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
Supervisors:	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

This is where the abstract goes.

ZUSAMMENFASSUNG

Kurze Zusammenfassung des Inhaltes in deutscher Sprache…

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

Part I

HOARE TRIPLES, WEAKEST PRECONDITIONS, WEAKEST LIBERAL PRECONDITIONS

Some text about this part.

1 BACKGROUND

In 1739, the Scottish philosopher David Hume questioned why we know that the sun will rise tomorrow, "tho' 'tis plain we have no further assurance of these facts, than what experience affords us" [7]. Hume's question about causality is daunting, yet most of us are not in crisis because we doubt if the sun rises tomorrow. The reason is probably that we believe in physics, astrology, and the rules and formulas that assure us the universe works in a certain way, hence the sun rises tomorrow. It is exactly the rules and formulas this thesis attempts to investigate, in the realm of computer programs, with which we are certain that the equivalent version of the sun in a program will rise tomorrow.

Computer programs are ubiquitous in almost every aspect of human life. We want them to solve our problem efficiently, and correctly. Imagine being driven by an autonomous car. It is desirable that it delivers us to the correct destination, and never get stuck driving around the same block without making progress. Delivering the correct result and stopping eventually is called total correctness. Once we know that a program is totally correct, then we are sure that the sun rises tomorrow.

To know "for sure", we could verify programs using formal methods. One famous method is Hoare triples [6]. A Hoare Triple contains three parts: a precondition, a program, and a postcondition. They are written as such: G $\{C\}$ F. It states that if the system starts 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. Hoare triples are elegant in that once we have appropriate preconditions, we can follow their reference rules on sequential programs with ease. But with Hoare triples in their original form, we know the program is correct, but we are not sure of its termination. This is called partial correctness.

To prove a program totally correct, Dijkstra presented the weakest precondition transformer [2] (wp): starting with a postcondition, it works backwards and calculates what the weakest precondition is that guarantees both correctness and termination. In Hoare triples, the precondition is a sufficient condition for the program to be correct in that the final state will satisfy the desired postcondition, while with wp we obtain a necessary and sufficient precondition.

Since then, a plethora of research projects blossomed and yielded fruitful results. This thesis aims to follow the steps of the predecessors and investigate the weakest liberal precondition transformer [4] (wlp), which gives preconditions that are necessary and sufficient so that the program either terminates correctly or never terminates, proving partial correctness.

We first introduce Hoare triples, the wp transformer, and the wlp transformer using the Guarded Command Language [2] to present programs in Chapter 2. We also explain their connections and differences.

Then we proceed to Chapter 3 **[fill in content of this chapter]** [1]

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

2 PRELIMINARIES

2.1 HOARE LOGIC

Since the beginning of the 1960s, scholars have been researching the establishment of mathematics in computation [5, 10] to have a formal understanding and reasoning of programs. One of the most known methods is Hoare logic.

In 1969, C.A.R. Hoare wrote *An Axiomatic Basis for Computer Programming* [6] to explore the logic of computer programs using axioms and inference rules to prove the properties of programs. He introduced sufficient preconditions that guarantee correct results but do not rule out non-termination. A selection of the axioms and rules are shown in Table 1. [1][2]

[3]

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

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 $(F \wedge B)\ \{C\}\ F$ then $F\ \{\text{while } B \text{ do } C\}\ \neg B \wedge F$

Table 1: Valid Hoare Triple

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 definitions in Table 1 indeed correspond 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 triple $x \leqslant 2\ \{\text{while } x \leqslant 1 \text{ do } x := x * 2\}\ 1 < x \leqslant 2$, it is provable in Hoare logic with the following proof tree. However, this while-loop will not terminate in case $x \leqslant 0$ in the initial state.

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

2 Non-determinism 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 in case of non-termination we assign a final state $\perp$.

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

$$\dfrac{\dfrac{}{x \leqslant 1\ \{x := x * 2\}\ x \leqslant 2}\ \textbf{Axiom of Assignment}}{x \leqslant 2\ \{\text{while } x \leqslant 1 \text{ do } x := x * 2\}\ 1 < x \leqslant 2}\ \textbf{Rule of Iteration}$$

Using style taken from Benjamin L. Kaminski's dissertation [8], 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 right denotes the execution of the program C.

