Translation Incorrectness and KAT

Author: Antal Huisman

Supervisors: Dan Frumin, Jorge A. Pérez

Computing Science
RuG
Juli 2021

Contents

1 Introduction

Program verification is checking if programs comply on certain specifications. One possible type of specifications states that the program is error- and bug-free. In most programs we permit that the occasional error slips though, but for some programs this rule must be strict. For example in a hospital, we wouldn't want a machine to crash while a patient is hooked up to it. Encryption should not become predictable because of a bug. This is why Program Correctness is important.

In our studies we had the course Program Correctness. In it we learned methods to prove programs correct. One of the methods was a rigorous mathematics called Hoare Logic. In Hoare Logic you prove program correct by applying particular rules. But the vast majority of the time we are not working with programs that work correctly. We are working with buggy programs that we want to make correct. It is fruitless to try to prove that such a program is correct.

What if instead you could prove a bug existed in the code? You could test inputs one by one. To prove the program is error-free, you'd have to check all, potential infinite, inputs. Recently a paper was published about Incorrectness Logic [1]. Incorrectness Logic is an axiomatic approach, like Hoare Logic. Unlike Hoare Logic, it instead can prove that a program contains an error.

There is a formal Algebraic structure behind Hoare Logic. This structure is called Kleene Algebra with Tests (KAT). It is an extension of Kleene Algebra, which is used for regular expressions. We learned about Kleene Algebra in the course Languages and Machines.

This structure allows us to use an equational, algebraic reasoning to verify programs. Instead of applying the propositional rules in Hoare Logic to prove a program correct, we now simply evaluate an equation in KAT. With this structure we can more easily make general statements about programs. You could prove that two programs are equivalent, which is harder to do with Hoare Logic.

The structure of Incorrectness Logic seems similar to that of Hoare Logic. If there were a Algebraic structure of Incorrectness Logic, then finding an error with Incorrectness Logic could similarly be reduced to solving an equation. That would mean that we can detect errors by solving an equation, instead of testing inputs one-by-one. This could speed up error detection algorithms.

Which brings me to my research question:
Can we represent Incorrectness Logic with Kleene Algebra with Tests?

To answer this question we have to answer three questions:

- What is the translation from Incorrectness Logic to KAT?
- Is this translation sound?
- Is this translation complete?

In the rest of the thesis we will give a partial answer to these questions. In Section 2 will recall the preliminaries on Hoare Logic, and introduce Incorrectness Logic and KAT. Section 3 will detail the translation of Hoare Logic to KAT and its soundness and completeness. We will answer the first two questions in Section 4 and then discuss the difficulties in answering the third question in Section 5.

2 Preliminaries

2.1 Hoare Logic

Propositional Hoare Logic (PHL) is used to prove if a program is correct.

Hoare Logic uses triples of the form $\{b\}p\{c\}$. In the triple $\{b\}p\{c\}$, b is the pre-condition, c is the post-condition and p is the code. The triple is true if the post-condition is an over-approximation of the result of applying the pre-condition to the code. The result of applying the code p to the pre-condition b is included in the post-condition c.

Hoare Logic over-approximates the result based on the presumption. In other words, in every triplet the result is only what is true in every possible path the code can take the presumption. If this result is the expected result of the program, we have proven that the program is correct.

We demonstrate Hoare Logic on an example to contrast it later with Incorrectness Logic. Consider the following program p:

```
// Pre-condition: z = 11
if (x == 39)
    then if (y == 5)
        then z := 42;
    ifend
ifend
// Post-condition: z = 11 or z = 42
```

The code with its specifications would result in the triple: $\{z = 11\}p\{z = 11 \lor z = 42\}$.

This triple $\{z = 11\}p\{z = 11 \lor z = 42\}$ is correct. It's easy to see that there are only two options: either the assignment `z := 42` is applied or not. Thus the post-condition is either $z = 11$ or $z = 42$.

