A Taxonomy of Hoare-Like Logics

Towards a Holistic View using Predicate Transformers and Kleene Algebras with Top and Tests

LENA VERSCHT, Saarland University, Germany and RWTH Aachen, Germany
BENJAMIN LUCIEN KAMINSKI, Saarland University, Germany and University College London, UK

We study Hoare-like logics, including partial and total correctness Hoare logic, incorrectness logic, Lisbon logic, and many others through the lens of predicate transformers à la Dijkstra and through the lens of Kleene algebra with top and tests (TopKAT). Our main goal is to give an overview – a taxonomy – of how these program logics relate, in particular under different assumptions like for example program termination, determinism, and reversibility. As a byproduct, we obtain a TopKAT characterization of Lisbon logic, which – to the best of our knowledge – is a novel result.

CCS Concepts: • **Theory of computation** → Program semantics; **Hoare logic**; *Programming logic*; **Logic and verification**; **Pre- and post-conditions**; *Program verification*; **Program specifications**.

Additional Key Words and Phrases: program logics, predicate transformers, Kleene algebra with top and tests

ACM Reference Format:
Lena Verscht and Benjamin Lucien Kaminski. 2025. A Taxonomy of Hoare-Like Logics: Towards a Holistic View using Predicate Transformers and Kleene Algebras with Top and Tests. *Proc. ACM Program. Lang.* 9, POPL, Article 60 (January 2025), 30 pages. https://doi.org/10.1145/3704896

1 Introduction

Arguably one of the most prominent and well-studied program logics is Hoare [1969] logic for verifying program *correctness*: the *ability of* a specified set of *initial states to reach only* some specified set of (safe) *final states*. More recently, *incorrectness* logic has attracted considerable attention [O'Hearn 2019; Zhang et al. 2022; Zhang and Kaminski 2022; Zilberstein et al. 2023]. Here, the property of interest is to prove that an entire specified set of undesired *final states is reachable from* a specified set of *initial states*, thus proving the (true positive) presence of a bug. Program correctness and incorrectness were described as "two sides of the same coin" [O'Hearn 2019]. We argue that there are at least three sides or rather *dimensions* to the program logic coin:

(1) *correctness* (being able to reach) vs. *incorrectness* (being reachable)
(2) *totality* vs. *partiality*
(3) *angelic* vs. *demonic* resolution of nondeterminism

We will explore how one can pigeonhole *total* and *partial correctness*, as well as *incorrectness* into this view and explore further program properties that emerge from exhausting all combinations of the above dimensions. Our primary objective is to build a taxonomy of program logics that is in a sense exhaustive and provides an overview of existing logics, which are often referenced to under different names (e.g. reverse Hoare logic [de Vries and Koutavas 2011] essentially is the same as incorrectness logic [O'Hearn 2019]).

Authors' Contact Information: Lena Verscht, lverscht@cs.uni-saarland.de, Saarland University, Saarbrücken, Germany and RWTH Aachen, Aachen, Germany; Benjamin Lucien Kaminski, kaminski@cs.uni-saarland.de, Saarland University, Saarbrücken, Germany and University College London, London, UK.

ACM 2475-1421/2025/1-ART60
https://doi.org/10.1145/3704896

We choose *predicate transformers* [Dijkstra 1976] as our primary formalism for expressing program properties. For example, using *weakest liberal preconditions*, we can express the validity of partial correctness Hoare triples as $b \subseteq \mathsf{wlp}[\![p]\!](c)$, expressing that the precondition b is included in the set $\mathsf{wlp}[\![p]\!](c)$ of states on which p either terminates in c or not at all. As a secondary formalism, we choose *Kleene algebra with tests* [Kozen 1997] or rather *Kleene algebra with top and tests (*TopKAT*)* [Zhang et al. 2022]. The basic idea is to model programs (as well as pre- and postconditions) as elements of a relational algebra. We then build *terms* by composing these relations and express program properties as equations between terms. For instance, partial correctness can be expressed in TopKAT as $\top bpc = \top bp$. This essentially expresses that the codomains of the relations bpc (filter initial states for b, execute p, filter final states for c) and bp (filter initial states for b, execute p) are equal.

Related Work. Given a recent surge in novel program logics, there has been substantial interest in developing frameworks which unify those logics. Zilberstein et al. [2023] introduce *outcome logic*, a framework capable of accommodating both correctness and incorrectness reasoning. Similarly, Cousot [2024] explores program logics through the lens of abstract interpretation, constructing and comparing various logics within that framework. Ascari et al. [2024] contrast Hoare logic, incorrectness logic, and the related logic of *necessary preconditions*. In their resulting taxonomy, they address a gap by introducing what they term *sufficient incorrectness logic*. More work in this direction includes [Bruni et al. 2023; Maksimović et al. 2023; Milanese and Ranzato 2022; Wickerson 2024]. We give many more pointers to related work scattered across the paper, in particular Section 4.2.

Contributions & Organization. In Section 2, we present syntax and semantics of a simple nondeterministic model programming language. In Section 3, we (gently and systematically) introduce 8 different predicate transformers à la Dijkstra. Some were described before, some are new, emerging from an investigation of nondeterminism in forward analysis.

We furthermore identify a simple, yet crucial difference between forward and backward analyses: Given a program p and an initial state σ, executing p on σ may (nondeterministically) both terminate in some final state (execution leads to somewhere) and also not terminate at all (execution leads to nowhere). However, given a final state τ, it cannot be that by executing p the state τ was both reachable (execution came from somewhere) and unreachable (execution cam from nowhere).

In Section 4, we use these predicate transformers to define 16 different program logics (by over and underapproximating each of the 8 transformers). These include well-known ones like partial and total correctness Hoare logic and incorrectness logic, but also lesser known (Lisbon logic) and new ones. We study relationships across these logics, thus building a taxonomy of them. We furthermore study how this taxonomy (partly) collapses under different assumptions like e.g. program termination.

In Section 5, we study how to express various of these 16 program logics in Kleene algebra with top and tests, *including Lisbon logic, which to the best of our knowledge is a new result.* In Section 6, we give an updated taxonomy through the lens of TopKAT. Throughout this paper, we will discover several incongruities in an otherwise quite symmetrical and dual taxonomy. We summarize these in Section 7. Proofs, definitions, and additional examples are provided in an extended version of this paper [Verscht and Kaminski 2024].

2 The Nondeterministic Guarded Command Language

We consider programs in a simple *nondeterministic guarded command language* (nGCL):

$$p \;::=\; \mathtt{skip} \;\mid\; x \coloneqq e \;\mid\; p \fatsemi p \;\mid\; \{\,p\,\} \,\square\, \{\,p\,\} \;\mid\; \mathtt{if}\,(b)\,\{\,p\,\}\,\mathtt{else}\,\{\,p\,\} \;\mid\; \mathtt{while}\,(b)\,\{\,p\,\}$$

Here, $x \in \mathsf{Vars}$ is a variable. The set of *program states* $(\sigma, \tau, \dots)$ is given by $\Sigma = \{\, \sigma \mid \sigma : \mathsf{Vars} \to \mathbb{Z} \,\}$, i.e. functions mapping program variables to integers. By $\sigma\,[x/v]$, we denote a new state that is obtained from σ by letting variable $x \in \mathsf{Vars}$ evaluate to $v \in \mathbb{Z}$. Formally: $\sigma\,[x/v]\,(y) = v$, if $y = x$; and $\sigma\,[x/v]\,(y) = \sigma(y)$, otherwise.

In the grammar above, e is an arithmetic expression, and b is a *predicate* or *test* or *condition* (we use these interchangeably). For our intents and purposes, a predicate is a subset of program states $b \subseteq \Sigma$, i.e. a function mapping program states to truth values $\{0,\ 1\}$. We denote by $\mathbb{B} = \{0,\ 1\}^{\Sigma}$ the set of all predicates. The negation of a predicate b is denoted by $\overline{b}$. Given a program state σ, we denote by $\sigma(\xi)$ the evaluation of an arithmetic expression or predicate ξ in σ.

The semantics of nGCL programs is given in terms of a *collecting* semantics (as is standard in program analysis, see [Cousot and Cousot 1977; Hecht 1977; Rival and Yi 2020]).

Definition 2.1 (Collecting Semantics for nGCL *Programs).* Let $S \subseteq \Sigma$ be a set of program states and let $[\![b]\!]S = \{\, \sigma \in S \mid \sigma \models b \,\}$ be a filtering of S to only those states where the predicate b holds. The *collecting semantics* $[\![p]\!] \colon \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ of an nGCL program p is defined inductively by

$$
\begin{aligned}
[\![\mathtt{skip}]\!]S &= S && \text{(effectless program)}\\
[\![x \coloneqq e]\!]S &= \{\sigma\,[x/\sigma(e)] \mid \sigma \in S\} && \text{(assignment)}\\
[\![p_1 \fatsemi p_2]\!]S &= ([\![p_2]\!] \circ [\![p_1]\!])S && \text{(sequential composition)}\\
[\![\mathtt{if}\ (\,b\,)\ \{\,p_1\,\}\ \mathtt{else}\ \{\,p_2\,\}]\!]S &= ([\![p_1]\!] \circ [\![b]\!])S \cup ([\![p_2]\!] \circ [\![\overline{b}]\!])S && \text{(conditional choice)}\\
[\![\mathtt{while}\,(\,b\,)\,\{\,p\,\}]\!]S &= [\![\overline{b}]\!]\big(\mathsf{lfp}\ X.\ S \ \cup\ ([\![p]\!] \circ [\![b]\!])X\big) && \text{(loop)}\\
[\![\{\,p_1\,\} \,\square\, \{\,p_2\,\}]\!]S &= [\![p_1]\!]S \cup [\![p_2]\!]S\,. && \text{(nondeterministic choice)}
\end{aligned}
$$

By slight abuse of notation, we write $[\![p]\!](\sigma)$ for $[\![p]\!]\{\sigma\}$. We denote the inverse semantics of p by $[\![p]\!]^{-1}T = \left\{\, \sigma \;\middle|\; T \cap [\![p]\!](\sigma) \neq \emptyset \,\right\}$. If a program p does not terminate, we say that p *diverges*. ◁

In words, for a program $p \in$ nGCL, the collecting semantics $[\![p]\!]S$ maps a set of initial program states $S \in \mathcal{P}(\Sigma)$ to the set of all final states reachable from a state in S.

Remark 1 (Diverging Programs). Consider the program `skip` that does nothing and the program $\{\,\mathtt{skip}\,\} \,\square\, \{\,\mathtt{while}\,(\,\mathtt{true}\,)\,\{\,\mathtt{skip}\,\}\,\}$ that nondeterministically decides between doing nothing and diverging. Their collecting semantics are given by

$$
\begin{aligned}
[\![\mathtt{skip}]\!]S \;=\; S \cup \emptyset \;&=\; [\![\mathtt{skip}]\!]S \;\cup\; [\![\mathtt{while}\,(\,\mathtt{true}\,)\,\{\,\mathtt{skip}\,\}]\!]S\\
&=\; [\![\{\,\mathtt{skip}\,\} \,\square\, \{\,\mathtt{while}\,(\,\mathtt{true}\,)\,\{\,\mathtt{skip}\,\}\,\}]\!]S\,,
\end{aligned}
$$

i.e. their collecting semantics coincide, despite their arguably different behaviors. ◁

3 Predicate Transformers

Deductive reasoning on source code level with *predicate transformers* is due to Dijkstra [1975]. There are fundamentally two types of predicate transformers: backward moving weakest preconditions and forward moving strongest postconditions. The terms *weakest* and *strongest* are rooted historically in their relationship to Hoare logic. They loose their plausibility in other contexts like incorrectness logic, but – also for lack of better names – we stick here to these historical terms.

3.1 Weakest Preconditions

The weakest precondition calculus features predicate transformers of type $\mathsf{wp}[\![p]\!] \colon \mathbb{B} \to \mathbb{B}$ for a program $p \in$ nGCL. Given a postcondition $c \in \mathbb{B}$, the weakest precondition of c (under p) is a predicate $\mathsf{wp}\,[\![p]\!](c)$ containing precisely those (initial) states from which the computation of p

(1) terminates and (2) does so in a (final) state satisfying c. We will also say that $\mathsf{wp}\,[\![p]\!](c)$ are the initial states that are *coreachable* from postcondition c by executing p.

In the presence of nondeterminism, we must decide whether, for each initial state σ, we require *all* computation paths emerging from σ to terminate in c or whether we require merely the *existence* of such a path. Traditionally, the former (*all*) is referred to as *demonic* nondeterminism, and the latter (*exist*) as *angelic*. These two design choices can be accommodated within Dijkstra's calculi, resulting in two slightly different variants of weakest precondition transformers.

Definition 3.1 (Weakest Precondition Transformers [Dijkstra 1976]*).* Given a program p and a postcondition c, the *angelic weakest precondition* is defined as

$$\mathsf{awp}[\![p]\!](c) \quad = \quad \lambda\sigma.\ \bigvee_{\tau \in [\![p]\!](\sigma)} c(\tau)\ .$$

The *demonic weakest precondition* is defined as

$$\mathsf{dwp}[\![p]\!](c) \quad = \quad \lambda\sigma.\ \begin{cases} \mathsf{false}, & \text{if } p \text{ can diverge on } \sigma \\ \bigwedge\limits_{\tau \in [\![p]\!](\sigma)} c(\tau), & \text{otherwise}\ . \end{cases}$$

◁

It is worthwhile to note that dwp is the standard transformer introduced by Dijkstra.

Both awp and dwp are *total correctness* transformers in the sense that both deem nontermination undesired behavior. More precisely, $\mathsf{dwp}[\![p]\!](c)$ indeed contains no initial states whatsoever on which p could possibly diverge. $\mathsf{awp}[\![p]\!](c)$, on the other hand, may contain such initial states σ, but there must then also exist a (nondeterministic) possibility for σ to terminate in c.

