Partial Incorrectness Logic

Lena Verscht
RWTH Aachen
Germany
Saarland University
Germany

Ānrán Wáng
Saarland University
Germany

Benjamin Lucien Kaminski
Saarland University
Germany
University College London
United Kingdom

1 INTRODUCTION

Reasoning about program correctness has been a central topic in static analysis for many years, with *Hoare logic* (HL) playing an important role [1, 6]. The key notions in HL are *partial* and *total correctness*. Both require that program executions starting in a specified set of initial states (the *precondition*) reach a designated set of final states (the *postcondition*). Partial correctness is more lenient in that it does not require termination, effectively deeming divergence acceptable.

We explore *partial incorrectness logic* [10], which stands in relation to O'Hearn's "total" incorrectness logic [9] as partial correctness does to total correctness: Partial correctness allows divergence, partial *incorrectness* allows *unreachability*. While the duality between divergence and unreachability may not be immediately apparent, we explore this relationship further.

Our chosen formalism is *predicate transformers* à la Dijkstra [3]. We focus here on *deterministic* and *reversible* programs, though the discussion extends to nondeterministic and irreversible computations, both of which introduce additional nondeterminism that must be addressed.

2 CORRECTNESS AND TERMINATION

Hoare logic [6] is concerned with triples of the form $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$, where b is a precondition on initial states, p is a program, and c is a postcondition on final states. $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is said to be *valid for total correctness* if all computations of p started in initial states satisfying precondition b (1) terminate, and (2) do so in some state satisfying the postcondition c.

As is well known, proving termination is undecidable. This is motivation to define a more lenient notion of correctness, *partial correctness*, which requires only criterion (2) above. In other words: *If* p terminates, it has to do so in c.

2.1 Weakest (Liberal) Pre

We use *predicate transformers* due to Dijkstra as the formal framework for expressing program specifications [4]. The *weakest precondition* $\mathsf{wp}\,[\![p]\!](c)$ of a postcondition c with respect to a program p contains *all* states from which p always terminates in c. Notably, this is the weakest predicate b such that $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for *total* correctness. Dually, the *weakest liberal precondition* $\mathsf{wlp}[\![p]\!](c)$ of c with respect to p contains *all* states from which p either diverges or terminates in c. This is the weakest predicate b such that $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for *partial* correctness.

Computing weakest (liberal) preconditions can be done inductively on the program structure. The most intricate part of this are loops, which require reasoning about fixed points. The weakest (liberal) preconditions of while loops are defined as fixed points of a characteristic function Φ which, intuitively, captures one execution of the loop body:

$$\mathsf{wp}\,[\![\texttt{while}\,(\,\varphi\,)\,\{\,C'\,\}]\!](F) \;=\; \mu\,\Phi$$
$$\text{and}\qquad \mathsf{wlp}[\![\texttt{while}\,(\,\varphi\,)\,\{\,C'\,\}]\!](F) \;=\; \nu\,\Phi\,,$$

where $\mu\,\Phi$ denotes the least and $\nu\,\Phi$ the greatest fixed point of Φ [7]. For our intents, the relevant aspect here is that weakest preconditions are *least* fixed points while weakest liberal preconditions are *greatest* fixed points.

2.2 Total and Partial Correctness

Above, we have hinted at the close connection between total (partial) correctness and weakest (liberal) preconditions, which comes as an *underapproximation*:

$$\langle\, b \,\rangle\ p\ \langle\, c \,\rangle \text{ is valid for } \left|\begin{array}{ll} \text{total correctness} & \text{iff}\quad b \subseteq \mathsf{wp}\,[\![p]\!](c) \\ \text{partial correctness} & \text{iff}\quad b \subseteq \mathsf{wlp}[\![p]\!](c) \end{array}\right.$$

For proving total correctness of a loop, we thus have to prove a lower bound on a *least* fixed point – for partial correctness, we also have to prove a lower bound, but on a *greatest* fixed point. The latter can be conveniently proven using the following theorem:

Lemma 2.1 (Park Induction). *Let $(X, \preceq)$ be a complete partial order and $f : X \to X$ be monotonic. Then for all $I \in X$,*

$$I \;\preceq\; f(I) \quad \text{implies} \quad I \;\preceq\; \nu f\,.$$

If we have a suitable candidate or *invariant* I, we can thus prove *lower* bounds on *greatest* fixed points by just one application of f. For *lower* bounds on *least* fixed points, there is no such simple rule. The consequence for proofs of correctness is that we can prove partial correctness easily, while total correctness requires more intricate arguments [5].