Hoare Logic Rules

As a refresher, the rules are given in Figure 1.

$$\textsc{Skip Axiom}\ \dfrac{}{\{b\}\text{skip}\{b\}} \qquad \textsc{Assignment Axiom}\ \dfrac{}{\{b[E/x]\}x := E\{b\}} \qquad \textsc{Composition}\ \dfrac{\{b\}p\{c\} \qquad \{c\}q\{d\}}{\{b\}p;q\{d\}}$$

$$\textsc{Weakening}\ \dfrac{b' \to b \qquad \{b\}p\{c\} \qquad c \to c'}{\{b'\}p\{c'\}} \qquad \textsc{Conditional}\ \dfrac{\{b \land c\}p\{d\} \qquad \{\neg b \land c\}q\{d\}}{\{c\}\text{if } b \text{ then } p \text{ else } q \text{ ifend}\{d\}} \qquad \textsc{While}\ \dfrac{\{b \land c\}p\{c\}}{\{c\}\text{while } b \text{ then } p \text{ done}\{\neg b \land c\}}$$

Figure 1: Hoare Logic rules.

With these rules you can prove simple programs correct. There are extensions of Hoare Logic to reason with, for example, functional programming or parallel programming.

In the example, we could also arrive to its answer by applying the rules to get the narrowest over-approximation:

$$\{z = 11\}p\{(x \neq 39 \land z = 11) \lor (x = 39 \land y \neq 5 \land z = 11) \lor (x = 39 \land y = 5 \land z = 42)\}$$

Then apply the Weakening Rule to arrive at the given triple $\{z = 11\}p\{z = 11 \lor z = 42\}$. Note that with the Rule of Weakening we are able to remove conjunctions from the post-condition.

2.2 Incorrectness Logic

While Hoare Logic proves a program is correct, Incorrectness Logic [1] is used to prove that errors exist. Incorrectness Logic uses the triples $[b]p[ok : c]$ and $[b]p[er : d]$, where b is the presumption, p is the code, c are the states that are reachable and d the states of b that resulted in an error. These two triples are usually combined in the quadruple $[b]p[ok : c][er : d]$ or the triple $[b]p[\epsilon : c]$. Incorrectness Logic is written similar as Hoare Logic, but to differentiate the two, square brackets are used instead of curly brackets.

Hoare Logic over-approximates the result based on the presumption. In Incorrectness Logic we instead under-approximate the result based on the presumption. The post-condition must be reachable. In other words, for every output in the post-condition c there is a input in the pre-condition b that would lead to the output if the code p is applied to the input.

If one possible path leads to an error, then that error is possible in the program. Thus the point is to discard all paths that don't lead to an error and keep specific paths that would lead to an error.

Incorrectness Logic Rules

The Incorrectness Rules are given in Figure 2. The functions are defined in Figure 3.

$$\textsc{Empty Under-approximation}\quad \frac{}{[p]C[false]}$$

$$\textsc{Consequence}\quad \frac{p \to p' \qquad [p]C[q] \qquad q' \to q}{[p']C[q']}$$

$$\textsc{Disjunction}\quad \frac{[p_1]C[q_1] \qquad [p_2]C[q_2]}{[p_1 \lor p_2]C[q_1 \lor q_2]}$$

$$\textsc{Unit}\quad \frac{}{[p]\text{skip}[ok : p][er : false]}$$

$$\textsc{Sequencing (short circuit)}\quad \frac{[p]C_1[er : r]}{[p]C_1; C_2[er : r]}$$

$$\textsc{Sequencing (normal)}\quad \frac{[p]C_1[ok : q] \qquad [q]C_2[r]}{[p]C_1; C_2[r]}$$

$$\textsc{Iterate Zero}\quad \frac{}{[p]C^*[ok : p]}$$

$$\textsc{Iterate Non-zero}\quad \frac{[p]C^*; C[q]}{[p]C^*[q]}$$

$$\textsc{Backwards Variant } (n \text{ is fresh})\quad \frac{[p(n) \land nat(n)]C[ok : p(n+1) \land nat(n)]}{[p(0)]C^*[ok : \exists n, p(n) \land nat(n)]}$$

$$\textsc{Choice } (\text{where } i = 1 \text{ or } i = 2)\quad \frac{[p]C_i[q]}{[p]C_1 \land C_2[q]}$$

$$\textsc{Error}\quad \frac{}{[p]Error()[ok : false][er : p]}$$

$$\textsc{Assume}\quad \frac{}{[p]Assume(B)[ok : p \land B][er : false]}$$

Figure 2: Incorrectness Logic rules.

$$\textsc{While}\quad \frac{\text{while B do C done}}{(Assume(B); C)^*; Assume(\neg B)}$$

$$\textsc{Conditional}\quad \frac{\text{if B then C else C' ifend}}{(Assume(B); C) + (Assume(\neg B); C')}$$

$$\textsc{Assert}\quad \frac{Assert(B)}{(Assume(B)) + (Assume(\neg B); Error())}$$

Figure 3: Incorrectness Logic functions.