3.1.1 Liberality. Proving total correctness can be separated into two subtasks, namely (1) proving *partial correctness* (i.e. correctness modulo termination) and (2) proving *termination*. Proving partial correctness motivates the definition of *liberal* variants of the aforementioned calculi: Given a postcondition c, the weakest *liberal* precondition of c (under p) is a predicate $\mathsf{wlp}[\![p]\!](c)$ containing precisely those states from which the computation of p either (1) diverges or (2) terminates in a state satisfying c. In other words, the computation reaches c, *if* it terminates at all. As in the non-liberal case, we again have both demonic and angelic variants of wlp.

Definition 3.2 (Weakest Liberal Precondition Transformers [Dijkstra 1976]*).* Given a program p and a postcondition c, the *demonic weakest liberal precondition* is defined as

$$\mathsf{dwlp}[\![p]\!](c) \ = \ \lambda\sigma.\ \bigwedge_{\tau \in [\![p]\!](\sigma)} c(\tau)\ .$$

The *angelic weakest liberal precondition* is defined as

$$\mathsf{awlp}[\![p]\!](c) \ = \ \lambda\sigma.\ \begin{cases} \mathsf{true}, & \text{if } p \text{ can diverge on } \sigma \\ \bigvee\limits_{\tau \in [\![p]\!](\sigma)} c(\tau), & \text{otherwise}\ . \end{cases}$$

◁

Again, we note that the demonic variant dwlp is the standard one studied by Dijkstra.

Both dwlp and awlp are *partial correctness* transformers in the sense that both deem nontermination as acceptable behavior. More precisely, $\mathsf{dwlp}[\![p]\!](c)$ indeed contains all initial states on which p *must* either diverge or terminate in c. On the other hand, $\mathsf{awlp}[\![p]\!](c)$ contains all initial states on which p *may* either diverge or terminate in c.

3.1.2 Inductive Definitions of Weakest Precondition Transformers. The angelic/demonic weakest (liberal) precondition transformers can *all* be defined by induction on the program structure, see [Verscht and Kaminski 2024, Appendices D.1 and D.2].

We note also that the direction of analysis is *backwards*, as we start with a predicate over final states and transform it into a predicate on initial states. The result of the analysis on the other hand is a *forecast*: a weakest precondition forecasts for each initial state whether after the computation the postcondition will be satisfied.

3.1.3 Anatomy of Weakest Precondition Transformers. Given an initial state σ, a program p, and a postcondition c, we can make out three behavioral dimensions with respect to how p behaves when executed on σ:

(wd1) p can terminate in c or not,
(wd2) p can terminate in $\overline{c}$ or not,
(wd3) p can diverge or not.

In the presence of nondeterminism, these dimensions span a space of $2^3 = 8$ different types of behaviors that p can exhibit. For example, a program could nondeterministically both terminate in c or diverge. We call the behavioral classes in that space *coreachability classes*, since they speak about whether initial states are coreachable from c (or $\overline{c}$).

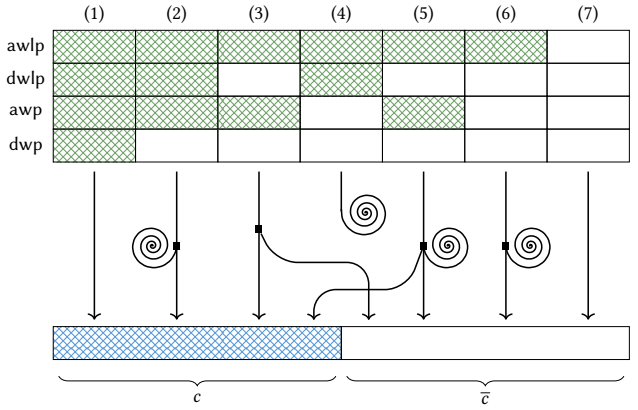


Fig. 1. Illustration of different coreachability classes and different wp transformers. The top part represents initial states, divided (in columns) into coreachability classes. The lower part represents final states, divided into those that satisfy postcondition c and those that do not. On top, the colored boxed indicate which coreachability classes are included in which transformers.

For example, there is a coreachability class containing all initial states from which p can both diverge and reach c. However, it cannot happen that p neither terminates in c, nor in $\overline{c}$, nor diverges. This thus leaves $2^3 - 1 = 7$ sensible coreachability classes, which are illustrated in Figure 1:

(1) p terminates in c.
(2) p terminates in c or diverges (i.e. p does not terminate in $\overline{c}$).
(3) p terminates in c or terminates in $\overline{c}$ (i.e. p terminates).
(4) p diverges.
(5) p terminates in c, or in $\overline{c}$, or diverges (i.e. no restriction on behavior of p).
(6) p terminates in $\overline{c}$ or diverges (i.e. p does not terminate in c).
(7) p terminates in $\overline{c}$.

These coreachability classes fully partition the state space. A precondition transformer can now opt to include a class (i) in its result or not. The four rather natural transformers we described above are illustrated in Figure 1. Along a *row*, green shaded boxes indicate which classes are included in the respective weakest precondition transformer with respect to postcondition c (in blue). For instance, $\mathrm{awp}[\![p]\!](c)$ includes classes (1), (2), (3), and (5).

Assuming that the above coreachability classes are meaningful, this makes for $2^7 = 128$ possible weakest precondition transformers[1] of which some might not even depend on the postcondition c (e.g. include only (4)), others might even be trivial (e.g. include none, include all). Some are

[1]Without nondeterminism, there would be only 3 coreachability classes (reach c, reach $\overline{c}$, diverge) and the number of possible precondition transformers would reduce to $2^3 = 8$.

contrapositives of each other. For example, awp and dwlp are *contrapositive* in the sense that

$$\mathrm{awp}[\![p]\!](c) \;=\; \overline{\mathrm{dwlp}[\![p]\!](\overline{c})} \qquad \text{and} \qquad \mathrm{dwlp}[\![p]\!](c) \;=\; \overline{\mathrm{awp}[\![p]\!](\overline{c})}$$

This contrapositivity relationship also holds for dwp and awlp. The contrapositivities can also be rediscovered graphically in Figure 1: To get from awlp to dwp, for instance, proceed as follows:

(1) Take row 1 (awlp).
(2) Invert colors (i.e. turn green shaded boxes into empty ones and vice versa; corresponds to negating the entire result).
(3) Mirror the entire row horizontally (corresponds to negating the postcondition).
(4) Obtain row 4 (dwp).

3.2 Strongest Postconditions

Weakest precondition transformers are well-researched and have been extended for various purposes, including quantities, probabilistic programs, and even quantum programs [D'hondt and Panangaden 2006; Kaminski et al. 2018; Morgan et al. 1996]. Dually to weakest preconditions, Dijkstra and Scholten [1990] defined *strongest postcondition transformers*, also of type $\mathrm{sp}[\![p]\!] : \mathbb{B} \to \mathbb{B}$. Given now a *pre*condition $b \in \mathbb{B}$, the strongest postcondition of b (under p) is a predicate $\mathrm{sp}\,[\![p]\!](b)$ containing precisely those (final) states that are *reachable* from an (initial) state in b by executing p.

Definition 3.3 (Strongest Postcondition Transformer [Dijkstra 1976]*).* Given a program $p \in \mathsf{nGCL}$ and a precondition $b \in \mathsf{Pred}$, the *angelic strongest postcondition* is defined as

$$\mathrm{asp}\,[\![p]\!](b) \;=\; \lambda\tau.\; \bigvee_{\sigma \in [\![p]\!]^{-1}(\tau)} b(\sigma). \qquad \triangleleft$$

As suggested by the operator name asp, we have opted here for *angelic* resolution of nondeterminism. This is also the standard choice of Dijkstra. We will discuss the nature of this nondeterminism as well as demonic sp later in Section 3.2.2.

3.2.1 Liberality. Dually to strongest postconditions, there are also strongest liberal postconditions, originally described by Cousot and then later given an inductive definition by Zhang and Kaminski [2022]. Given a *pre*condition $b \in \mathbb{B}$, the strongest liberal postcondition of b (under p) is a predicate $\mathrm{slp}[\![p]\!](b)$ containing precisely those (final) states that are reachable (1) *exclusively* from (initial) states in b or (2) *entirely unreachable* (i.e. from *any* initial state in Σ) by the computation of p.

Definition 3.4 (Strongest Liberal Postcondition Transformers [Zhang and Kaminski 2022]*).* Given a program p and a precondition b, the *demonic strongest liberal postcondition* is defined as

$$\mathrm{dslp}[\![p]\!](b) \;=\; \lambda\tau.\; \bigwedge_{\sigma \in [\![p]\!]^{-1}(\tau)} b(\sigma). \qquad \triangleleft$$

As the name dslp suggests, we now consider *demonic* resolution of nondeterminism. As shown above, strongest (liberal) postconditions can again be defined via the collecting semantics. The inductive rules can be found in [Verscht and Kaminski 2024, Appendices D.3 and D.4].

The analysis direction is now *forwards*: starting with a predicate on initial states, we transform it into a predicate on final states. The result of the analysis on the other hand is a *backcast*: a strongest postcondition backcasts for each final state whether the computation started in the precondition.

The *demonic* strongest liberal post is intentional because it makes for a very dual theory as we will see later. In the context of the strongest post reasoning, liberality refers to unreachability rather than divergence. A state from which computation diverges does not have a final state that it reaches, i.e. it does not terminate. Conversely, an unreachable state does not have an initial

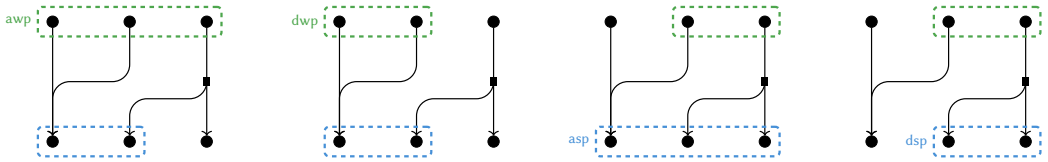


Fig. 2. Duality of angelic and demonic weakest pre versus angelic and demonic strongest post, demonstrated on four copies of the same program. Preconditions are circled in dashed green, postconditions in dashed blue. The rightmost initial state *can* terminate in the postcondition, but may also terminate outside. Therefore, it is included in the angelic but not in the demonic weakest precondition. Dually, the leftmost final state is reachable from the precondition but also from outside. Therefore, it is included in the angelic but not in the demonic strongest postcondition.

state that reaches it, i.e. it is not reachable from any initial state. The duality of termination and reachability has previously been discussed by Zhang and Kaminski [2022] and Ascari et al. [2024].

The requirement of exclusive reachability - excluding states that can also be reached from outside of b - is what makes this transformer demonic. This concept will be further explored in the following section, where we discuss how nondeterminism arises in forward analyses, a topic that, to the best of our knowledge, has not been addressed in detail in the literature.

3.2.2 Resolution of Nondeterminism in Strongest Postconditions. Whereas nondeterminism in weakest preconditions arises from explicit nondeterministic branching in the program, the nondeterminism relevant for strongest postconditions arises from *confluence*, i.e. when multiple initial states lead to the same final state, even for deterministic computations. Consider for instance the fully deterministic program $x \coloneqq 2$. Then both states $\sigma(x) = 5$ and $\sigma'(x) = 17$ lead to the same final state $\tau(x) = 2$. If two different initial states can reach τ, this is somewhat dual to one initial state possibly terminating in two final states. For an illustration, see Figure 2.

When deciding whether a final state τ is in the strongest postcondition of some precondition b, we now need to choose whether we require *all* initial states that can reach τ to satisfy b or if it suffices if there *exists* such a state. Following backward terminology, we refer to the former as *demonic* and the latter as *angelic* resolution of nondeterminism. We have previously defined *angelic* strongest (non-liberal) and *demonic* strongest liberal postconditions. Consequently, we will now define the missing *demonic* strongest (non-liberal) and *angelic* strongest liberal postconditions.

Definition 3.5 (Demonic Strongest Postcondition Transformers). Given a program p and a precondition b, the *demonic strongest postcondition* is defined as

$$\mathrm{dsp}\,[\![p]\!](b) \;=\; \lambda\,\tau.\; \begin{cases} \bigwedge_{\sigma \in [\![p]\!]^{-1}(\tau)} b(\sigma), & \text{if } [\![p]\!]^{-1}(\tau) \neq \emptyset \\ \text{false}, & \text{otherwise}\,. \end{cases} \qquad \triangleleft$$

The demonic strongest post transformer maps a precondition b to the set of states that are exclusively reachable by the initial states contained in b. Aligning with the intuition above, this is a stronger requirement than for the angelic strongest post, and demonic in the sense that *all* paths leading to the final state in question have to originate in the given precondition.

Definition 3.6 (Angelic Strongest Liberal Postcondition Transformers). Given a program p and a precondition b, the *angelic strongest liberal postcondition* is defined as

$$\mathsf{aslp}[\![p]\!](b) \;=\; \lambda\tau.\; \begin{cases} \bigvee\limits_{\sigma \in [\![p]\!]^{-1}(\tau)} b(\sigma), & \text{if } [\![p]\!]^{-1}(\tau) \neq \emptyset \\ \mathsf{true}, & \text{otherwise .} \end{cases}$$ ◁

The angelic strongest liberal post maps a precondition b to the set of states that either are reachable from b or unreachable. Therefore, this is indeed a liberal extension of the angelic strongest post calculus as it accepts unreachable states.

3.2.3 Anatomy of Strongest Postcondition Transformers. Dual to the coreachability classes of initial states we considered for weakest preconditions, we will now consider *reachability classes* of *final* states for strongest postconditions. Given a final state τ, a program p, and a precondition b, we can make out two dimensions regarding what was the case before p reached τ:

(sd1) p could have been started in b or not,
(sd2) p could have been started in $\overline{b}$ or not,

These dimensions span a space of $2^2 = 4$ different types of behaviors. We call the behavioral classes in that space *reachability classes*, since they speak about whether final states are reachable from b (or $\overline{b}$). For example, there is a reachability class containing all final states that are reachable by an execution of p both from b and from $\overline{b}$. The four reachability classes are:

(1) p was started in b.
(2) p could have been started in b or in $\overline{b}$.
(3) The unreachable states.
(4) p was started in $\overline{b}$.

These reachability classes fully partition the state space.

