

# NECESSARY LIBERAL PRECONDITIONS: A PROOF SYSTEM

MASTER'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**

**MASTER'S THESIS IN INFORMATICS**

**ANRAN WANG, B.SC.**  
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS  
TECHNICAL UNIVERSITY OF MUNICH

Examiner: Prof. Jan Křetínský  
Supervisors: 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

---

This is where the abstract goes.

## ZUSAMMENFASSUNG

---

Kurze Zusammenfassung des Inhaltes in deutscher Sprache...

# CONTENTS

---

<b>I</b>	<b>HOARE TRIPLES, WEAKEST PRECONDITIONS, WEAKEST LIBERAL PRECONDITIONS</b>	<b>1</b>
1	BACKGROUND	2
2	PRELIMINARIES	3
2.1	Lloyd-Hoare Logic	3
2.2	Guarded Command Language	4
2.3	Weakest Precondition	5
2.3.1	The Deterministic Case	5
2.3.2	The Nondeterministic Case	6
2.4	Defining Loops	6
2.5	Weakest Liberal Precondition	6
<b>II</b>	<b>NECESSARY LIBERAL PRECONDITIONS</b>	<b>7</b>
3	A PROOF SYSTEM	8
3.1	A Proof System	8
4	CONCLUSIONS	9
4.1	Conclusions	9
4.2	Future Work	9
<b>III</b>	<b>APPENDIX</b>	<b>10</b>
	BIBLIOGRAPHY	11

LIST OF FIGURES

---

Figure 1	Valid Hoare Triple (Deterministic)	4
----------	------------------------------------	---

LIST OF TABLES

---

Table 1	Valid Hoare Triples	3
Table 2	The Weakest Precondition Transformer (Deterministic Programs) [4]	5
Table 3	The Weakest Liberal Precondition Transformer	6

LISTINGS

---

ACRONYMS

---



## Part I

### HOARE TRIPLES, WEAKEST PRECONDITIONS, WEAKEST LIBERAL PRECONDITIONS

Some text about this part.

## BACKGROUND

---

**TODO:** Make first letter big?

**TODO:** Decide on all the colors in the end.

**TODO:** Rewrite; add chapter contents.

## PRELIMINARIES

### 2.1 LLOYD-HOARE LOGIC

**TODO:** A history lesson, rewrite to include LLoyd. See [4] P.27.

In 1969, C.A.R. Hoare wrote *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 as [Hoare Logic](#). 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](#).<sup>12</sup>

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

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.

The definition indeed corresponds to this intended semantics. (Formal soundness proofs can be found in Krzysztof R. Apt's survey [1] in 1981. ) As an example, consider the rule of composition: if the execution of program  $C_1$  changes the state from  $G$  to  $F_1$ , and  $C_2$  changes the state from  $F_1$  to  $F$ , then executing them consecutively should bring the program state from  $G$  to  $F$ , with the intermediate state  $F_1$ .

The missing guarantee of termination can be seen in the rule of iteration: consider the example **TODO: Add example here**.

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

<sup>2</sup> Nondeterminism was not considered in the original paper, so we treat the programs here as deterministic. With deterministic programs, one initial state corresponds to one final state, and by non-termination we assign a final state  $\perp$ .

**TODO:** Think about whether to add liberally deterministic (Hesselink 1992, *Programs, Recursion and Unbounded Choice*).

Figure 1 illustrates a valid Hoare Triple,  $\Sigma$  represents the set of all states, the section denoted with  $G$  includes the states that satisfy the predicate  $G$ . The arrow from left to right denotes the execution of the program  $C$ .

**TODO:** In case I change color for  $\setminus \text{mathl}$ , I should change the color for hoare triple GCF.

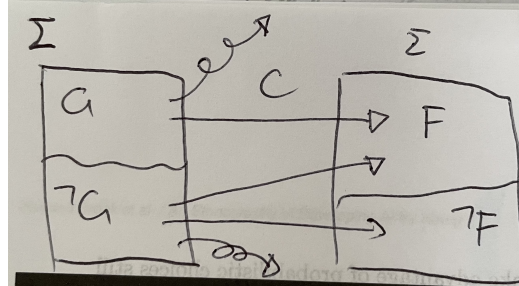


Figure 1: Valid Hoare Triple (Deterministic)

**TODO:** Digitize image.

Hoare Logic 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, and to be able to prove termination, i.e. to eliminate the arrows from  $G$  to the abyss<sup>3</sup>, which is what Edsger W Dijkstra accomplished with his **weakest precondition** transformer in 1975 [2], among other things.