Notice that Incorrectness Logic doesn't have explicit rules for if and while, but they are constructed by the other rules.

Example 1

To compare Incorrectness Logic with Hoare Logic, let's look at the same program of the previous section. In the previous example we had the post-condition $z = 11 \lor z = 42$ with Hoare Logic. Is this post-condition also correct with Incorrectness Logic?

```
// Pre-condition: z = 11
if (x == 39)
    then if (y == 5)
        then z := 42;
    ifend
ifend
// Post-condition: z = 11 or z = 42?
```

The answer is no. In Incorrectness Logic the post-condition must be reachable. The post-condition $z = 11 \lor z = 42$ includes the state $x = 39 \land y = 5 \land z = 11$. There is no pre-condition that would lead to this post-condition, because if $x = 39 \land y = 5$, then z becomes 42.

The reason for this restriction is that we do not want to create a false-positive error, where we detect an error when there is none. It is possible that an error occurs when $x = 39 \land y = 5 \land z = 11$ later in the program. For example with the code `x := x / (z - 2y- 1)`. This code p would prevent that error. With the post-condition $z = 11 \lor z = 42$, this possibility is included and with it we included a false-positive error.

The correct triple with all information is:

$$[z = 11]p[(x \neq 39 \land z = 11) \lor (x = 39 \land y \neq 5 \land z = 11) \lor (x = 39 \land y = 5 \land z = 42)].$$

You may have noticed that the correct triple is the same as the narrowest over-approximation from the Hoare Logic example. This is because the triple is the exact answer, which is both the narrowest over-approximation and the biggest under-approximation.

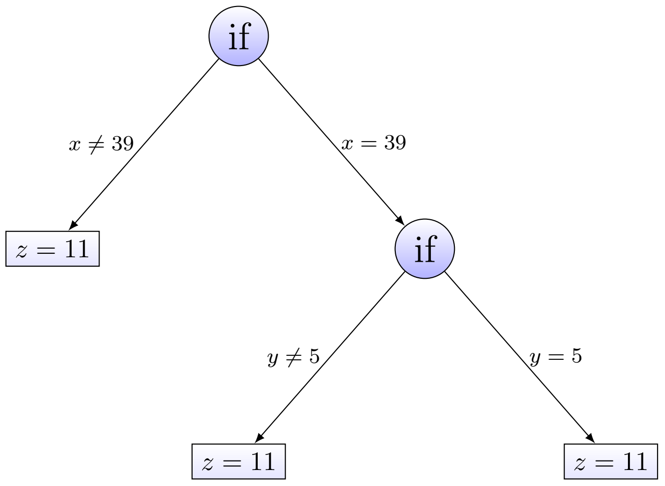
Example 2

Let's say we are only interested in the post-condition $z = 42$. We either suspect an error occurs later in the code when the post-condition is $z = 42$ or we know for a fact that no errors occur later when $z = 11$. We can then just remove the $z = 11$ part to get the triple $[z = 11]p[ok : x = 39 \land y = 5 \land z = 42]$.

```
// Pre-condition: z = 11
if (x == 39)
    then if (y == 5)
        then z := 42;
    ifend
ifend
// Post-condition: x = 39 and y = 5 and z = 42
```

What we did is apply the Rule of Consequence. With the Rule of Consequence we are able to remove disjunctions from the post-condition. In contrast, the Weakening Rule of Hoare Logic can remove conjunctions.

Why can we apply this Rule? Look at all the paths the code can take.



$$(x \neq 39 \land z = 11) \lor (x = 39 \land y \neq 5 \land z = 11) \lor (x = 39 \land y = 5 \land z = 42).$$

A path splits at every if and while statement. At the bottom are the results of each path. Disjunctions separate different paths. Note that this is the post-condition of the previous example.

Each path either contains an error or results in a successful termination. The results of the other paths has no influence on whether a path succeeds or fails. We are only interested in errors, so we can remove paths that lead to success. To remove paths, we should be able to remove disjunctions in the post-condition. Because the Rule of Consequence can remove disjunctions, it can also remove paths with errors.