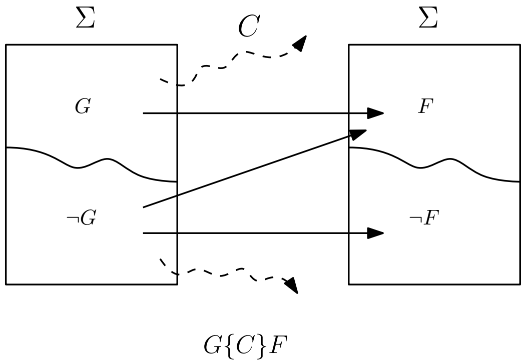

Figure 1: Valid Hoare Triple (Deterministic)

A sensible advancement of Hoare logic would be to also prove termination, i.e. to eliminate the arrows from G to the abyss. Supplementing Hoare logic with a termination proof is done by Zohar Manna and Amir Pnueli in 1974 [9], 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** in Table 1, which is constant before and after the loop.

Another 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, 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 2.3.3). For better readability, we use an equivalent[4] form of GCL that is similar to modern pseudo-code as shown in Table 2.

The assignment, sequential composition, conditional choice, while-loop commands conform to their usual meaning. The non-deterministic choice $\{C_1\}\Box\{C_2\}$ chooses from two programs randomly. It is however not probabilistic, meaning we do not know the probabilistic distribution of the outcome of the choice.

When skip is executed, the program state does not change and the consecutive part is executed. When diverge is executed, the program goes to a state symbolizing non-termination, and the execution stops.

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

C	$::=$	$x := e$	$\mid C; C$	$\mid \{C\}\Box\{C\}$
		assignment	sequential composition	non-deterministic choice
		$\mid$ if (φ) $\{C\}$ else $\{C\}$	$\mid$ while (φ) $\{C\}$	$\mid$ skip $\quad\mid$ diverge
		conditional choice	while-loop	

Table 2: Guarded Command Language

In our representation of GCL, non-determinism is explicitly constructed via the infix operator $\Box$, whereas in its original definition, non-determinism occurs when the guards within the if and while commands are not mutually exclusive [4]. Additionally, the if statement in Dijkstra's GCL is equivalent to divergence in case non of its guards are true, but in our version this can no longer happen because of the Law of Excluded Middle: the predicate φ must be either true or false, so either the "then" branch or the "else" branch is activated. Consequently, non-termination can only originate from either the diverge or the while command.

2.3 WEAKEST PRECONDITIONS

2.3.1 *The Deterministic Case*

To better relate Hoare triples and Dijkstra's weakest precondition transformer, we first focus on deterministic programs. The goal is to find the necessary and sufficient precondition such that the program is guaranteed to terminate in a state that satisfies the postcondition. Figure 2 shows it graphically alongside the figure for valid Hoare triples. We can see that on the right side, the arrows from G to non-termination and from $\neg G$ to F are absent.

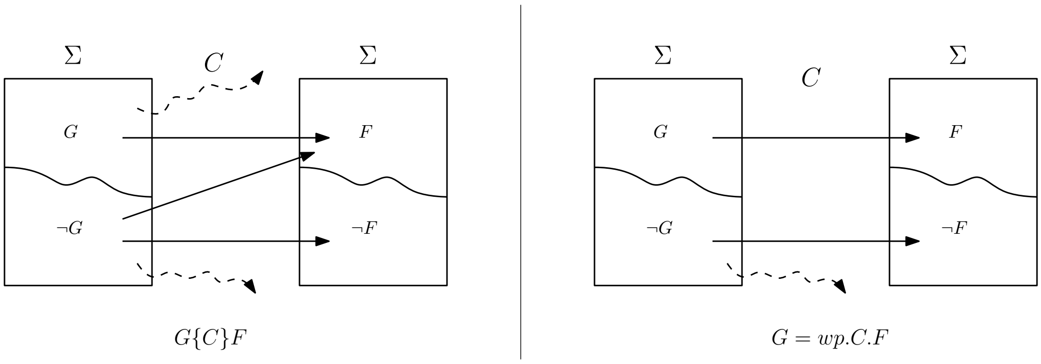

Figure 2: Valid Hoare Triple vs. Weakest Precondition (Deterministic)

We define the weakest precondition transformer inductively over the program structure in lambda-calculus style[5] as in Table 3:

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

5 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 to $wp.C.F.\sigma$ where σ means the initial state. Here wp is treated

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

Table 3: The Weakest Precondition Transformer for Deterministic Programs [8]