## 2.2 GUARDED COMMAND LANGUAGE

From now on we will use Dijkstra's (non-deterministic) **guarded command language (GCL)** [2] to represent programs and to include non-determinism (starting from Section 2.3.2). For better understanding, we use an equivalent<sup>4</sup> form of GCL that is similar to modern pseudo-code:

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

The **nondeterministic choice**  $\{C_1\} \square \{C_2\}$  chooses from two programs randomly. It is however not **probabilistic**, where we know the probabilistic distribution of the outcome of the choice. With the nondeterministic choice, we have no such knowledge.

<sup>3</sup> Adding termination proof is also done by Zohar Manna and Amir Pnueli in 1974 [5], where they introduced what we call a **loop variant**, a value that decreases with each iteration. The name is in contrast to **loop invariant**, concretely the  $F$  in Rule of Iteration, which is constant before and after the loop.

<sup>4</sup> Specifically,  $\text{if } (\varphi) \{C_1\} \text{ else } \{C_2\}$  is equivalent to  $\text{if } \varphi \rightarrow C_1 \square \neg \varphi \rightarrow C_2 \text{ fi}$  in Dijkstra's original style[2];  $\{C_1\} \square \{C_2\}$  is equivalent to  $\text{if true} \rightarrow C_1 \square \text{true} \rightarrow C_2 \text{ fi}$ .

When `skip` is executed, the program state does not change and the consecutive part is executed. When `diverge` is executed, the program goes to state  $\perp$  to denote non-termination, and the execution stops.

## 2.3 WEAKEST PRECONDITION

### 2.3.1 The Deterministic Case

To better relate Hoare Triples and Dijkstra's weakest precondition transformer, we first ignore nondeterminism. **TODO: Call it wp-? But the calculus is the same.**

We define the **weakest precondition** transformer structurally in lambda-calculus style<sup>5</sup> as in [Table 2](#):

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

Table 2: The Weakest Precondition Transformer (Deterministic Programs) [4]

$F[x/e]$  is  $F$  where every occurrence of  $x$  is syntactically replaced by  $e$ .

$\text{lfp } X.f$  is the least fixed point of function  $f$  with variable  $X$ .

Let

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

be the characteristic function.

To justify this definition, we must first clarify the intended semantics/meaning of the wp-transformer. Let  $\llbracket C \rrbracket$  denote the **execution** of program  $C$ ,  $\llbracket C \rrbracket.\sigma$  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  $\Sigma = \{\sigma \mid \sigma : \text{Vars} \rightarrow \text{Vals}\}$ .)

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

The weakest precondition  $\text{wp}.C.F$  is then **TODO: Justify all the definitions except while.**

**TODO: Explain least point iteration from bottom.**

<sup>5</sup> For example,  $\text{wp}.C.F$  can be seen as  $\text{wp}(C, F)$  in "typical" style, where  $\text{wp}$  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  $\text{wp}.C.F.\sigma$  where  $\sigma$  means the initial state. Here  $\text{wp}$  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  $\text{wp}$ .

## 2.3.2 The Nondeterministic Case

## 2.4 DEFINING LOOPS

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

Let

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

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

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

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:

$$\text{wp}.\text{WHILE}.F = (\exists k \geq 0 : H_k(F)) \quad (1)$$

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.

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

Table 3: The Weakest Liberal Precondition Transformer

## Part II

### NECESSARY LIBERAL PRECONDITIONS

Some text about this part.

## A PROOF SYSTEM

---

### 3.1 A PROOF SYSTEM

In this section we study the necessary liberal precondition:

$$\text{wlp.C.F} \implies G$$



## CONCLUSIONS

---

### 4.1 CONCLUSIONS

### 4.2 FUTURE WORK

## Part III

### APPENDIX

## BIBLIOGRAPHY

---

- [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](https://doi.org/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.
- [4] Benjamin Lucien Kaminski. “Advanced weakest precondition calculi for probabilistic programs.” PhD thesis. RWTH Aachen University, 2019.
- [5] Zohar Manna and Amir Pnueli. “Axiomatic approach to total correctness of programs.” In: *Acta Informatica* 3 (1974), pp. 243–263.

## 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 L<sup>A</sup>T<sub>E</sub>X and L<sup>y</sup>X:

<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 4, 2023 (classicthesis version 0.1).