2.3 Termination

Given a proof of partial correctness, we prove total correctness by additionally proving termination. In terms of weakest preconditions, this means

$$\underbrace{b \subseteq \mathsf{wp}\,[\![p]\!](c)}_{\text{total correctness}} \quad\Longleftarrow\quad \underbrace{b \subseteq \mathsf{wlp}[\![p]\!](c)}_{\text{partial correctness}} \;\wedge\; \underbrace{b \subseteq \mathsf{wp}\,[\![p]\!](\mathsf{true})}_{\text{termination}}\,.$$

Note that we in fact do not require *universal* termination, i.e. termination from *all* initial states, but only from the states of interest (those satisfying b).

There are several reasons why to separate these concerns: In the spirit of divide and conquer, we can split a problem into smaller subproblems. Proving partial correctness is indeed conceptually easier than proving total correctness, as we have seen in Section 2.2. Additionally, proving termination is a subfield on its own, meaning we can profit from well developed techniques and algorithms.

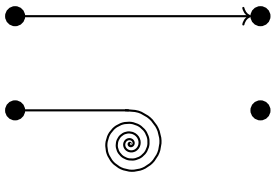

Figure 1: Termination versus reachability.

Proving termination using variants. To prove universal terminationof a while loop $\texttt{while}\,(\,\varphi\,)\,\{\,p\,\}$, we can make use of *variants*: A variant is a function $v\colon \text{States} \to \mathbb{N}$ such that for all $n \in \mathbb{N}$, the triple $\langle\, \varphi \wedge v = n \,\rangle\ p\ \langle\, v < n \,\rangle$ is valid for total correctness. Intuitively, each program state is assigned a natural number that has to strictly decrease in every iteration. Since the natural numbers are well-founded, there cannot be infinitely many iterations. Therefore, if a variant exists, then the loop $\texttt{while}\,(\,\varphi\,)\,\{\,p\,\}$ terminates.

3 INCORRECTNESS AND REACHABILITY

Reasoning about program correctness is a well-established area of research. Recently, a dual concept has gained significant attention: program *incorrectness* [2, 9]. Intuitively, incorrectness is concerned with reachability of a set of final states assumed to represent unwanted behavior, or, *bugs*. A triple $[\,b\,]\ p\ [\,c\,]$ is valid for incorrectness if all states satisfying c are reachable via execution of program p from some initial state in b.

Similar to correctness, we can also characterize incorrectness as the underapproximation of a predicate transformer. This raises the question whether for incorrectness we can have a similar separation of concerns as with total correctness in the form of

$$\text{incorrectness} \;=\; \text{something} + \text{something else}\,.$$

We will see that we can answer this positively. For this, let us again start by looking at the required predicate transformers.

3.1 Strongest (Liberal) Post

The *strongest postcondition* $\mathsf{sp}\,[\![p]\!](b)$ of a program p and a precondition b contains all *final* states which computation starting from b can reach [4]. Strongest *liberal* postconditions on the other hand are much less investigated and were only recently introduced by Zhang and Kaminski [10]. Liberality here refers to permitting unreachability, in contrast to divergence in the correctness setting. This comes rather natural, because predicate transformers, as other approaches, take a relational view on computation: In the end, we are not so much interested in what happens within the program, instead, we abstract this to which initial states lead to (or *are related to*) which final states.

A simple illustration for this is provided in Figure 1. The program state space only spans two states. Starting in the first state the program terminates, from the second state the program diverges. We see that the second initial state from which computation diverges is related to no final state. Dually, the second final state is related to no initial state; it is *unreachable*. Therefore, the strongest liberal post $\mathsf{slp}[\![p]\!](b)$ contains the states which computation starting from b can reach, and additionally the *unreachable states*.