$\mathtt{lfp}\ X.f$ 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, then wp for while-loop can be defined as:

$$wp.(\mathtt{while}(\varphi)\{C'\}).F = \mathtt{lfp}\ X.\Phi(X)$$

Most of the definitions in Table 3 are intuitive and correspond to their counterparts in Hoare logic, while those for $\mathtt{diverge}$ and $\mathtt{while}$ deserve special attention. Since wp aims for total correctness, a program starting in an initial state satisfying the precondition $wp.\mathtt{diverge}.F$ should terminate in a final state satisfying the postcondition F. Because $\mathtt{diverge}$ does not terminate, there is no such precondition and wp for $\mathtt{diverge}$ should be $\mathtt{false}$.

The definition for the while-loop [8] is trickier, but we can verify its correctness by recalling Dijkstra's original definition in the following section.

2.3.2 *Defining Loops*

In Dijkstra's original paper [2], he defined wp for while-loops based on its (intended) semantics, i.e. the precondition that guarantees loop termination with the required postcondition within a certain number of iterations.

Let

$$WHILE = \mathtt{while}(\varphi)\{C'\} \qquad \text{and} \qquad IF = \mathtt{if}\ (\varphi)\{C'\}\ \mathtt{else}\ \{\mathtt{diverge}\}.$$

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

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

IF is defined in such way that $wp.IF.X$ is the weakest precondition that makes sure the guard of IF discharges and C' is executed once, leaving the program in a state satisfying X. As a result, $H_k(F)$ corresponds to the weakest precondition such that the program terminates in a final state satisfying F after at most k iterations.

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.

Then by definition:

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

The definition in Table 3, however, uses the least fixed point of the characteristic function that is not obvious. We understand the use of fixed point in two ways. First, a precondition G being a fixed point of the characteristic function $G = \Phi(G) = (\neg\varphi \wedge F) \vee (\varphi \wedge wp.C'.G)$ means that under control of G, termination is possible (left side of the disjunction) and repeated execution of C' is possible(right side of the disjunction), since G is invariant before and after the execution of C'. Second, if we were to believe that the semantics of WHILE should be equivalent to the semantics of $\mathtt{if}(\varphi)\{C;\mathit{WHILE}\}\mathit{else}\{\mathit{skip}\}$, we can derive the need for fixed point:

$$\begin{aligned}
wp.\mathit{WHILE}.F &\overset{!}{=} wp.(\mathtt{if}(\varphi)\{C;\mathit{WHILE}\}\mathit{else}\{\mathit{skip}\}.F) \\
&\overset{!}{=} \varphi \wedge wp.(C;\mathit{WHILE}).F \vee \neg\varphi \wedge wp.\mathit{skip}.F \\
&\overset{!}{=} \varphi \wedge wp.C.(wp.\mathit{WHILE}.F) \vee \neg\varphi \wedge F \\
&\overset{!}{=} \Phi(wp.\mathit{WHILE}.F)
\end{aligned}$$

The question then arises: can we define wp with any fixed point? The answer is no and we show it by verifying that the definition in Table 3 coincides with Dijkstra's definition at the beginning of this chapter.[6] We borrow a theorem from domain theory that yields a computation for least fixed points, provided they exist:

Theorem 1. ***[Insert theorem]***

Coincidentally, $H_k(F)$ is the $(k+1)$–th iteration of the characteristic function Φ from the bottom element, denoted by $\Phi^{k+1}(\mathit{false})$. For all predicates F and all programs C':

Lemma 1. $\forall k \geqslant 0 : H_k(F) = \Phi^{k+1}(\mathit{false})$

Proof. Proof by induction.

BASE CASE:

$$\begin{aligned}
\Phi(\mathit{false}) &= (\neg\varphi \wedge F) \vee (\varphi \wedge wp.C'.\mathit{false}) \\
&= (\neg\varphi \wedge F) \vee (\varphi \wedge \mathit{false}) && \mid (*) \\
&= \neg\varphi \wedge F && \mid \text{predicate calculus} \\
&= H_0(F)
\end{aligned}$$

The equation (*) is supported by the Law of Excluded Miracle [3, p.18]: for all programs C, $wp.C.\mathit{false} = \mathit{false}$. It states that it is impossible for a program to terminate in a state satisfying no postcondition.

6 In fact, Dijkstra and Scholten[4] later also gave definitions for wp and wlp in an equivalent form of least and greatest fixed points, they called it "strongest" and "weakest solution". They also proved that it is necessary to use the extreme solutions.