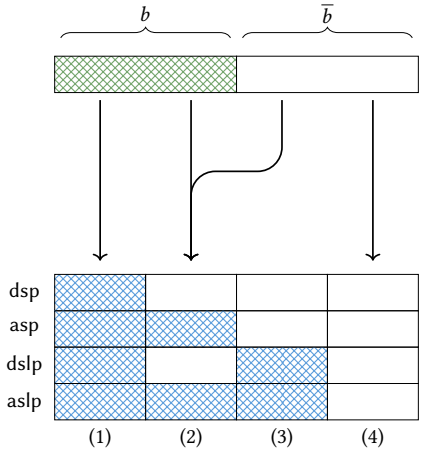


Fig. 3. An illustration of the different sp-style transformers. The upper part depicts all possible program executions for a fixed final state and a precondition b. Below, the colored boxed indicate which sets of final states are included in which transformers.

We note that a final state can either be unreachable or not, but never both. This is in contrast to coreachability where computations from an initial state can very well both diverge and terminate.

A postcondition transformer can now opt to include a class (i) in its result or not. The four natural transformers we described above are illustrated in Figure 3. Along a *row*, blue shaded boxes indicate which classes are included in the respective strongest postcondition transformer with respect to precondition b (in green). For instance, $\mathsf{asp}\,[\![p]\!](c)$ includes classes (1) and (2).

Assuming that the above reachability classes are meaningful, this makes for $2^4 = 16$ possible strongest postcondition transformers of which some might not even depend on the precondition b (e.g. include only (3)), others might be trivial (e.g. include none, include all). Some are contrapositives of each other. For example, asp and dslp are *contrapositive* in the sense that

$$\mathsf{asp}\,[\![p]\!](c) \;=\; \overline{\mathsf{dslp}[\![p]\!](\overline{c})} \qquad \text{and} \qquad \mathsf{dslp}[\![p]\!](c) \;=\; \overline{\mathsf{asp}\,[\![p]\!](\overline{c})}$$

This contrapositivity relationship also holds for dsp and aslp. The contrapositivities can also be rediscovered graphically in Figure 1 analogously to how this was the case for wp transformers.

3.2.4 Inductive Rules for Strongest Postcondition Transformers. The transformers asp and dslp can be defined by induction on the program structure (see [Verscht and Kaminski 2024, Appendices D.3 and D.4] for the concrete rules). For the novel transformers dsp and aslp, we cannot quite give an inductive set of rules. To see why, consider the nondeterministic program $\{ p_1 \} \square \{ p_2 \}$. Recall that the *angelic* strongest postcondition of b is the set of states that are reachable from b. The set of states reachable by $\{ p_1 \} \square \{ p_2 \}$ is the union of the states reachable from p_1 and from p_2, i.e. we get $\mathsf{asp}[\![\{ p_1 \} \square \{ p_2 \}]\!](b) = \mathsf{asp}[\![p_1]\!](b) \cup \mathsf{asp}[\![p_2]\!](b)$. Similarly, if we want to compute the demonic strongest liberal post, which contains the set of states unreachable or exclusively reachable from b, we take the intersection of the results for both subprograms and get $\mathsf{dslp}[\![\{ p_1 \} \square \{ p_2 \}]\!](b) = \mathsf{dslp}[\![p_1]\!](b) \cup \mathsf{dslp}[\![p_2]\!](b)$.

The demonic strongest post of b contains all states that are exclusively reachable from b. For $\{ p_1 \} \square \{ p_2 \}$, a final state τ is contained in this set if it fulfills one of the following three cases:

(1) τ is exclusively reachable from b by executing p_1 and unreachable by executing p_2.
(2) τ is exclusively reachable from b by executing p_2 and unreachable by executing p_1.
(3) τ is exclusively reachable from b by executing p_1 and exclusively reachable by executing p_2.

If we restrict to using only the demonic strongest post for the subprograms, the only case we can represent is (3) by $\mathsf{dsp}[\![p_1]\!](b) \cap \mathsf{dsp}[\![p_2]\!](b)$. For (1) and (2), we need to reason about unreachability, which is not possible using $\mathsf{dsp}[\![p_1]\!](b)$ or $\mathsf{dsp}[\![p_2]\!](b)$. Similar problems arise for aslp.

As a silver lining, we can characterize dsp as a combination of other transformers: We have that

$$\mathsf{dsp}[\![p]\!](b) = \mathsf{asp}[\![p]\!](b) \cap \mathsf{dslp}[\![p]\!](b),$$

as a final state τ must be exclusively reachable from b in order to be contained in $\mathsf{dsp}[\![p]\!](b)$. If $\tau \in \mathsf{dslp}[\![p]\!](b)$, we know that τ is either (1) unreachable or (2) exclusively reachable from b. If additionally $\tau \in \mathsf{asp}[\![p]\!](b)$, we know that τ is definitely reachable from b, which excludes case (1). This can also be seen in Figure 3: The intersection of $\mathsf{asp}[\![p]\!](b)$ and $\mathsf{dslp}[\![p]\!](b)$ only contains the first reachability class, which is equivalent to $\mathsf{dsp}[\![p]\!](b)$. Similarly, we have that

$$\mathsf{aslp}[\![p]\!](b) = \mathsf{asp}[\![p]\!](b) \cup \mathsf{dslp}[\![p]\!](b).$$

Therefore, although we cannot provide an inductive set of rules directly, we can make use of the existing rules and compute dsp (resp. aslp) with two inductive computations.

3.3 Backward vs. Forward Analysis

The characterization of the novel transformers dsp and aslp goes via union and intersection of existing transformers, raising the question whether we can do the same for the weakest pre calculi, i.e. whether we have that

$$\mathsf{dwp}[\![p]\!](c) \stackrel{?}{=} \mathsf{awp}[\![p]\!](c) \cap \mathsf{dwlp}[\![p]\!](c) \qquad \text{and} \qquad \mathsf{awlp}[\![p]\!](c) \stackrel{?}{=} \mathsf{awp}[\![p]\!](c) \cup \mathsf{dwlp}[\![p]\!](c).$$

This is – perhaps somewhat surprisingly – not the case. To see why, recall Figure 1. The set $\mathsf{dwlp}[\![p]\!](c)$ contains states from which computation always either terminates in c or diverges, and $\mathsf{awp}[\![p]\!](c)$ contains all states that can reach c. In Figure 1, their intersection corresponds to coreachability classes (1) and (2). This is not equivalent to $\mathsf{dwp}[\![p]\!](c)$, which excludes class (2), containing states from which computation either diverges or terminates in c.

The union of $\mathsf{awp}[\![p]\!](c)$ and $\mathsf{dwlp}[\![p]\!](c)$ contains states from which computation either can reach c or computation always diverges. A state from which computation either diverges or terminates outside of c (class no. 6) is not contained in this set, but in $\mathsf{awlp}[\![p]\!](c)$.

Note that both counterexamples are concerned with states with *branching divergence*, i.e. a computation that nondeterministically either diverges or terminates in some states. In fact, if we exclude such behavior, the equations above hold.

Table 1. An overview of the intuition for all presented predicate transformers.

transformer	captured states (precondition b, postcondition c)
angelic weakest pre (awp)	initial states that can reach c
angelic weakest liberal pre (awlp)	initial states that can reach c or diverge
demonic weakest pre (dwp)	initial states that can only reach c
demonic weakest liberal pre (dwlp)	initial states that can only reach c or diverge
angelic strongest post (asp)	final states that are reachable from b
angelic strongest liberal post (aslp)	final states that are reachable from b or unreachable
demonic strongest post (dsp)	final states that are exclusively reachable from b
demonic strongest liberal post (dslp)	final states that are exclusively reachable from b or unreachable

This also gives intuition as to why the respective equations hold for sp transformers. As previously mentioned, the dual concept of divergence is unreachability. Illustrated in Figure 4, branching divergence represents the division of a computation into one that reaches a final state and one that does not. Therefore, the dual should reflect the confluence of a computation that is reachable from an initial state and one that is not, as shown in the lower half of Figure 4. But, this is paradox: A final state can never be unreachable and reachable at the same time. We refer to this observation as the *absence of unreachability confluence*.

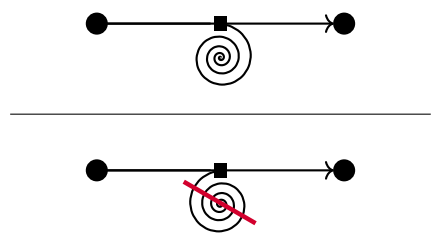


Fig. 4. Branching divergence versus confluence of unreachability.

Observation 2 (Absence of Unreachability Confluence). For a program p, an initial state σ, and a final state τ, we have that

$$\tau \text{ is unreachable by any computation of } p \quad \text{iff} \quad [\![p]\!]^{-1}(\tau) = \emptyset\,, \qquad \text{but}$$
$$p \text{ can diverge on input } \sigma \quad \cancel{\text{iff}} \quad [\![p]\!](\sigma) = \emptyset\,. \qquad \triangleleft$$

Observation 2 in particular enables us to characterize unreachability in a relational setting by testing equality to the empty set. This is not possible for divergence, weakening the previously discussed duality of divergence and unreachability (see e.g. Figure 2). We will see how this intrinsic difference between forward and backward analysis affects program logics in Sections 5 and 6. Another direct consequence of Observation 2 is that the demonic weakest pre is not truly dual to demonic strongest post, after all. For an illustration, see [Verscht and Kaminski 2024, Appendix A].

3.4 Relating Predicate Transformers

Table 1 summarizes the sets of states captured by the eight predicate transformers defined in the previous section. As indicated before, the transformers are closely related. The remainder of this section is dedicated to formalizing these relations.

First, we observe that we can order wp and sp transformers by set inclusion. This can also be seen in Figures 1 and 3, respectively.

Theorem 3.7 (Ordering on Predicate Transformers). *For all programs p and predicates c, b, the following inclusions hold:*

$$\begin{array}{ccc} \mathsf{awp}[\![p]\!](c) & \subseteq & \mathsf{awlp}[\![p]\!](c) \\ \cup\!| & & \cup\!| \\ \mathsf{dwp}[\![p]\!](c) & \subseteq & \mathsf{dwlp}[\![p]\!](c) \end{array} \qquad \begin{array}{ccc} \mathsf{asp}\,[\![p]\!](b) & \subseteq & \mathsf{aslp}[\![p]\!](b) \\ \cup\!| & & \cup\!| \\ \mathsf{dsp}\,[\![p]\!](b) & \subseteq & \mathsf{dslp}[\![p]\!](b) \end{array}$$

Proof. This follows directly from Definitions 3.1 to 3.6. □

It was hinted at before that the transformers are in contrapositive relation.

Theorem 3.8 (Contrapositive Transformers). *For all programs p and predicates b, c, we have:*

(1) $\mathsf{awp}[\![p]\!](c) = \overline{\mathsf{dwlp}[\![p]\!](\overline{c})}$ *(3)* $\mathsf{asp}\,[\![p]\!](b) = \overline{\mathsf{dslp}[\![p]\!](\overline{b})}$

(2) $\mathsf{dwp}[\![p]\!](c) = \overline{\mathsf{awlp}[\![p]\!](\overline{c})}$ *(4)* $\mathsf{dsp}\,[\![p]\!](b) = \overline{\mathsf{aslp}[\![p]\!](\overline{b})}$

Proof. Properties (1) to (3) are folklore knowledge. For the proof of (4), see [Verscht and Kaminski 2024, Appendix E.1]. □

4 A Taxonomy of Hoare-Like Program Logics

Arguably the best known program logic is Hoare logic [Hoare 1969]. Its central notion are *triples* $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ consisting of a program p, a precondition b, and a postcondition c. Such triples have also been called *asserted programs* in various literature. To give meaning to a triple, it must be *exegeted* when a triple is considered *valid* and when it is not. For the standard partial correctness interpretation of Hoare logic, it is well known that this exegesis can be captured in terms of predicate transformers, namely

$$\langle\, b \,\rangle\ p\ \langle\, c \,\rangle \text{ is valid for partial corr.} \qquad \text{iff} \qquad b \subseteq \mathsf{dwlp}[\![p]\!](c) \qquad \text{iff} \qquad \mathsf{asp}\,[\![p]\!](b) \subseteq c\ .$$

Another program logic that somewhat recently (re)gained attention under the name *incorrectness logic* [O'Hearn 2019] features triples that are exegeted through

$$\langle\, b \,\rangle\ p\ \langle\, c \,\rangle \text{ is valid for incorr.} \qquad \text{iff} \qquad c \subseteq \mathsf{asp}\,[\![p]\!](b)\ .$$

We see above that for some exegeses we *over*approximate the result of a predicate transformer, for others we *under*approximate it. In total, we have defined 8 different predicate transformers, which we can each over- and underapproximate, yielding potentially $2 \cdot 8 = 16$ different exegeses of triples and thus program logics. Some of these will coincide, as we have already seen above for partial correctness. Others will imply others, yet others will be contrapositives of each other. In the following, we give an overview of all 16 possibilities and study their relationships, thus yielding a taxonomy of predicate transformer definable program logics.

4.1 Program Logics

As described just above, there are 16 possible exegeses of triples $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ that can be obtained by over- or underapproximating each of the 8 predicate transformers defined in Section 3. A full overview of these is provided in Figure 5. The "colloquial terms" (broadly interpreted) of these exegeses/logics (some more, some less common) are provided immediately below the individual exegeses. Some of these names are more common than others. We would like to mention, however, that these names are not necessarily accurately chosen, especially the attribution of being a *correctness* or an *incorrectness* logic[2]. For example, *incorrectness logic* was prior to [O'Hearn 2019]

[2]But we acknowledge that these attributes made sense for the individual originally intended purposes of those logics.