C	$\mathsf{slp}[\![p]\!](F)$
$\texttt{diverge}$	$\texttt{true}$
$x \coloneqq e$	$\forall \alpha : x \neq e[x/\alpha] \vee F[x/\alpha]$
$C_1 \fatsemi C_2$	$\mathsf{slp}[\![C_2]\!](\mathsf{slp}[\![C_1]\!](F))$
$\texttt{if}\,(\,\varphi\,)\,\{\,C_1\,\}\,\texttt{else}\,\{\,C_2\,\}$	$\mathsf{slp}[\![C_1]\!](\neg\varphi \vee F) \wedge \mathsf{slp}[\![C_2]\!](\varphi \vee F)$
$\texttt{while}\,(\,\varphi\,)\,\{\,C'\,\}$	$\varphi \vee \big(\nu Y.\, F \wedge \mathsf{slp}[\![C']\!](\neg\varphi \vee Y)\big)$

Table 1: The strongest liberal postcondition transformer [10]

Similar to weakest (liberal) preconditions, the strongest (liberal) postconditions can be computed using inductive rules. Those for the slp calculus with predicates are given in Table 1.

The loop definitions again invoke *least* and *greatest* fixed points of a characteristic function Φ:

$$\mathsf{sp}\,[\![\texttt{while}\,(\,\varphi\,)\,\{\,C'\,\}]\!](F) = \mu\,\Phi$$
$$\text{and} \qquad \mathsf{slp}[\![\texttt{while}\,(\,\varphi\,)\,\{\,C'\,\}]\!](F) = \nu\,\Phi.$$

Again, the liberal calculus corresponds to a *greatest* fixed point and the non-liberal to a *least* fixed point.

3.2 Total and Partial Incorrectness

Similar to total correctness, we can characterize *in*correctness as an underapproximation of the *non-liberal* strongest post. It is thus a natural choice to define a partial variant of incorrectness as an underapproximation of the *liberal* transformer, giving us:

$$[\,b\,]\ p\ [\,c\,] \text{ is valid for } \begin{array}{|ll} \text{“total” incorrectness} & \text{iff}\ \ c \subseteq \mathsf{sp}\,[\![p]\!](b) \\ \text{partial incorrectness} & \text{iff}\ \ c \subseteq \mathsf{slp}[\![p]\!](b) \end{array}$$

For partial incorrectness, c may contain unreachable states, but *if* a state satisfying c is reachable, then it was reachable from some initial state satisfying b.

Using Park induction (Lemma 2.1) again, we can prove partial incorrectness easily (upon finding a suitable invariant). Yet again, proving total incorrectness by finding a lower bound on the non-liberal sp is hard, which is why we divide this problem into subproblems in the next section.

3.3 Reachability

We can derive partial from total correctness by discharging termination. Dually, we obtain partial incorrectness from “total” incorrectness by discharging reachability. Reachability is expressible in terms of predicate transformers, very similar as to what we have seen for termination:

$$\underbrace{c \subseteq \mathsf{sp}\,[\![p]\!](b)}_{\text{“total” incorrectness}} \quad\Longleftarrow\quad \underbrace{c \subseteq \mathsf{slp}[\![p]\!](b)}_{\text{partial incorrectness}} \wedge \underbrace{c \subseteq \mathsf{sp}\,[\![p]\!](\text{true})}_{\text{reachability}}\,.$$

The strongest post of true contains all final states that are reachable. Again, we do not have to require reachability of *all* final states, but only of those in the postcondition c.

Proving reachability using variants. Unreachability is caused by more than loops: Even a simple assignment can already introduce unreachable states. Still, loops remain the most complex part of the reachability proof, as it requires the computation of a fixed

point. Dual to the variant method to prove termination, there are (backward) variant rules to prove reachability [2, 9]. A function $v\colon \text{States} \to \mathbb{N}$ is called a *backward variant* if for all $n \in \mathbb{N}$, the triple $[\, v < n \wedge \varphi \,]\ p\ [\, v = n \,]$ is valid for incorrectness. Intuitively, each state in which the variant has value n must be reachable from a state with value strictly less than n. Again, since the natural numbers are well-founded, there are only finitely many values strictly less than n. Therefore, if a backwards variant v exists, then there exists an $n \in \mathbb{N}$ such that the states where $\neg\varphi$ holds and $v = n$ are reachable when executing $\texttt{while}\,(\,\varphi\,)\,\{\,p\,\}$.

