Triple $G\ \{C\}\ F$ is said to be valid for (partial) correctness, if the execution of the program C with an initial state satisfying the precondition G leads to a final state that satisfies the postcondition F, provided that the program terminates. Figure 1 illustrates a valid Hoare Triple, Σ represents the set of all states, the section denoted with G includes the states that satisfy the predicate G. The arrow from left to write denotes the execution of the program C.

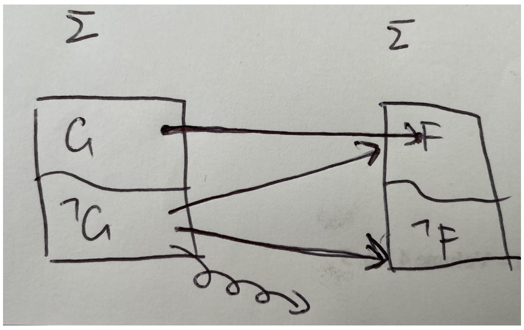

Figure 1: Valid Hoare Triple (Deterministic)

1 We omit the symbol $\vdash$ in front of a Hoare Triple, which denotes "valid/provable", for better readability.

2 Nondeterminism was not considered in the original paper, so we treat the programs here as deterministic.

TODO: Digitize image, also add program C.

The proof system built by Hoare's rules is sound, expressive, yet incomplete. [1] A sensible advancement would be to find the necessary and sufficient preconditions that grant us the post-properties, i.e. to eliminate the arrows from $\neg G$ to F in Figure 1.

2.2 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 [3] form of nGCL that is similar to modern pseudo-code:

$$\begin{aligned} C \ ::= \ & x := e \ \mid \ C; C \ \mid \ \{C\}\Box\{C\} \ \mid \ \mathtt{if}\ (\varphi)\ \{C\}\ \mathtt{else}\ \{C\} \ \mid \ \mathtt{while}\ (\varphi)\ \{C\} \\ & \mid \mathtt{skip} \mid \mathtt{diverge} \end{aligned}$$

2.3 WEAKEST PRECONDITION

We define the weakest precondition transformer structurally in lambda-calculus style[4] as follows:

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

Table 2: The Weakest Precondition Transformer

$F[x/e]$ is F where every occurrence of x is syntactically replaced by e.
$\mathrm{lfp}\ X.f$ is the least fixed point of function f with variable X.
Let

$$\Phi(X) := (\neg\varphi \wedge F) \vee (\varphi \wedge wp.C'.X)$$

be the characteristic function.

3 Specifically, $\mathtt{if}\ (\varphi)\ \{C_1\}\ \mathtt{else}\ \{C_2\}$ is equivalent to $\mathtt{if}\ \varphi \to C_1\ [\!]\ \neg\varphi \to C_2\ \mathtt{fi}$ in Dijkstra's original style[2]; $\{C_1\}\Box\{C_2\}$ is equivalent to $\mathtt{if}\ \mathtt{true} \to C_1\ [\!]\ \mathtt{true} \to C_2\ \mathtt{fi}$.

4 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 type of wp.

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 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, then $[\![C]\!].\sigma$ is a set of a single state, either a final state σ' or $\perp$, if the execution does not terminate. If C is non-deterministic, $[\![C]\!].\sigma$ can be a set with multiple elements, since multiple final states can be possible.

The weakest precondition wp.C.F is then **TODO:** Justify all the definitions except while.

TODO: Explain least point iteration from bottom.

2.4 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 $\perp$ to calculate the least fixed point of the characteristic function: $\Phi^k(\perp)$. Thus by finding the least fixed point, we've found a k that satisfies (1).

2.5 WEAKEST LIBERAL PRECONDITION

We define the weakest liberal precondition transformer in Table 3.

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

Table 3: 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

[1] Krzysztof R. Apt. "Ten Years of Hoare's Logic: A Survey—Part I." In: *ACM Trans. Program. Lang. Syst.* 3.4 (1981), 431–483. ISSN: 0164-0925. DOI: 10.1145/357146.357150. URL: https://doi.org/10.1145/357146.357150.

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

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

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 May 3, 2023 (`classicthesis` version 0.1).