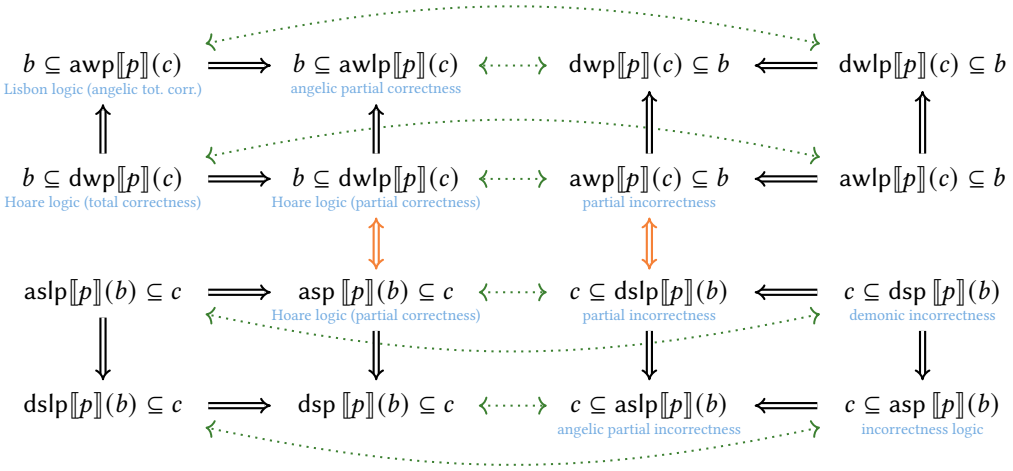


Fig. 5. A taxonomy of predicate transformer-based program logics. Black arrows are simple implications, green dotted arrows are contrapositive relations, and orange two-sided arrows are Galois connections.

introduced by de Vries and Koutavas [2011] under the name *reverse Hoare logic* and was intended for proving that all *good* things can happen – arguably more of a *correctness* criterion.

Other attributions are being an *over-* or an *under*approximate logic. For example, incorrectness logic was attributed underapproximate and partial correctness overapproximate. But this also is not entirely accurate since the supposedly overapproximate partial correctness can also be exegeted via an underapproximation, namely of dwlp. Before we next provide intuitions and background on the individual logics, let us first have a closer look at the *structure* of Figure 5.

The Implications. If we divide Figure 5 into four quadrants (i.e. horizontally and vertically in the middle), we obtain four squares of implications. For example (top left quadrant), total correctness implies partial correctness and the Lisbon logic exegesis. The latter two each imply angelic partial correctness. For each quadrant, the respective implications stem entirely from the ordering of the predicate transformers provided in Theorem 3.7.

The Contrapositions. If we divide Figure 5 vertically into two halves, then every exegesis in one half has a mirrored contrapositive in the other half, indicated by green dotted arrows. For example, partial correctness and partial incorrectness are contrapositive to each other, meaning that

$$\langle\, b \,\rangle\ p\ \langle\, c \,\rangle \text{ is valid for part. corr.} \qquad \text{iff} \qquad \left\langle\, \overline{b} \,\right\rangle\ p\ \left\langle\, \overline{c} \,\right\rangle \text{ is valid for part. incorr.}$$

In that sense, an exegesis and its contrapositive are "equivalent" and one of the two halves could be discarded. We would argue, however, that – depending on the proof objective – it may well be more intuitive to annotate code in non-negated versions so that program proofs remain more understandable. All contrapositions of exegesis in Figure 5 arise from the contrapositivities of the predicate transformers described in Theorem 3.8.

The Equivalences. In the very center of Figure 5, we have a square of, respectively, two vertically equivalent exegesis. For example, partial correctness can be exegeted equivalently through $b \subseteq \mathrm{dwlp}[\![p]\!](c)$ or $\mathrm{asp}\,[\![p]\!](b) \subseteq c$. Analogously, partial incorrectness can be exegeted in two equivalent ways. The partial correctness equivalence stems from the very well-known Galois connection

$$b \subseteq \mathrm{dwlp}[\![p]\!](c) \qquad \text{iff} \qquad \mathrm{asp}\,[\![p]\!](b) \subseteq c\ .$$

The partial incorrectness equivalence stems from the much less well-known Galois connection [Zhang and Kaminski 2022]

$$\mathsf{awp}[\![p]\!](c) \subseteq b \qquad \text{iff} \qquad c \subseteq \mathsf{dslp}[\![p]\!](b)\ .$$

Note that such Galois connections are not only of theoretical interest: *They* allow to reason either forward or backward through a program, whatever is more feasible. They even allow to reason in both directions simultaneously: For example, $\langle\, b \,\rangle\ p_1 \fatsemi p_2\ \langle\, c \,\rangle$ is valid for partial correctness if $\mathsf{asp}\,[\![p_1]\!](b) \subseteq \mathsf{dwlp}[\![p_2]\!](c)$, meaning that we have done backward reasoning on p_2 and forward reasoning on p_1. Analogous bidirectional reasoning can be performed for partial incorrectness.

In the following, we will discuss those logics of Figure 5 that have a colloquial name in more detail. We will proceed more or less in (partial) order of popularity. As a running example, we use a program p_{login} which realizes some login procedure. The user must enter a password and is then either granted access or not.

Hoare logic for partial correctness [Hoare 1969]. $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for *(demonic) partial correctness* iff $b \subseteq \mathsf{dwlp}[\![p]\!](c)$ or equivalently $\mathsf{asp}\,[\![p]\!](b) \subseteq c$. This is the *standard* notion of partial correctness, stating that all computations of p started in b *must* either diverge or terminate in c.

For example, validity of the partial correctness Hoare triple $\langle\, \mathit{pwd\ incorrect} \,\rangle\ p_{\text{login}}\ \langle\, \mathit{access\ denied} \,\rangle$ expresses the following: Should the user provide the wrong password, the login will fail by either terminating in a state where the user is denied access, or diverge. This is a correctness property as it expresses behavior we would expect from a login procedure (except perhaps the divergence).

The contrapositive triple $\mathsf{awp}[\![p]\!](c) \subseteq b$ was discussed by Cousot et al. [2013] under the name *necessary precondition*. The intuition for this is that b necessarily has to hold for computation to terminate in c. All other computation is guaranteed to either diverge or terminate outside of c.

Hoare logic for total correctness [Hoare 1969]. $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for *(demonic) total correctness* iff $b \subseteq \mathsf{dwp}[\![p]\!](c)$. This is the *standard* notion of total correctness, stating that all computations of p started in b *must* terminate in c. This can be classically used to specify correctness properties of the program. For example, the triple $\langle\, \mathit{pwd\ correct} \,\rangle\ p_{\text{login}}\ \langle\, \mathit{access\ granted} \,\rangle$ expresses that if a user enters the correct password, access is definitely granted.

Incorrectness logic [de Vries and Koutavas 2011; O'Hearn 2019]. $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for *(angelic) incorrectness* iff $c \subseteq \mathsf{asp}\,[\![p]\!](b)$. This exegesis was popularized by O'Hearn [2019] under the name *incorrectness logic* and ensures that *all* states in c must be reachable from *some* state in b.

Why *incorrectness*? O'Hearn thought of c as a set of bugs whose (true positive) presence was supposed to be proved. For example, the triple $\langle\, \mathsf{true} \,\rangle\ p_{\text{login}}\ \langle\, \mathit{error} \,\rangle$ expresses that the error state is reachable, thus proving the existence of some bug.

As mentioned before, incorrectness logic was described some 8 years earlier by de Vries and Koutavas [2011] under the name *reverse Hoare logic*. Contrary to O'Hearn, de Vries and Koutavas thought of c as a set of *desirable* states who should *all* be reachable, thus exemplifying that neither the attribute *incorrectness* nor *correctness* is entirely accurate for this logic.

Lisbon logic. $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for *angelic total correctness* iff $b \subseteq \mathsf{awp}[\![p]\!](c)$, stating that from all states in b, there must *exist* a computation of p terminating in c. Regarding its name, Zilberstein et al. [2023] describe that such triples have first been described in [Möller et al. 2021] as *backwards under-approximate triples* and been discussed as a possible foundation for *incorrectness* reasoning during a meeting in Lisbon, hence *Lisbon logic*. Recently, a proof system for the logic was developed [Raad et al. 2024]. However, underapproximating angelic weakest pre(conditions) has been studied much earlier, for example by Hoare [1978] as *possible correctness* and later by McIver and Morgan [2005]. Another name for Lisbon logic is *sufficient incorrectness logic*, due to Ascari et al. [2024].

As an example, consider again the triple $\langle\, pwd\ correct \,\rangle\ p_{\text{login}}\ \langle\, access\ granted \,\rangle$. Exegeted as angelic total correctness, this expresses that with a correct password, it is always possible to get access – indeed a *correctness* property. On the other hand, if $\langle\, pwd = 1234 \,\rangle\ p_{\text{login}}\ \langle\, access\ granted \,\rangle$ is valid for angelic total correctness, the password "1234" can always result in access. This is likely *incorrect* behavior. Again, neither *correctness* nor *incorrectness* seem appropriate attributes.

Partial incorrectness [Zhang and Kaminski 2022]. $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for *(demonic) partial incorrectness* iff $c \subseteq \mathsf{dslp}[\![p]\!](b)$, requiring all states in c to be either unreachable or *exclusively* reachable from b. In other words, computation starting in $\overline{b}$ may only terminate in $\overline{c}$, or: $\left\langle\, \overline{b} \,\right\rangle\ p\ \left\langle\, \overline{c} \,\right\rangle$ is valid for demonic partial correctness. This is not surprising because of contrapositivity.

Angelic partial correctness. $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for *angelic partial correctness* iff $b \subseteq \mathsf{awlp}[\![p]\!](c)$, stating that for all states in b, there must exist either a computation of p terminating in c, or the possibility of p to diverge. As the name suggests, this is very closely related to angelic total correctness, the only difference being that divergence is deemed acceptable.

This logic can be used to identify states from which divergence is possible by choosing the post to be empty. Raad et al. [2024] emphasize the relevance of reasoning about divergence, defining an *under-approximate non-termination logic* which aligns with the aforementioned angelic partial correctness triple $\langle\, b \,\rangle\ p\ \langle\, \mathsf{false} \,\rangle$.

Demonic incorrectness. $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for *demonic incorrectness* iff $c \subseteq \mathsf{dsp}\,[\![p]\!](b)$, stating that all final states in c must be *exclusively* reachable from the states in b. In particular, all states in c must be reachable. If we waive this requirement, we end up with partial incorrectness. This is dual to going from total to partial correctness by waiving the termination requirement.

To the best of our knowledge, demonic incorrectness logic has not been described in the literature before. It can be used, for example, to restrict the initial states from which errors can occur. This is similar to the motivation for outcome logic and sufficient incorrectness logic [Ascari et al. 2024; Zilberstein et al. 2023]. If $\langle\, pwd\ incorrect \,\rangle\ p_{\text{login}}\ \langle\, error \,\rangle$ is valid for demonic incorrectness, errors can only be reached when entering the wrong password, possibly making a bug at hand less critical.

Angelic partial incorrectness. $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for *angelic partial incorrectness* iff $c \subseteq \mathsf{aslp}[\![p]\!](b)$, stating that all states in the c are either unreachable or reachable from a state in b.

To the best of our knowledge, this is also a novel logic. If $\langle\, pwd\ incorrect \,\rangle\ p_{\text{login}}\ \langle\, access\ granted \,\rangle$ is valid for angelic partial incorrectness, we know that all states where access was granted are either entirely unreachable or can be reached with an incorrect password, which is definitely undesired.

4.2 Related Taxonomies

Several other works have developed taxonomies for program logics. Both Zhang and Kaminski [2022] and Ascari et al. [2024] concentrate on logics based on angelic semantics, which align with our awp and asp transformers. Cousot [2024] presents a framework that defines program logics by applying various abstraction functions to a collecting semantics. Even though the semantics in this paper in essence is also angelic, divergence is explicitly represented by the symbol $\bot$. In this way, demonic variants of correctness logics are expressible as well.

The abstract interpretation approach offers a versatile and expressive structure for capturing a wide range of logics. Many of these logics, including several if not all of the prominent ones, are organized in a 4-dimensional cube-like schema (see [Cousot 2024, Figure 3]). At first glance, this cube does not appear to resemble our taxonomy in Figure 5. However, a closer inspection reveals that the logics occupying the upper half of the cube once again correspond to those defined using the awp and asp transformers, as well as their contrapositives. The main distinction to the corresponding fragment of Figure 5 lies in how the logics are arranged.

When comparing the cube to our planar diagram, the sets at the cube's upper corners correspond to awp, asp, dwlp, and dslp for a concrete pre- or postcondition. The cube's arrangement emphasizes the symmetry between logics through *set inclusions*, aligning with the close connection between Hoare logic and incorrectness logic. This particular symmetry is less immediate in our diagram, which instead emphasizes the *implications* and *contrapositives* across different logics.

The logics in the lower half of Cousot's cube are more challenging to align with our taxonomy. These logics intuitively mirror their counterparts in the upper half but introduce additional conditions related to termination behavior, allowing, for instance, the expression of *demonic* partial correctness. However, since the interpretation of these lower-half logics depends on whether nontermination (represented by $\bot$) is included in the postcondition, they are somewhat incompatible with our logics. In particular, we do not consider incorrectness logics with termination constraints.

Notably absent from Cousot's cube are the two novel predicate transformers we proposed - demonic strongest post and angelic strongest liberal post - and the corresponding logics. However, the logics represented in the cube are only a subset of what is possible within the framework of abstract interpretation. We conjecture that these new logics could be accommodated by extending the framework to include an additional symbol representing unreachability, dual to $\bot$ for divergence.

Cousot additionally introduces what he calls *Hoare incorrectness logic*, which is included in the cube but does not neatly fit within its structure. This logic represents the negation of a standard Hoare triple. Specifically, $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for Hoare incorrectness if it is *not* valid for partial Hoare logic. Intuitively, this means that there exists an execution of p starting in b and terminating outside of c. Unlike traditional Hoare logic or incorrectness logic, Hoare incorrectness logic does not require conditions to hold for *all* states in the precondition or postcondition; it only requires the existence of a single execution path violating the postcondition. In our framework, this corresponds to the condition $\mathsf{awp}[\![p]\!](\overline{c}) \cap b \neq \emptyset$ [Verscht and Kaminski 2023]. So, while in principle expressible in our framework, Hoare incorrectness logic does not align with the structure of the other logics — just as it does not fit within the cube.