Note that this does not prove *universal* reachability, i.e. reachability of *all* final states, but only of those where the loop guard does not hold and where the variant takes on value n. It would be interesting to see if we can find rules that prove reachability of the states in some postcondition c only.

3.4 Partial Incorrectness: Applications

Beyond serving as a step toward classic ("total") incorrectness, partial incorrectness is an interesting property on its own. We now explore two scenarios where partial incorrectness proves useful.

Underapproximation triples. Consider the following program inspired by the beginning example of [9, Section 3.1]:

$$p \coloneqq \texttt{if}\ (\,x \text{ even}\,)\ \{\,y \coloneqq y + 1\,\}\ \texttt{else}\ \{\,y \coloneqq 2 \cdot y\,\}$$

Assume the precondition $y = 10$ and the postcondition $y = 11$. As O'Hearn [9], we can now ask: Is this postcondition an underapproximation of the states reachable by executing P_y starting from the precondition. Or equivalently: Is triple $[\, y = 10 \,]\ p\ [\, y = 11 \,]$ valid for incorrectness?

It is not. This seems unintuitive at first. However, consider a final state where $y = 11$ and $x = 1$. This state is unreachable from initial states where $y = 10$; in fact, it is entirely unreachable from *any* initial state. This is because if x is odd, as it is in this example, the final value of y must be even.

This is where *partial* incorrectness logic comes into application. The triple $[\, y = 10 \,]\ p\ [\, y = 11 \,]$ *is* valid for partial incorrectness! Similar to how partial correctness treats divergence as acceptable behaviour, partial incorrectness accepts universal unreachability. By doing so, we mitigate the cause of the unintuitive scenario encountered before.

Formally, we can prove the validity of the triple for partial correctness by calculating $\mathsf{slp}[\![p]\!](y = 10)$, which is done by applying the rules presented in Table 1. Detailed calculations are shown in Figure 2. Since the postcondition $y = 11$ implies $\mathsf{slp}[\![p]\!](y = 10) = (x \text{ odd } \vee\, y = 11) \wedge (x \text{ even } \vee\, y \text{ odd} \vee\, y = 20)$, the above triple is valid for partial incorrectness. Thus, partial incorrectness can help focus the reasoning on variables of interest.

Backtracking responsibilities. With partial incorrectness triples in the form of $[\,\_\,]\ p\ [\,\text{true}\,]$, we can reason about responsibilities: The blank can be filled by initial states that are responsible for some result. Consider Schrödinger's cat sitting in a sealed box, described by the program $p \coloneqq \texttt{while}\,(\,\neg\text{open}\,)\,\{\,\text{dead} \coloneqq \text{spill}\,\};\ \text{dead} \coloneqq \text{spill}$. The cat may or may not be dead depending on whether a vial of poison was spilled. Once we open the box, we observe the cat's health. If we do not open the box, the variable dead may be true or false at any time, but we have no observations.

$$
\begin{array}{l}
\mathbin{/\!/\!/}\ \boldsymbol{y = 10} \\
\texttt{if}\,(\,x \text{ even}\,)\,\{ \\
\quad \mathbin{/\!/\!/}\ \boldsymbol{x \text{ odd} \vee y = 10} \\
\quad y \coloneqq y + 1 \\
\quad \mathbin{/\!/\!/}\ \boldsymbol{x \text{ odd} \vee y = 11} \\
\}\,\texttt{else}\,\{ \\
\quad \mathbin{/\!/\!/}\ \boldsymbol{x \text{ even} \vee y = 10} \\
\quad y \coloneqq 2 \cdot y \\
\quad \mathbin{/\!/\!/}\ \boldsymbol{x \text{ even} \vee y \text{ odd} \vee y = 20} \\
\} \\
\mathbin{/\!/\!/}\ \boldsymbol{(x \text{ odd} \vee y = 11) \wedge (x \text{ even} \vee y \text{ odd} \vee y = 20)}
\end{array}
$$

Figure 2: Calculating the strongest liberal post.