STEP CASE:

$$
\begin{aligned}
H_{k+1}(F) &= wp.IF.H_k(F) \vee H_0(F) \\
&= (\varphi \wedge wp.C'.H_k(F)) \vee (\neg\varphi \wedge wp.diverge.H_k(F)) \vee H_0(F) \\
&\qquad\qquad\qquad\qquad \mid \text{unfold IF; definition of wp} \\
&= (\varphi \wedge wp.C'.H_k(F)) \vee (\neg\varphi \wedge \mathit{false}) \vee H_0(F) \\
&\qquad\qquad\qquad\qquad \mid \text{definition of wp} \\
&= (\varphi \wedge wp.C'.\Phi^{k+1}(\mathit{false})) \vee H_0(F) \\
&\qquad\qquad\qquad\qquad \mid \text{induction hypothesis} \\
&= (\varphi \wedge wp.C'.\Phi^{k+1}(\mathit{false})) \vee (\neg\varphi \wedge F) \\
&= \Phi^{k+2}(\mathit{false})
\end{aligned}
$$

□

Thus by identifying the least fixed point, we find a k that satisfies Equation 1. The advantage of using least fixed point to define wp is that there are heuristics to find it, whereas Equation 1 excels at giving intuitions for the preconditions that guarantee loop termination. Essentially, they express the same predicate, i.e. the "weakest" precondition for while-loops which is unique. Consequently, it means that we can not use other fixed points to define $wp.WHILE$, which are weaker than the least fixed point. For the same reason, we will see that greatest fixed point is necessary to define weakest liberal precondition.

2.3.3 *The Non-deterministic Case: Angelic vs. Demonic*

Now we bring the non-deterministic choice back into the picture and add its wp to Table 4. Here we assume a setting with angelic non-determinism, where we assume that whenever non-determinism occurs, it will be resolved in our favor. This results in the weakest precondition for our non-deterministic choice being a disjunction of the wp for its subprograms. We are hopeful that a precondition satisfying the wp of one of the subprograms can also lead to termination in our desired postcondition. This is a design choice that is different from Dijkstra's [2], where the wp for non-deterministic choice is a conjunction, hinting at a demonic setting. Both choices are justifiable, we choose to follow Zhang and Kaminski's work, favoring the resulting Galois connection between the weakest (liberal) precondition transformers and the strongest (liberal) postcondition transformers [11].

The left side of Figure 3 shows wp with non-deterministic programs. Each arrow from left to right shows a possible execution of program C. The effects of demonic and angelic non-determinism is highlighted in green. A condition under whose control the required postcondition is reachable but not guaranteed is considered as a valid precondition in an angelic setting (left half of Figure 3), but not in a demonic setting (right half of Figure 3).

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

Table 4: The Weakest (Liberal) Precondition Transformer for Non-deterministic Programs [8]

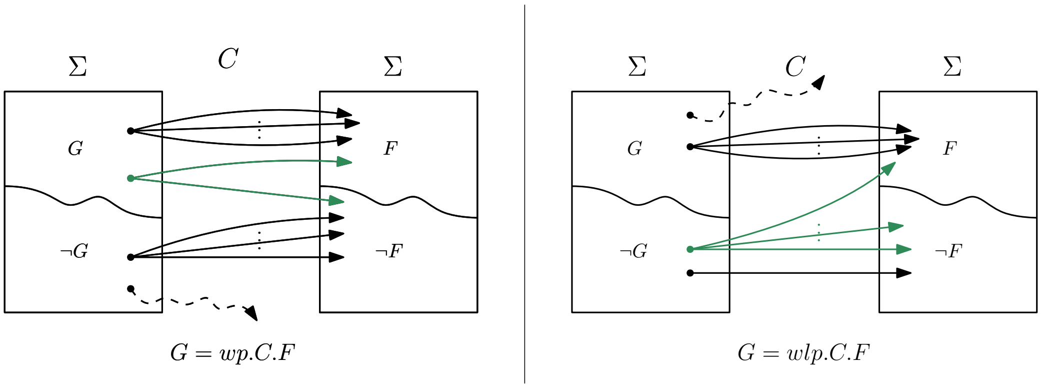

Figure 3: Weakest Precondition (Angelic Non-determinism), Weakest Liberal Precondition (Demonic Non-determinism)

2.4 WEAKEST LIBERAL PRECONDITIONS

While the wp-transformer excludes non-termination, the wlp-transformer takes a more liberal approach. The weakest precondition delivers a precondition so that the program terminates and a state satisfying the postcondition is reachable. The weakest liberal precondition, however, delivers a precondition so that the program either terminates satisfying the postcondition, or diverges. The postcondition in the wlp setting is guaranteed upon termination, because we regard the non-deterministic choice as demonic, again favoring to establish a Galois connection [11].