4.3 On the Impact of Additional Assumptions

It is well known that under the assumption of program *termination* (say on all initial states), standard partial and total correctness coincide, or rather *collapse* to one notion. More symbolically,

$$\text{termination} \quad \text{implies} \quad \text{partial corr.} \iff \text{total corr.}$$

In the following, we will inspect what other of the 16 logics from Figure 5 collapse under the assumption of termination, and we will explore three more natural assumptions which will make yet other logics collapse, namely *reachability*, *determinism*, and *reversibility*.

4.3.1 Termination. Liberality in weakest precondition transformers is about whether they deem nontermination acceptable behavior or not. If p terminates on all states, then dwlp and dwp coincide for all postconditions. The same goes for awlp and awp. Formally, we have the following theorem:

Theorem 4.1 (Termination Collapse). *Let p be a program that* must *terminate on all initial states. Then*

$$\mathsf{dwp}[\![p]\!](c) \;=\; \mathsf{dwlp}[\![p]\!](c) \qquad \text{and} \qquad \mathsf{awp}[\![p]\!](c) \;=\; \mathsf{awlp}[\![p]\!](c)\,.$$

Proof. See [Verscht and Kaminski 2024, Appendix E.2.1]. □

If two transformers t_1 and t_2 coincide, then two logics whose definitions are equal up to whether their definition invokes t_1 or t_2 immediately collapse into a single logic. Hence, for instance, partial and total correctness collapse to one notion. Logics that collapse are illustrated by the dashed boxes in Figure 6. We see that in the upper half, logics collapse along a horizontal axis. The bottom half

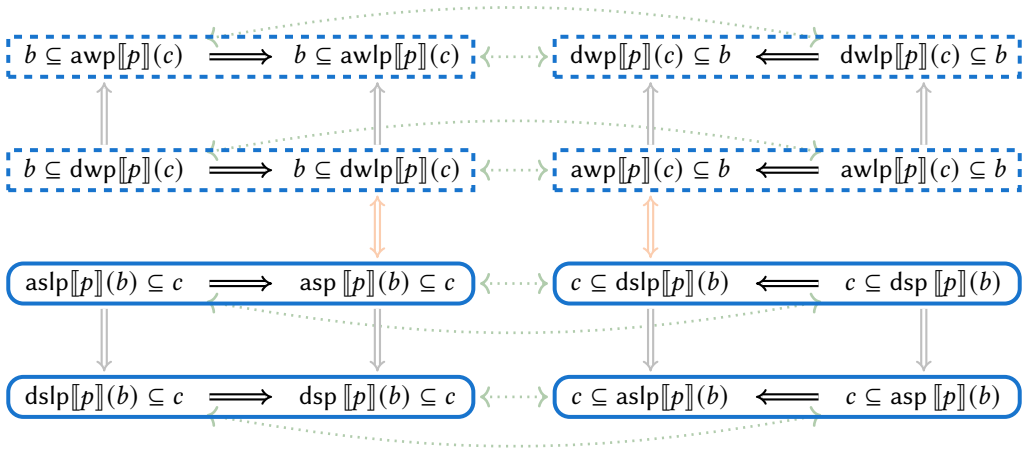


Fig. 6. Collapse of program logics under assumptions regarding totality and partiality. Blue *dashed* boxes enclose logics that collapse (i.e. the implications become equivalences) under *termination*. Blue *solid* boxes enclose logics that collapse under *reachability*.

of the logics is unaffected by termination, as strongest post based logics are incapable of reasoning about termination or divergence.

For the purpose of logics collapsing, note that we can loosen the requirement to computations starting in initial states of interest, i.e. those satisfying the precondition b. Concretely, if a program p terminates on all initial states satisfying b, then $b \subseteq \mathrm{dwp}[\![p]\!](c) \iff b \subseteq \mathrm{dwlp}[\![p]\!](c)$. The same holds for the other wp based logics.

We can moreover use the predicate transformers to express termination properties, which follows directly from Theorem 4.1.

Corollary 4.2 (Expressing Termination Properties). *A program p* must *terminate on all initial states*[3] *if and only if*

$$\mathrm{dwp}[\![p]\!](\mathrm{true}) = \mathrm{true} \qquad \text{or equivalently} \qquad \mathrm{dwlp}[\![p]\!](\mathrm{false}) = \mathrm{false}.$$

Theorem 4.1 and Corollary 4.2 together yield precisely the well-known technique of proving partial correctness and proving termination separately in order to obtain total correctness.

4.3.2 Reachability. For weakest preconditions, we saw that – under the assumption of termination – liberal and non-liberal weakest preconditions coincide. This immediately raises the question whether such a criterion can be found for strongest postcondition transformers to coincide. The answer is yes: *reachability*. If all (final) states are reachable from some initial state by some computation of p, then liberal and non-liberal strongest postconditions coincide:

Theorem 4.3 (Reachability Collapse). *Let all final states be reachable by some computation of a program p. Then*

$$\mathrm{asp}[\![p]\!](b) = \mathrm{aslp}[\![p]\!](b) \qquad \text{and} \qquad \mathrm{dsp}[\![p]\!](b) = \mathrm{dslp}[\![p]\!](b)\,.$$

[3] One could be a bit more fine-grained here and distinguish between *may* and *must* termination. While must implies may termination, we could only check for may termination, which is the case if and only if

$$\mathrm{awp}[\![p]\!](\mathrm{true}) = \mathrm{true} \qquad \text{or equivalently} \qquad \mathrm{awlp}[\![p]\!](\mathrm{false}) = \mathrm{false}.$$

In that case, only the angelic but not necessarily the demonic weakest precondition transformers would coincide (cf. Theorem 4.1) and in Figure 6 only the dashed boxes containing angelic transformers would be present.

Proof. See [Verscht and Kaminski 2024, Appendix E.2.2]. □

As in the case for termination, coincidence of transformers causes the associated logics to collapse. Logics that collapse for reachability are illustrated by the solid boxes in Figure 6. The top half of the logics is unaffected by reachability.

Reachability of *all* final states is a *very* strong (and perhaps sometimes even undesired) assumption. As a silver lining, for the collapse of logics, we can loosen this requirement to the final states satisfying c, similar to what we have seen for termination.

In Theorem 4.1, we had to assume that *all* computations of p terminate. In contrast to this, Theorem 4.3 only requires a final state to be reachable by *some* computation. Again, this difference is caused by Observation 2: A final state can either be unreachable or not, but never both.

As with termination, we can express reachability in terms of predicate transformers:

Corollary 4.4 (Expressing Reachability Properties). *All final states are reachable from some initial state by some computation of program p, if and only if*

$$\mathrm{dsp}\,[\![p]\!](\mathsf{true})(\tau) = \mathsf{true} \quad \text{or equivalently} \quad \mathrm{dslp}[\![p]\!](\mathsf{false})(\tau) = \mathsf{false} \quad \text{or equivalently}$$
$$\mathrm{asp}\,[\![p]\!](\mathsf{true})(\tau) = \mathsf{true} \quad \text{or equivalently} \quad \mathrm{aslp}[\![p]\!](\mathsf{false})(\tau) = \mathsf{false}.$$

Proof. See [Verscht and Kaminski 2024, Appendix E.2.3]. □

This interestingly differs from the analogous corollary for termination (Corollary 4.2), where we had to distinguish between *may* and *must* termination (and opted for must as it is more general). This is again rooted in Observation 2, as the existence of a path to a final state is equivalent to the final state being reachable.

As with total correctness, Theorem 4.3 and Corollary 4.4 together yield a technique for proving incorrectness by proving angelic partial incorrectness and reachability.

4.3.3 Determinism. So far, we have seen how to produce "horizontal collapses" in Figure 5 by assuming that total and partial predicate transformers coincide. Now, we will see that we can get similar results for angelic and demonic transformers, thus producing "vertical collapses". For weakest preconditions, angelic and demonic weakest preconditions differ in their treatment of explicit nondeterministic branching of the program. Therefore, it is evident that when restricting to *deterministic* programs (even syntactically), they should be equivalent.[4]

Theorem 4.5 (Determinism Collapse). *If program p is* deterministic *then*

$$\mathrm{awp}[\![p]\!](c) = \mathrm{dwp}[\![p]\!](c) \qquad \text{and} \qquad \mathrm{awlp}[\![p]\!](c) = \mathrm{dwlp}[\![p]\!](c)\,.$$

Proof. See [Verscht and Kaminski 2024, Appendix E.2.4]. □

In Figure 7, we now see what we were expecting: The top row of our taxonomy collapses into the second but top row under the assumption of determinism, symbolized by the blue dashed boxes. As before, for the collapse it suffices to require determinism of computation started in b.

Ascari et al. [2024, Prop. 5.5] established that angelic total correctness (their sufficient incorrectness logic) and demonic partial correctness (Hoare logic) are equivalent for deterministic programs which terminate. By looking at the bigger picture, we see why this holds: Combining Theorems 4.1 and 4.5, all four logics in the upper left (as well as the upper right) quadrat collapse, including the two logics mentioned above.

[4]Of course, determinism is also a semantic property, but we will not go into this as syntactic determinism obviously implies semantic determinism and semantic determinism is difficult to reason about.

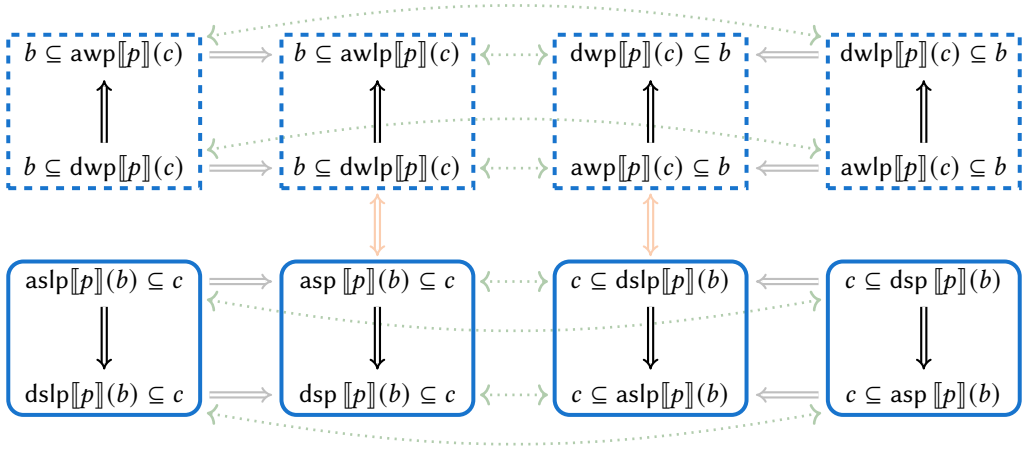


Fig. 7. Collapse of program logics under certain assumptions, part 2. Blue *dashed* boxes enclose logics that collapse (i.e. the implications become equivalences) under *determinism*. Blue *solid* boxes enclose logics that collapse under *reversibility*.

4.3.4 Reversibility. Finally, we are missing to collapse the bottom row of Figure 7 into the second but last row. This can be achieved by ensuring "backward determinism" of p, meaning that every final state could have only been reached from (at most) one initial state. In other words, the computation of the program is *reversible*. This is of interest, for example, in compression algorithms or quantum computations. Ascari et al. [2024] observed that under reversibility, incorrectness logic implies demonic partial incorrectness. This can be seen as a consequence of the following theorem:

Theorem 4.6 (Reversibility Collapse). *It p is a* reversible *program (i.e. $[\![p]\!]^{-1}(\tau)$ is either a singleton or the empty set for all $\tau \in \Sigma$), then*

$$\mathrm{asp}\,[\![p]\!](b) \;=\; \mathrm{dsp}\,[\![p]\!](b) \qquad \text{and} \qquad \mathrm{aslp}[\![p]\!](b) \;=\; \mathrm{dslp}[\![p]\!](b).$$

Proof. See [Verscht and Kaminski 2024, Appendix E.2.5]. □

Analogously to the other cases, reversibility does not effect the top half of the logics for the same reason that (non)determinism has no effect on the lower half. Also, we can again weaken the requirement to reversibility of computation terminating in c.

4.4 Symmetries and Asymmetries

Program correctness and incorrectness are two sides of the same coin.
— Peter O'Hearn [2019]

Figure 5 is at first glance full of symmetry and duality: The upper half contains all weakest precondition based logics, the lower half dually contains all strongest postcondition based logics. Also, the left half is a contrapositive mirroring of the right half. The assumptions discussed in the preceding section also act completely symmetrically. On the top left we have "correctness logics", on the bottom right we have "incorrectness logics". Indeed, this all seems like two opposite sides of a (multidimensional) coin.

Two sides of the same coin? Not quite. However, when taking a closer look, this fully symmetric picture starts to become a bit brittle. First of all, correctness and incorrectness are not really at

opposite sides of the "coin" represented by Figure 5. If total correctness (second row left), was truly on the opposite side of incorrectness logic (bottom row right), we should either expect the standard notion of program correctness to be *Lisbon logic* (top row left), or alternatively the standard notion of incorrectness to be *demonic incorrectness* (third row right). It seems off that the current standard notions of correctness and incorrectness are *not* really at opposite sides of Figure 5.

Missing Galois connections. Amongst the logics in Figure 5, there is *essentially only one* Galois connection (there are two, but one is the contrapositive of the other). In particular, since there is a Galois connection amongst the two transformer-based definitions of partial correctness, one might expect more Galois connections, e.g. between *angelic* partial correctness and the contrapositive of angelic partial incorrectness. Such Galois connection does not exists, however: Assume that $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is valid for angelic partial correctness, i.e. $b \subseteq \mathsf{awlp}[\![p]\!](c)$. So all states in b must either have a diverging path or a path to c. Let $\tau \in c$ be a final state which is exclusively reachable from $\overline{b}$. This does not violate the assumption of angelic partial correctness. However, the contrapositive of angelic partial incorrectness, $\mathsf{dsp}\,[\![p]\!](\overline{b}) \subseteq \overline{c}$ requires all states that are exclusively reachable from $\overline{b}$ to be included in $\overline{c}$. This implies that $\tau \in \overline{c}$, which contradicts the previous assumption. Hence, $\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is *not* valid for the contrapositive of angelic partial incorrectness. Similar counterexamples exist for all other combinations of triples (see [Verscht and Kaminski 2024, Appendix B] for a collection). We conclude that there is (essentially) only one Galois connection between the logics we considered. We can, however, force such Galois connections by additional assumptions. E.g., the above discussed Galois connection *is* valid for programs that are both *deterministic and reversible*.