Here, $[\,\text{open}\,]\ p\ [\,\text{true}\,]$ is a valid partial incorrectness triple. The postcondition true tells us that any reachable final state must have an origin satisfying open: The action of opening the box caused us to have knowledge over the cat.

Similar reasoning can be acquired by *total* incorrectness. However, this is subject to the strong condition that there are no unreachable states at all. For this example, $[\,\text{open}\,]\ p\ [\,\text{true}\,]$ is indeed invalid for total incorrectness.

4 CONCLUSION

We discussed the relatively unexplored notion of partial incorrectness. In doing so, we stressed the duality between the weakest (liberal) pre and strongest (liberal) post; correctness and incorrectness; termination and reachability. We argue that partial incorrectness also proves useful in computational terms, as does partial correctness: Park's induction allows simple proofs of *partial* incorrectness but not of *total* incorrectness. By adding a proof of reachability, we can get from partial to total incorrectness.

The examples show that partial incorrectness allows us to drop irrelevant variables and reason about responsibilities. Further applications for partial incorrectness are still open to investigation.

Quantitative programs. For simplicity, we chose the pre and post to be Boolean predicates. However, techniques mentioned here extend to quantitative *expectations* [7, 8] and we conjecture that the results transfer.

Proving reachability. We presented a variant rule for proving reachability of loops. However, assignments can also cause unreachability, and their computation is not straightforward; efficient methods for this would be valuable to investigate.

Nomenclature. We feel that the names "correctness" and "incorrectness" are not covering the entire range of applications of the logics. Certainly, when we choose the postcondition to represent bugs, we reason about unwanted properties of our programs, but as is shown in our examples, we can just as well analyse properties where the post represents desirable final states.

REFERENCES

[1] Krzysztof R. Apt and Ernst-Ruediger Olderog. 2019. Fifty Years of Hoare's Logic. arXiv:1904.03917 [cs] http://arxiv.org/abs/1904.03917

[2] Edsko de Vries and Vasileios Koutavas. 2011. Reverse Hoare Logic. In *Software Engineering and Formal Methods*, Gilles Barthe, Alberto Pardo, and Gerardo Schneider (Eds.). Vol. 7041. Springer Berlin Heidelberg, Berlin, Heidelberg, 155–171. https://doi.org/10.1007/978-3-642-24690-6_12

[3] Edsger W Dijkstra. 1975. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. 18, 8 (1975).

[4] Edsger W. Dijkstra and Carel S. Scholten. 1990. *Predicate Calculus and Program Semantics*. Springer, New York, NY. https://doi.org/10.1007/978-1-4612-3228-5

[5] Marcel Hark, Benjamin Lucien Kaminski, Jürgen Giesl, and Joost-Pieter Katoen. 2020. Aiming Low Is Harder – Induction for Lower Bounds in Probabilistic Program Verification. *Proceedings of the ACM on Programming Languages* 4, POPL (Jan. 2020), 1–28. https://doi.org/10.1145/3371105 arXiv:1904.01117 [cs]

[6] C A R Hoare. 1969. An Axiomatic Basis for Computer Programming. 12, 10 (1969).

[7] Benjamin Lucien Kaminski. 2019. *Advanced weakest precondition calculi for probabilistic programs*. Ph. D. Dissertation. RWTH Aachen University, Germany. http://publications.rwth-aachen.de/record/755408

[8] Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 18, 3 (1996), 325–353.

[9] Peter W. O'Hearn. 2020. Incorrectness Logic. *Proceedings of the ACM on Programming Languages* 4, POPL (Jan. 2020), 1–32. https://doi.org/10.1145/3371078

[10] Linpeng Zhang and Benjamin Lucien Kaminski. 2022. Quantitative Strongest Post: A Calculus for Reasoning about the Flow of Quantitative Information. *Proceedings of the ACM on Programming Languages* 6, OOPSLA1 (April 2022), 1–29. https://doi.org/10.1145/3527331