We define the weakest liberal precondition transformer in Table 4. A graphical representation can be found on the right side of Figure 3.

As preluded earlier, greatest fixed points are used to define wlp for while-loops. It is an easy choice, since wlp is semantically the weakest liberal precondition, and $wlp.WHILE.F$ should be a fixed point of its characteristic function, similar to Section 2.3.2.

2.5 PROPERTIES OF WP AND WLP

wp and wlp are each other's conjugate:

$$wp.C.F = \neg wlp.C.\neg F$$

[TO BE CONTINUED]

Part II

NECESSARY LIBERAL PRECONDITIONS

Some text about this part.

3 A PROOF SYSTEM

3.1 A PROOF SYSTEM

In this section we study the necessary liberal precondition:

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

3.2 SKETCHES

Note for readers: In this section are informal sketches of my thoughts.

POSSIBLE WAYS TO APPROACH THIS

1. $wlp.C.F \implies G$ but restrict G by requiring that the postcondition F is always reachable. Effectively this transfers wlp in demonic setting to wlp in angelic setting, the postcondition being "guaranteed" changes into "reachable". Remember: both Dijkstra's wp and wlp are in demonic setting, and they are related by $wp_d.C.X = wp_d.C.true \wedge wlp_d.C.X$. Explore the relationship between $wp_angelic$, $wp_demonic$, $wlp_angelic$, $wlp_demonic$? Then the results in [11] and [4] can be linked. But then who uses angelic wlp? Also, does it even make sense to investigate this, since they are both extrema, specially in a quantitative setting.

2.

$$wlp.C.F \implies G \equiv \neg G \implies \neg wlp.C.F \equiv \neg G \implies wp.C.\neg F$$

This corresponds to total correctness, but negatively. $\neg G\ \{C\}\ \neg F$ would be a valid Hoare Triple. But negative is not pretty.