Missing Asymmetries. We discussed an intrinsic asymmetry of forward and backward analyses in Observation 2. This asymmetry is not (yet) visible in Figure 5. To gain more insights on the logics' relationships and (a)symmetries, we will now take another look at program properties from the perspective of Kleene algebras.

5 Kleene Algebra with Top and Tests

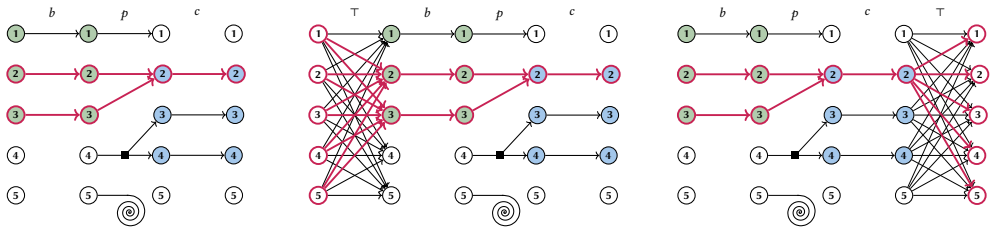
*Kleene Algebra with Tests (*KAT*).* Introduced by Kozen [1997], KAT is an algebraic approach to specifying and reasoning about program properties. For us, it will suffice to describe KAT at a rather high level. For reference, the formal definitions we require can be found in [Verscht and Kaminski 2024, Appendix C].

KAT terms are generalized regular expressions over a *two-sorted alphabet* consisting of (i) programs ($p, q, \ldots$) and (ii) tests ($b, c, \ldots$). We interpret these symbols as relations: A program p relates (or maps) initial states to final states through its execution. Initial states on which p *must* diverge are not related to any final states. Dually, unreachable final states are not related to any initial states.

A test b maps initial states that satisfy b to themselves. All other states are not contained in the relation. Hence, tests act as filters. Testing for false (the *empty* relation) is denoted by 0 and is the least element in the lattice of relations (ordered by set inclusion).

Composing symbols corresponds to composing relations. For example, the term $bpqc$ intuitively means: First test for b, then execute p, then execute q, and finally test for c. Executions that fail to satisfy b initially or c finally are filtered out and do not become part of the resulting relation.

*Kleene Algebra with Top and Tests (*TopKAT*).* In TopKAT [Zhang et al. 2022], an additional $\top$ element (the *universal* relation relating *all* states with each other) is added to KAT. Notice the difference to the identity relation, denoted 1, relating every state to itself. $\top$ can be used to "select" the domain or codomain of a relation, as the following example illustrates.



(a) Illustration of a TopKAT term *bpc*. The resulting relation is highlighted in bold red.

(b) Adding ⊤ to the left-hand side of the term *bpc* from Figure 8a, effectively selecting the *codomain* of the underlying relation.

(c) Adding ⊤ to the right-hand side of the term *bpc* from Figure 8a, effectively selecting the *domain* of the underlying relation.

Fig. 8. Using ⊤ to select the codomain (b) or the domain (c) of a TopKAT term *bpc*.

Example 5.1 (⊤ as (Co)domain Selector). Consider a precondition b, a program p, and a postcondition c over a state space of five states $\Sigma = \{1, 2, 3, 4, 5\}$, where

$$b = \{(1,1), (2,2), (3,3)\},$$
$$p = \{(1,1), (2,2), (3,2), (4,3), (4,4)\}, \quad \text{and}$$
$$c = \{(2,2), (3,3), (4,4)\}.$$

As described above, the KAT term *bpc* corresponds to the composition of the underlying relations, in this example being

$$bpc = \{(2,2), (3,2)\}.$$

This is illustrated in Figure 8a. The initial states satisfying b are green, the final states satisfying c are blue. Nondeterministic choices in p are visualized by a square and divergence by a spiral. The composed relation *bpc* is highlighted in red: We can see that the pairs in the relation correspond to the paths through the graph, originating in either initial state 2 or 3 and leading to final state 2. Intuitively, these are the executions of p starting in b and terminating in c.

The effect of appending ⊤ (i.e. the universal relation) on the left of *bpc* is visualized in Figure 8b and yields the relation

$$\top bpc = \{(1,2), (2,2), (3,2), (4,2), (5,2)\} = \{(\sigma, 2) \mid \sigma \in \Sigma\}.$$

Whereas in *bpc* only initial states 2 and 3 were related to final state 2, in $\top bpc$ *all* initial states are related to 2. Appending ⊤ on the left thus in some sense erases the information about the initial states and leaves only information about the final states – or in other words: the *codomain*.

Dually, the effect of appending ⊤ on the right is visualized in Figure 8c and yields the relation

$$bpc\top = \{(2,\tau), (3,\tau) \mid \tau \in \Sigma\},$$

Whereas in *bpc* initial states 2 and 3 were related only to final state 2, in $\top bpc$ initial states 2 and 3 are related to *all* final states. Appending ⊤ on the right thus erases the information about the final states and leaves only information about the initial states – or in other words: the *domain*. ◁

As we have seen, appending ⊤ to the right (left) of *any* KAT term amounts to selecting the (co)domain of the underlying relation. In particular, for any two KAT terms s and t, we have that

$$\top s = \top t \quad \text{iff} \quad \text{the codomains of } s \text{ and } t \text{ are equal}, \quad \text{and}$$
$$s\top = t\top \quad \text{iff} \quad \text{the domains of } s \text{ and } t \text{ are equal}.$$

For expressing incorrectness logic, an explicit comparison of codomains is necessary. Hoare logic, on the other hand, can also be expressed without $\top$. For details, we refer to [Zhang et al. 2022].

TopKAT *and Predicate Transformers.* There is a close relation between TopKAT and predicate transformers namely that some predicate transformers can be expressed as TopKAT terms. Let us express, for instance, asp $[\![p]\!](b)$ in TopKAT. Consider for this first the term $bp = \{(\sigma, \tau) \mid \sigma \in b \text{ and } (\sigma, \tau) \in p\}$ describing all executions of p that start in b. As described above, appending $\top$ on the left selects the codomain of that term, i.e.

$$\top bp \;=\; \{(\sigma', \tau) \mid \sigma' \in \Sigma \text{ and } \exists\, sigma \in b \text{ and } (\sigma, \tau) \in p\}\ .$$

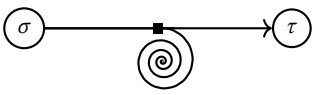
Since asp $[\![p]\!](b)$ is precisely the set of states reachable by executing p on initial states satisfying b and the codomain of bp is also precisely that set, we can morally equate asp $[\![p]\!](b)$ and $\top bp$.

Similarly, appending $\top$ on the right selects the domain and $pc\top$ hence describes the set of states that can reach c:

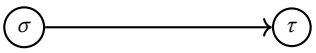
$$pc\top \;=\; \{(\sigma, \tau') \mid \tau' \in \Sigma \text{ and } \exists\, tau \in c \text{ and } (\sigma, \tau) \in p\}.$$

The domain of pc is precisely the set awp$[\![p]\!](c)$. Notably, asp and awp are the only two transformers that are directly expressible in a TopKAT term.

The Relational Perspective and Divergence. The purely relational perspective on programs introduces some limitations, particularly in reasoning about divergence. An initial state from which computation always diverges is related to no final state, making it distinguishable from states that lead to some final state. However, consider an initial state σ from which the computation can nondeterministically diverge or terminate in τ, as illustrated by the top program p_1 to our right. The relation corresponding to p_1 contains only the pair (σ, τ).



$p_1 = \{(\sigma, \tau)\}$



$p_2 = \{(\sigma, \tau)\}$

Now, consider the bottom program p_2 to our right which *always* terminates in τ when started in σ. The relational representation of p_2 is the same as the one of p_1, making these two programs relationally indistinguishable. Consequently, information about branching divergence is inevitably lost in a relational perspective, as was already observed in Remark 1.

5.1 Expressing Program Logics in TopKAT

Let us now explore how to express the program properties / logics of Figure 5 in the equational system of TopKAT.

Hoare Logic for Partial Correctness. Partial correctness requires that all computation started in b can only terminate in c. This can be expressed in KAT, for example, by $bp\overline{c} = 0$ or $bpc = bp$. The first equation intuitively states that there is no computation starting in b and terminating outside of c. The latter compares all states in c in which p can terminate starting from b to the states in which p can terminate from b at all. In general, the equations do not uniquely express partial correctness. In fact, there are many more (in)equations characterizing partial correctness, in particular the TopKAT equation $\top bpc = \top bp$. We choose the latter as it aligns well with the equations for the other logics.

Partial Incorrectness. Partial *in*correctness is contrapositive to partial correctness and thus immediately both KAT and TopKAT expressible by negating the conditions: $\top \overline{b} p \overline{c} = \top \overline{b} p$. An equivalent equation, however, is $bpc\top = pc\top$ and we will consider this latter one.

Table 2. Overview of TopKAT expressible program logics.

angelic total correctness	$b \subseteq \mathsf{awp}[\![p]\!](c)$	$bpc\top = b\top$
demonic partial correctness	$b \subseteq \mathsf{dwlp}[\![p]\!](c)$	$\top bpc = \top bp$
angelic total incorrectness	$c \subseteq \mathsf{asp}\,[\![p]\!](b)$	$\top bpc = \top c$
angelic partial incorrectness	$c \subseteq \mathsf{aslp}[\![p]\!](b)$	$\top bpc = \top pc$
demonic partial incorrectness	$c \subseteq \mathsf{dslp}[\![p]\!](b)$	$bpc\top = pc\top$
???	???	$bpc\top = bp\top$

Incorrectness Logic. Incorrectness logic requires that all states in c be reachable from b. This cannot be captured in a standard KAT equation, as it involves reasoning about the codomain of relations. Given that overcoming this was a key motivation behind TopKAT, it is not surprising that incorrectness logic *is* expressible in TopKAT [Zhang et al. 2022], namely by $\top bpc = \top c$. The left-hand side of the equation selects all final states in c reachable by executing p on b. The right-hand side selects *all* final states in c.

Hoare Logic for Total Correctness and Angelic Partial Correctness. For total correctness, all computation started in b must terminate and do so in c. This requires reasoning about branching divergence, which, as argued earlier, is impossible. Thus, standard total correctness is inexpressible both in KAT [von Wright 2002] and TopKAT. The same goes for angelic partial correctness.

Lisbon logic. Lisbon logic expresses that from all initial states in b, it is *possible* for p to terminate in c. As this requires reasoning about the domain of a relation, Lisbon logic cannot be expressed in KAT. However, it *is* indeed expressible in TopKAT and to the best of our knowledge a *novel result*:

Theorem 5.2 (Expressibility of Lisbon Logic (Angelic Total Correctness) in TopKAT).

$\langle\, b \,\rangle\ p\ \langle\, c \,\rangle$ is a valid Lisbon triple / valid for angelic total correctness iff $bpc\top = b\top$

Demonic Incorrectness. The novel logic demonic incorrectness requires all states in c to be exclusively reachable from b. We can divide this into two requirements, namely *partial* incorrectness ($bpc\top = pc\top$) and reachability of all states in c ($\top c = \top pc$). These two cannot be expressed as a single equation.

Angelic Partial Incorrectness. The second novel logic, angelic partial incorrectness, requires that all states in post are either unreachable or can be reached from b. This is expressible in KAT by $bpc = pc$ and in TopKAT by the equivalent equation $\top bpc = \top pc$, comparing all final states in c reachable from b to the reachable fragment of c. This is somewhat surprising (and asymmetric), since angelic partial *correctness* cannot be expressed. The root cause is, again, Observation 2. An overview of the five TopKAT expressible program logics is given in Table 2. Their contrapositives are of course also expressible by negation of all tests. All equations are syntactically very similar. There are more equations following this pattern, one of which is shown in the sixth row of Table 2. We will discuss this logic in the following section. While other equations also syntactically fit into the scheme, we exclude them from further investigation as they would, for example, interpret the precondition b over final states or do other semantically nonsensical things.

5.2 The In-Between Logics

Table 2 shows that five of the six basic TopKAT equations directly correspond to a predicate transformer-based logic, as outlined in Section 4. However, the sixth equation, $bpc\top = bp\top$, stands apart. This equation expresses that, for all states in b, the program either *always* diverges, or there

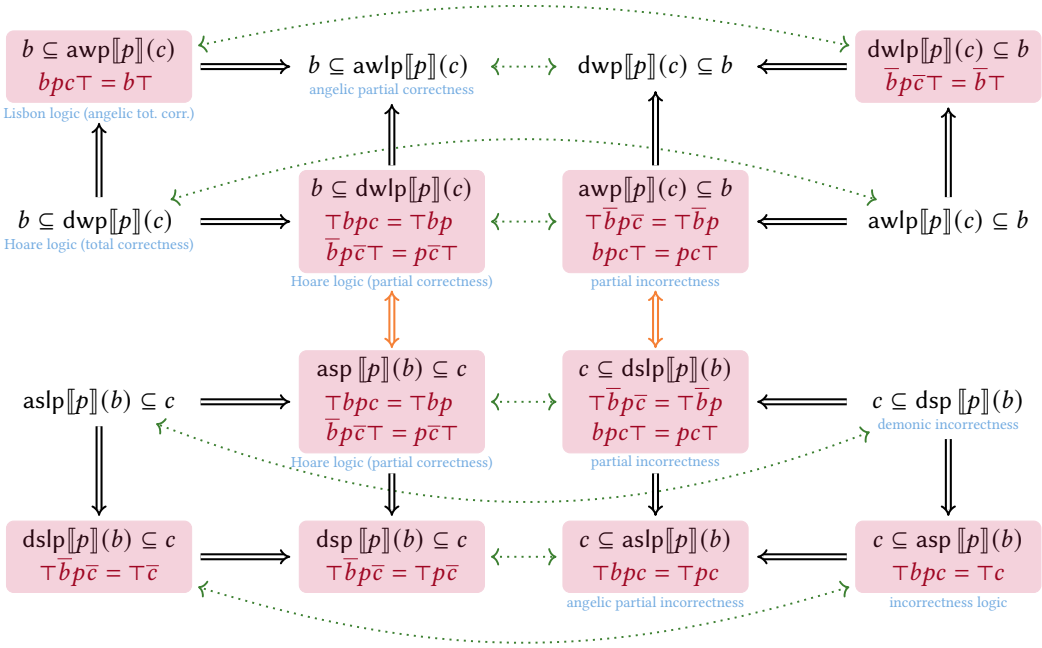


Fig. 9. The taxonomy presented in Figure 5 with corresponding TopKAT equations, if existing, in red.

exists a terminating path to c. Interestingly, in Section 3.3, we discussed a set that characterizes precisely such program states: $\mathrm{awp}[\![p]\!](c) \cup \mathrm{dwlp}[\![p]\!](c)$. Therefore, when the precondition b is chosen to underapproximate this set, we obtain:

$$bpc\top = bp\top \qquad \text{iff} \qquad b \subseteq \mathrm{awp}[\![p]\!](c) \cup \mathrm{dwlp}[\![p]\!](c)$$

In Section 3.3, we also examined the intersection $\mathrm{awp}[\![p]\!](c) \cap \mathrm{dwlp}[\![p]\!](c)$. Underapproximating this intersection yields a logic which combines partial correctness Hoare logic and Lisbon logic. Notably, this is a special case of outcome logic [Zilberstein et al. 2023], when restricted to traditional assertions, i.e. predicates, and instantiated to the powerset monad. Similar to the equations for demonic total incorrectness, we cannot give a single TopKAT equation for this, but a *system* of two: $bpc \neq 0$ and $bp\overline{c} = 0$.

6 A Taxonomy of Program Logics: Revisited

In Section 4, we presented a taxonomy of predicate transformer logics. Taking that picture and highlighting the TopKAT expressible logics in red yields Figure 9, *revealing an asymmetry* which was invisible before: The pattern of TopKAT expressible logics in the upper half is not symmetric to the pattern in the lower half: This asymmetry is attributed to the fact that we cannot capture branching divergence, whereas the dual concept in forward analyses, confluence of unreachability, does not exist (see Observation 2).

6.1 Adding the In-Between Logics

Missing from Figure 9 are the logics that arise from unions and intersections of predicate transformers, as discussed in Section 5.2. In Figure 10, we include these logics at the center of each quadrant. For wp transformers, this yields indeed new logics. For sp transformers, the resulting logics are

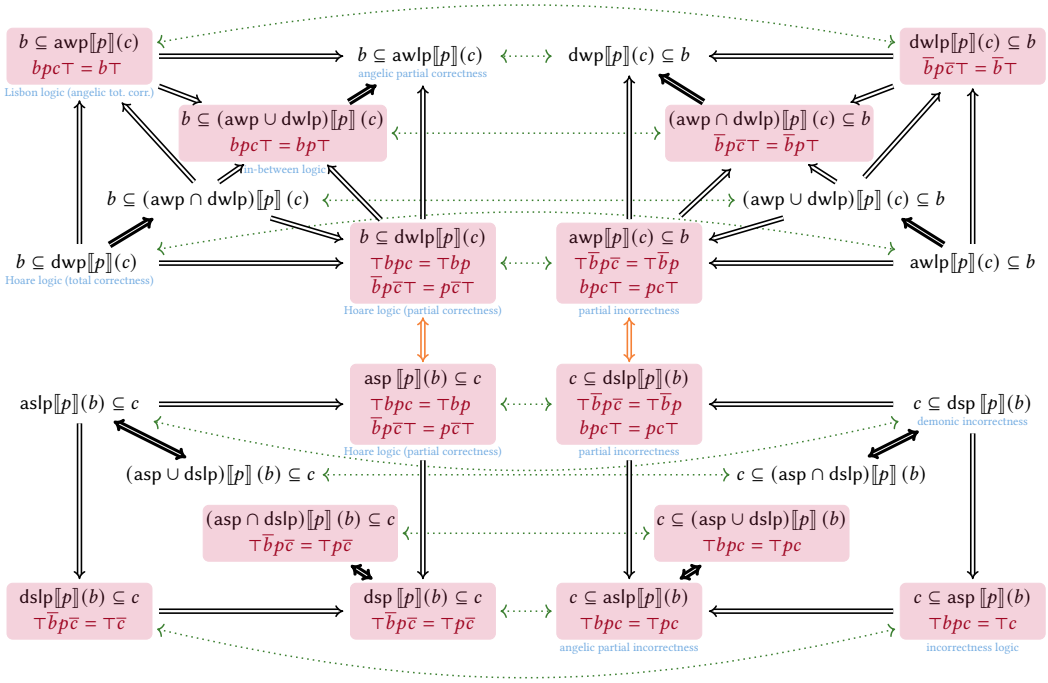


Fig. 10. The taxonomy including union and intersection logics and the TopKAT equations, if existing, in red.

equivalent to existing ones, see Section 3.3. Notably, neither intersections nor unions of any sp transformers generate new logics, nor do any of the wp transformers, apart from awp and dwlp. In the complete picture, the source of the earlier-mentioned asymmetry becomes clearer: The bold connectives in the top half represent only implications, while those in the lower half represent equivalences, a distinction once again driven by Observation 2.

6.2 Absence of Branching Divergence

In Section 4.3, we examined how assumptions on p let our taxonomy partially collapse. With the addition of union and intersection logics, we can now consider another assumption: The absence of branching divergence. If we assume that branching divergence is not present, the taxonomy becomes fully dual, as the bold implications in the upper half of the diagram turn into equivalences, see Figure 11, where the logics added in the center collapse into the corners. Meanwhile, the bottom half remains unchanged, as the logics within the blue dashed lines were already equivalent.

Theorem 6.1 (Branching Divergence Collapse). *Let p be a program and c be a postcondition. If computation of p either always diverges or always terminates, we have*

$$\mathsf{dwp}[\![p]\!](c) = \mathsf{awp}[\![p]\!](c) \cap \mathsf{dwlp}[\![p]\!](c) \qquad \text{and} \qquad \mathsf{awlp}[\![p]\!](c) = \mathsf{awp}[\![p]\!](c) \cup \mathsf{dwlp}[\![p]\!](c).$$

Proof. See [Verscht and Kaminski 2024, Appendix E.3.1]. □

6.3 On the Semantics of Syntactic TopKAT Transformations

Figure 10 shows TopKAT equations for each logic, if expressible. As mentioned earlier, the equations are syntactically very similar (see Section 5). In fact, the equations can be transformed into others

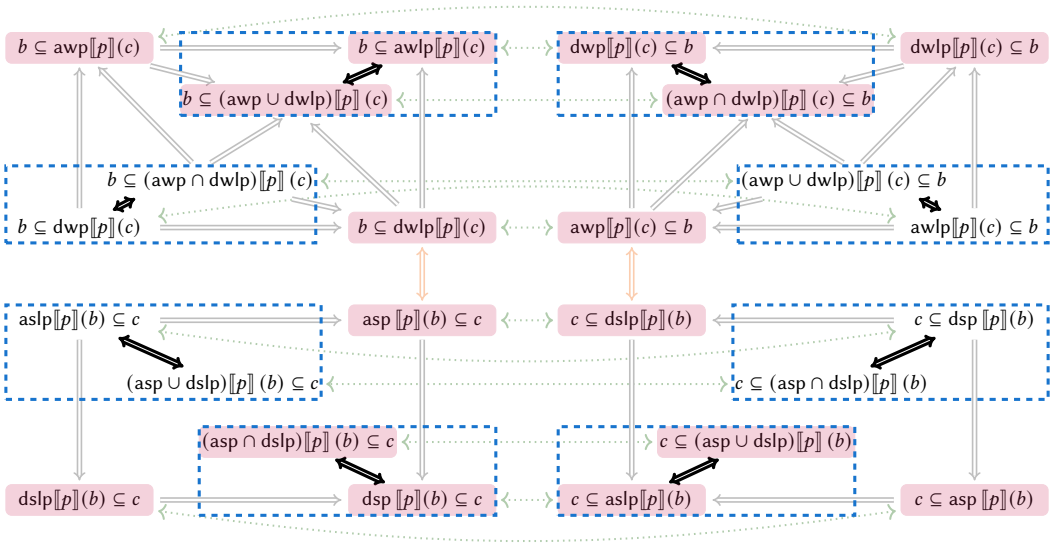


Fig. 11. Collapse of program logics under certain assumptions, part 3. Blue *dashed* boxes enclose logics that collapse (i.e. the implications become equivalences) under *absence of branching divergence*. Note that the logics in the lower half are equivalent in general, we mark them to demonstrate the symmetry.

by systematical syntactic means. Consider, for instance, $\top bpc = \top bp$ for partial correctness. When moving $\top$ from left to right, we obtain $bpc\top = bp\top$ for the in-between logic (see Section 5.2). However, just switching $\top$-sides in itself is not a meaningful transformation: Switching sides on $bpc\top = b\top$ (Lisbon logic) yields $\top bpc = \top b$: By comparing *co*domains, this equation would somehow treat the *pre*condition b as a set of *final* states (i.e. a *post*condition) which is not meaningful.

Another transformation is adding or removing p on the right-hand side of the equation (depending on whether or not p is already present or not). For example, from $\top bpc = \top c$ (incorrectness logic) we obtain $\top bpc = \top pc$ (angelic partial incorrectness). Since adding (removing) p can be seen as a filtering (or not) of unreachable states, this seems like a meaningful transformation. However, from $\top bpc = \top bp$ (partial correctness) we obtain $\top bpc = \top b$, which again interprets b as a postcondition. Consequently, when searching for meaningful syntactic transformations, we must ensure that b is a pre- and c is a postcondition. We propose hence the following set of meaningful transformations:

(t1) Switch the $\top$-side *and* switch between b and c (or bp and pc) on the right-hand side. This corresponds to switching between incorrectness and correctness reasoning. For example, we get from partial *correctness* ($\top bpc = \top bp$) to partial *incorrectness* ($bpc\top = pc\top$). In Figure 10, this corresponds to going from the upper to the lower part.

(t2) Switching from $*\top$ to $*p\top$ or $\top*$ to $\top p*$. This corresponds to switching from liberal to non-liberal reasoning, which makes sense as adding the program to the equation concentrates the reasoning on reachable states or dually states from which computation terminates. By requiring $\top$ to be a part of the equation, we avoid the issue described above and ensure that pre- and postconditions are interpreted correctly.

(t3) Switching from bp to pc and negating all conditions. Broadly speaking, this corresponds to switching between angelic and demonic resolution of nondeterminism. For example, we go from *angelic* partial incorrectness ($\top bpc = \top pc$) to *demonic* partial incorrectness ($\top \overline{b}p\overline{c} = \top \overline{b}p$). However, starting with partial correctness, for example, we know that the

corresponding angelic variant is inexpressible in TopKAT. Applying this syntactic transformation on the equation $\overline{b}p\overline{c}\top = p\overline{c}\top$ for angelic total correctness, we end up with $bpc\top = pc\top$. This characterizes the in-between logic, which under the assumption of nonexistence of branching divergence is equivalent to partial correctness. So, even though we do not end up exactly with the demonic variant, we end up somewhere "close".

Applying combinations of these three transformations, we can get from each logic in Figure 9 to any other logic. Additionally, the TopKAT expressible logics are closed under t1 – t3. This speaks in favor of the meaningfulness of these transformations. Nevertheless, the transformations considered appear still somewhat arbitrary to us and we wonder whether there is more underlying structure that we can find in these syntactic transformations.

7 Open Questions

Complexity of Weakest Pre- and Strongest Postcondition Analyses. Throughout this paper, we compared analyses using wp with analyses using sp. On several occasions, it appeared as if one is simpler than the other. For example, in Section 3, we saw that there are no inductive definitions for dsp and aslp. This suggests that these transformers are harder to compute. Is there an intuitive reason for this limitation? Are there also wp transformers for which no inductive definitions exists?

On the other hand, in Figure 1, we distinguished *seven* coreachability classes related to wp, but only *four* reachability classes related to sp. Also in Figure 10, the structure of the bottom (i.e.: the sp) half, seems simpler. This in turn leaves the impression that wp is more complex than sp.

A related observation is that we can characterize each reachability class in Figure 3 using (Boolean combinations of) sp transformers:

(1) is dsp.
(2) is asp without dsp.
(3) is dslp without dsp.
(4) are all states without aslp.

Making a similar list for coreachability classes using wp transformers is not possible: Classes (3) and (5) in Figure 1 are always either contained in any particular wp transformer or not, as evidenced by the fact that they are always either both colored green or both left white. The difference between (3) and (5) is that, while computation can terminate both inside and outside of the postcondition, (5) additionally allows divergence. Is there a sensible transformer that we can define to distinguish classes (3) and (5)? We also wonder whether there is a deeper reason as to why the full separation is possible for coreachability classes, but not for reachability classes.

Additionally, the sp transformers are complete in the sense that combining them does not yield any new logics, as can be seen in Figure 10. This is not the case for the wp setting, validated by the in-between logics presented in Section 5.2, which were defined as the union and intersection of transformers. There seem to be more intricacies in the wp setting than in the sp one, but this seems difficult to pin down.

Kleene Algebraic Reasoning. While numerous TopKAT equations express the same program property, determining which should constitute the "canonical" equations remains an open challenge. We chose one such a set in Section 5. Based on these, we saw that there are some logics which *cannot* be expressed in TopKAT, but *can* be expressed using predicate transformers. Can we thus infer that predicate transformers are more powerful than TopKAT? Addressing this requires an argument on why a certain set of TopKAT equations is indeed canonical. Additionally, as previously noted in Section 5, there exist extensions of Kleene algebra that can handle divergence. With such an algebra, we might get new insights into the expressiveness of KAT compared to predicate transformers.

Four logics were expressible using predicate transformers but required a *system* of two TopKAT equations, see Sections 5 and 6. Are these logics inherently more complex than others, and if so, in what sense? Are there formal arguments we can find for the complexity of program logics?