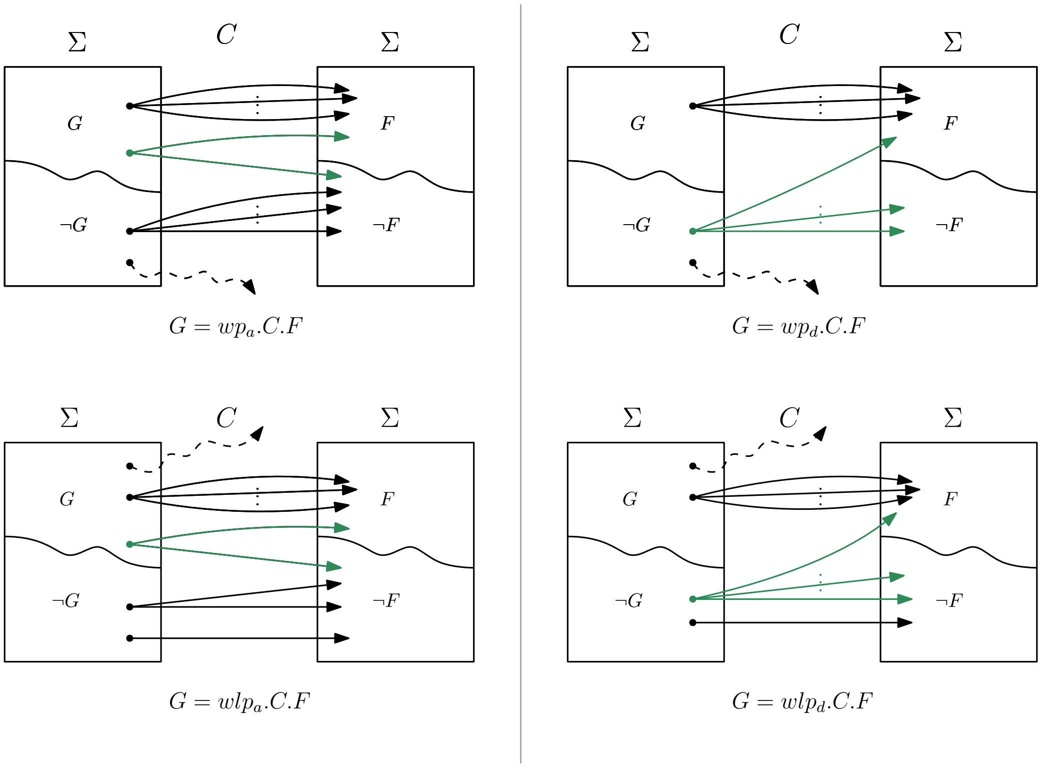

Figure 4: Angelic and Demonic Nondeterminism

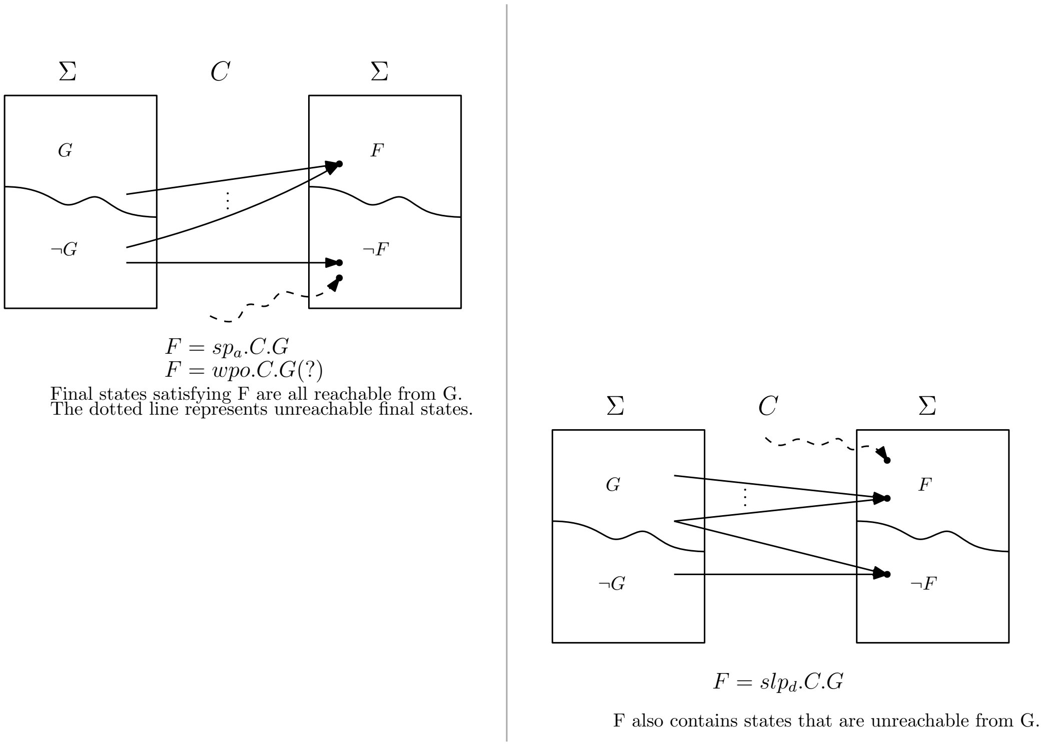

Figure 5: sp and slp

4 CONCLUSIONS

4.1 CONCLUSIONS

4.2 FUTURE WORK

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 (Oct. 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] Edsger W Dijkstra, Edsger Wybe Dijkstra, Edsger Wybe Dijkstra, and Edsger Wybe Dijkstra. *A discipline of programming*. Vol. 613924118. prentice-hall Englewood Cliffs, 1976.

[4] 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.

[5] Robert W Floyd. “Assigning meanings to programs.” In: *Program Verification: Fundamental Issues in Computer Science* (1993), pp. 65–81.

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

[7] David Hume. *A treatise of human nature*. Clarendon Press, 1896.

[8] Benjamin Lucien Kaminski. “Advanced weakest precondition calculi for probabilistic programs.” PhD thesis. RWTH Aachen University, 2019.

[9] Zohar Manna and Amir Pnueli. “Axiomatic approach to total correctness of programs.” In: *Acta Informatica* 3 (1974), pp. 243–263.

[10] John McCarthy. “Towards a mathematical science of computation.” In: *Program Verification: Fundamental Issues in Computer Science* (1993), pp. 35–56.

[11] 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 September 22, 2023 (`classicthesis` version 0.1).