Galois Connections in TopKAT. We have seen that there exist Galois connections between some of the logics we have considered. These Galois connections do not seem to be visible yet in the TopKAT equations. We wonder whether there exists *syntactical* transformations of TopKAT equations that correspond to the Galois connections, perhaps under a different set of canonical TopKAT (in)equations. In this context, we also wonder whether the *nonexistence* of other Galois connections can also be made visible somehow in TopKAT.

Algebraic Reasoning about Divergence. Kleene algebra was extended to structures such as demonic refinement algebra [von Wright 2004] or omega algebra [Cohen 2000] in order to facilitate the analysis of nontermination. Particularly interesting for our setting is *weak omega algebra* [Möller and Struth 2005], which always has a top element and thus fits well into the setting of TopKAT. For a more elaborate discussion of the algebraic treatment of divergence, we refer to [Jules et al. 2011]. In future work, we aim to explore how our findings can be extended to algebras capable of more nuanced handling of nontermination.

Deriving Proof Rules. Most of the predicate transformers presented in this paper are accompanied by a set of inductive rules. For the novel transformers dsp and aslp, establishing such rules is more challenging but possible, as discussed in Section 3.3. Cousot [2024] demonstrates how proof systems for program logics can be constructed via abstractions of the semantics. It would be interesting to explore whether proof rules could similarly be derived from TopKAT equations.

Healthy Transformers. In theory, the seven coreachability classes give rise to $2^7 = 128$ wp transformers, and the four reachability classes give rise to $2^4 = 16$ sp transformers. Which of these are sensible or meaningful? Dijkstra assessed the meaningfulness of predicate transformers based on *healthiness conditions* like strictness, monotonicity, conjunctiveness, etc. We wonder how many and which ones of the wp and sp transformers meet these criteria.

Relation to Runtime Transformers. An extension of weakest pre transformers is the expected runtime transformer [Kaminski et al. 2018]. In accordance with Corollary 4.2, the expected runtime transformer can be used to prove termination of programs. An interesting open question is whether a similar approach can be defined for reachability. One idea would be to use a strongest postcondition style transformer to compute the expected number of steps required to reach a state. If this number is finite, we might conclude that the state is reachable.

The Coin. Finally, in Section 4.4, we spoke about the asymmetries in the taxonomy Figure 5. We have yet to find a satisfactory answer to why the "two sides of the same coin" are not mirrored.

8 Conclusion

We have provided a systematic overview of program logics defined by predicate transformers and Kleene algebra with top and tests. Our graphical illustrations clarify the relationships among various logics. A main point of interest was the symmetries and asymmetries between forward and backward reasoning. Notably, we found that many asymmetries could be traced back to one main observation: Running a nondeterministic program on some initial state can *both* reach some final state *and* diverge. But no final state can be both reachable from somewhere and at the same time unreachable. In other words: a nondeterministic computation has the potential to lead to somewhere or nowhere, but it cannot at the same time originate from somewhere or from nowhere. We call this the *absence of reachability confluence* (Observation 2).

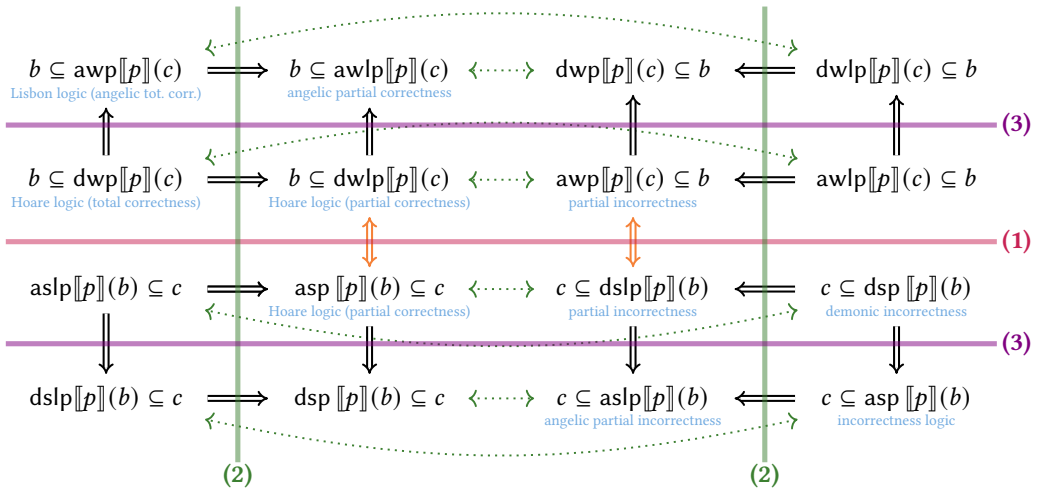


Fig. 12. A taxonomy of predicate transformer-based program logics with axes (1) to (3) corresponding to the dimensions of program logics.

Furthermore, we introduced new predicate transformers – angelic strongest and demonic strongest liberal postconditions – as well as novel logics involving union and intersection of transformers. Thereby, we filled some gaps in the landscape of program logics which seemed to naturally arise when taking the Kleene algebraic view. Additionally, we discussed in Section 4.3 how assumptions about program properties, such as determinism or the reachability of final states, influence the taxonomy. As conjectured at the very beginning, we can indeed identify three dimensions of program logics, each corresponding to an axis in Figure 12:

(1) *correctness* (being able to reach) vs. *incorrectness* (being reachable)
(2) *totality* vs. *partiality*
(3) *angelic* vs. *demonic* resolution of nondeterminism

As discussed in Section 4.3, if we assume that totality and partiality coincide, i.e. if p always terminates and all states are reachable, the logics collapse along the vertical axis (2). Dually, if we assume that p is deterministic and reversible, the logics collapse along the horizontal axis (3).

Apart from being of theoretical interest, the examination of the effect of assumptions is a step towards practical tools: We explore conditions that have to be discharged, so that different logics happen to collapse. The classical example of such a condition is

$$\text{partial correctness} + \text{termination} = \text{total correctness}.$$

If we have a partial correctness proof, we "merely" have to prove termination to obtain a total correctness proof. This is practically relevant because partial correctness is a lower bound on a greatest fixed point which can be discharged with invariant-based reasoning. Total correctness, on the other hand, is a lower bound on a least fixed point, which is much harder to discharge. Separation of concerns into partial correctness and termination aids to make proving total correctness more tractable. In a similar manner, we have

$$\text{partial incorrectness} + \text{reachability} = \text{incorrectness}.$$

Exactly the same least/greatest fixed point considerations apply to partial and "total" incorrectness. Hence, partial incorrectness logic is easier to discharge and we obtain that an additional reachability proof would give us "total" incorrectness.

For Kleene algebra with top and tests, we investigated the relationship between TopKAT expressible logics and predicate transformer logics. In the course of this, we showed that we can express Lisbon logic (angelic total correctness) in TopKAT. We also saw that in Table 2, there is a basic TopKAT equation which does not directly correspond to a predicate transformer equation. However, we showed that this equation can be expressed by combining predicate transformers in Section 5.2. This suggests that predicate transformers are stronger in the sense that all TopKAT equations are expressible using predicate transformers, but not the other way around. This is, however, due to the limitations of the chosen TopKAT approach and could be fixed by including some mechanism for identifying divergence.

Acknowledgments

We would like to thank Kevin Batz and Philipp Schröer for the valuable discussions on practical implications of this paper, as well as the anonymous reviewers for their very constructive and valuable feedback. This work was partially supported by the ERC Advanced Research Grant FRAPPANT (grant no. 787914).

References

Flavio Ascari, Roberto Bruni, Roberta Gori, and Francesco Logozzo. 2024. Sufficient Incorrectness Logic: SIL and Separation SIL. arXiv:2310.18156 [cs.LO] https://arxiv.org/abs/2310.18156

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. 2023. A correctness and incorrectness program logic. *J. ACM* 70, 2 (2023), 1–45. https://doi.org/10.1145/3582267

Ernie Cohen. 2000. Separation and reduction. In *International Conference on Mathematics of Program Construction*. Springer, 45–59. https://doi.org/10.1007/10722010_4

Patrick Cousot. 2024. Calculational Design of [In]Correctness Transformational Program Logics by Abstract Interpretation. *Proc. ACM Program. Lang.* 8, POPL, Article 7 (Jan. 2024), 34 pages. https://doi.org/10.1145/3632849

Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Los Angeles, California) *(POPL '77)*. Association for Computing Machinery, New York, NY, USA, 238–252. https://doi.org/10.1145/512950.512973

Patrick Cousot, Radhia Cousot, Manuel Fähndrich, and Francesco Logozzo. 2013. Automatic Inference of Necessary Preconditions. In *Verification, Model Checking, and Abstract Interpretation*, Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 128–148. https://doi.org/10.1007/978-3-642-35873-9_10

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

Ellie D'hondt and Prakash Panangaden. 2006. Quantum weakest preconditions. *Mathematical Structures in Computer Science* 16, 3 (2006), 429–451. https://doi.org/10.1017/S0960129506005251

Edsger W. Dijkstra. 1975. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Commun. ACM* 18, 8 (1975), 453–457. https://doi.org/10.1145/360933.360975

Edsger W. Dijkstra. 1976. *A discipline of programming*. Vol. 613924118. Prentice Hall PTR.

Edsger W. Dijkstra and Carel S. Scholten. 1990. *Predicate Calculus and Program Semantics*. Springer-Verlag, Berlin, Heidelberg. https://doi.org/10.1007/978-1-4612-3228-5

Matthew S. Hecht. 1977. *Flow Analysis of Computer Programs*. Elsevier Science Inc., USA.

C. A. R. Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (Oct. 1969), 576–580. https://doi.org/10.1145/363235.363259

C. A. R. Hoare. 1978. Some Properties of Predicate Transformers. *J. ACM* 25, 3 (July 1978), 461–480. https://doi.org/10.1145/322077.322088

Desharnais Jules, Bernhard Moeller, and Struth Georg. 2011. Algebraic Notions of Termination. *Logical Methods in Computer Science* Volume 7, Issue 1 (Feb. 2011). https://doi.org/10.2168/lmcs-7(1:1)2011

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2018. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM* 65, 5, Article 30 (Aug. 2018), 68 pages. https://doi.org/10.1145/3208102

Dexter Kozen. 1997. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.* 19, 3 (May 1997), 427–443. https://doi.org/10.1145/256167.256195

Petar Maksimović, Caroline Cronjäger, Andreas Lööw, Julian Sutherland, and Philippa Gardner. 2023. Exact Separation Logic: Towards Bridging the Gap Between Verification and Bug-Finding. In *37th European Conference on Object-Oriented Programming (ECOOP 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 263)*, Karim Ali and Guido Salvaneschi (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 19:1–19:27. https://doi.org/10.4230/LIPIcs.ECOOP.2023.19

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer. https://doi.org/10.1007/b138392

Marco Milanese and Francesco Ranzato. 2022. Local completeness logic on Kleene algebra with tests. In *International Static Analysis Symposium*. Springer, 350–371. https://doi.org/10.1007/978-3-031-22308-2_16

Bernhard Möller, Peter O'Hearn, and Tony Hoare. 2021. On Algebra of Program Correctness and Incorrectness. In *Relational and Algebraic Methods in Computer Science: 19th International Conference, RAMiCS 2021, Marseille, France, November 2–5, 2021, Proceedings* (Marseille, France). Springer-Verlag, Berlin, Heidelberg, 325–343. https://doi.org/10.1007/978-3-030-88701-8_20

Bernhard Möller and Georg Struth. 2005. wp Is wlp. In *International Conference on Relational Methods in Computer Science*. Springer, 200–211. https://doi.org/10.1007/11734673_16

Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996. Probabilistic Predicate Transformers. *ACM Trans. Program. Lang. Syst.* 18, 3 (1996), 325–353. https://doi.org/10.1145/229542.229547

Peter W. O'Hearn. 2019. Incorrectness Logic. *Proc. ACM Program. Lang.* 4, POPL, Article 10 (Dec. 2019), 32 pages. https://doi.org/10.1145/3371078

Azalea Raad, Julien Vanegue, and Peter O'Hearn. 2024. Non-termination Proving at Scale. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 280 (Oct. 2024), 29 pages. https://doi.org/10.1145/3689720

Xavier Rival and Kwangkeun Yi. 2020. *Introduction to Static Analysis – An Abstract Interpretation Perspective*. MIT Press.

Lena Verscht and Benjamin Kaminski. 2023. Hoare-Like Triples and Kleene Algebras with Top and Tests: Towards a Holistic Perspective on Hoare Logic, Incorrectness Logic, and Beyond. arXiv:2312.09662 [cs.LO] https://arxiv.org/abs/2312.09662

Lena Verscht and Benjamin Lucien Kaminski. 2024. A Taxonomy of Hoare-Like Logics: Towards a Holistic View using Predicate Transformers and Kleene Algebras with Top and Tests. arXiv:2411.06416 [cs.PL] https://arxiv.org/abs/2411.06416

Joakim von Wright. 2002. From Kleene algebra to refinement algebra. In *International Conference on Mathematics of Program Construction*. Springer, 233–262. https://doi.org/10.1007/3-540-45442-X_14

Joakim von Wright. 2004. Towards a refinement algebra. *Science of Computer Programming* 51, 1-2 (2004), 23–45. https://doi.org/10.1016/j.scico.2003.09.002

John Wickerson. 2024. What is the Other Incorrectness Logic? https://johnwickerson.wordpress.com/2024/02/15/what-is-the-other-incorrectness-logic/.

Cheng Zhang, Arthur Azevedo de Amorim, and Marco Gaboardi. 2022. On incorrectness logic and Kleene algebra with top and tests. *Proc. ACM Program. Lang.* 6, POPL, Article 29 (Jan. 2022), 30 pages. https://doi.org/10.1145/3498690

Linpeng Zhang and Benjamin Lucien Kaminski. 2022. Quantitative Strongest Post. *CoRR* abs/2202.06765 (2022). https://doi.org/10.48550/ARXIV.2202.06765 arXiv:2202.06765

Noam Zilberstein, Derek Dreyer, and Alexandra Silva. 2023. Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 93 (April 2023), 29 pages. https://doi.org/10.1145/3586045