Example 3

What if we had the pre-condition *true*? Would then the triple $[true]q[ok : z = 42]$ be correct?

```
// Pre-condition: true
if (x == 39)
    then if (y == 5)
        then z := 42;
    ifend
ifend
// Post-condition: z = 42?
```

The answer is yes, which may be surprising.

The short explanation is this: the triple $[z = 42]q[ok : z = 42]$ is obviously correct. We then apply the Rule of Consequence on the pre-condition to get the given triple $[true]q[ok : z = 42]$.

Here is an explanation why we can do this. The biggest under-approximation triple is

$$[true]q[x \neq 39 \lor (x = 39 \land y \neq 5) \lor (x = 39 \land y = 5 \land z = 42)]$$

We can rewrite the post-condition of this

$$(x \neq 39 \land (z = 42 \lor z \neq 42)) \lor (x = 39 \land y \neq 5 \land (z = 42 \lor z \neq 42)) \lor (x = 39 \land y = 5 \land z = 42)$$

into the equivalent post-condition

$$z = 42 \lor (x \neq 39 \land z \neq 42) \lor (x = 39 \land y \neq 5 \land z \neq 42)$$

We have three paths with the results: $z = 42$, $(x \neq 39 \land z \neq 42)$ and $(x = 39 \land y \neq 5 \land z \neq 42)$. With the Rule of Consequence we pick the post-condition $z = 42$ to get the given triple.

In the previous example the paths had the same structure as the code, where the paths will split at every if and while. This example shows that a path is more flexible than that. It doesn't have to follow the exact tree structure.

Error

Incorrectness Logic also includes triples with the error part. The special $Error()$ function creates an error and it has the definition $[b]Error()[er : b]$. The special function was added, because it was outside the scope to automatically detect bugs, like for dividing by zero or integer overflow.

Let's look at this example r with $Error()$.

```
// pre-condition: b = true
if (x == 1)
    then Error()
    else x := 1
ifend
```

All paths that do not lead to an error, have the result $x = 1$. Thus the ok post-condition is $x = 1$.
For the error post-condition, remember that it is the part of the input that leads to an error. The part of the input that leads to an error is $x = 1$.

This will result in the quadruple: $[true]r[ok : x = 1][er : x = 1]$. This example shows that the two post-condition in the Incorrectness quadruple do not have to be disjoint. For simplicity, for now we do not consider the error post-conditions.

2.3 Kleene Algebra with Tests

Kleene Algebra with Tests [2] is a Kleene Algebra with an added Boolean Algebra. Is usually shorted to KAT. Formally, it is the structure :

$$(\mathcal{K}, \mathcal{B}, +, \cdot, *, ^-, 0, 1)$$

where:

- $\mathcal{B} \subseteq \mathcal{K}$.
- $(\mathcal{K}, +, \cdot, *, 0, 1)$ is a Kleene Algebra.
- $(\mathcal{B}, +, \cdot, ^-, 0, 1)$ is a Boolean Algebra.

and $^-$ is only defined on $\mathcal{B}$.
The elements of $\mathcal{B}$ are called tests. We use the letters b, c, d, e for elements of $\mathcal{B}$ and the letters p, q, r for elements of $\mathcal{K}$. For Kleene Algebra the operations $+, \cdot, *$ represent non-deterministic choice, concatenation, the Kleene star respectively. For Boolean Algebra the operations $+, \cdot, ^-$ represent disjunction, conjunction, negation respectively. The elements of $\mathcal{K}$ represent the code of the program, while the elements of $\mathcal{B}$ represent states that are true in that place in the code.

Both Kleene Algebra and Boolean Algebra have the following axioms:

$x + (y + z) = (x + y) + z$	Associative
$x + y = y + x$	Commutative
$x + 0 = x$	Identity unit 0
$x + x = x$	Idempotent
$x(yz) = (xy)z$	Associative
$1x = x1 = x$	Identity unit 1
$x(y + z) = xy + xz$	Distribution of $\cdot$ over $+$
$(x + y)z = xz + yz$	Distribution of $\cdot$ over $+$
$0x = x0 = 0$	Annihilation

In addition Kleene Algebra has the following axioms for the Kleene star:

$$p^* = 1 + p^*p \tag{1}$$

$$p^* = 1 + pp^* \tag{2}$$

$$q + pr \sqsubseteq r \rightarrow p^*q \sqsubseteq r \tag{3}$$

$$q + rp \sqsubseteq r \rightarrow qp^* \sqsubseteq r \tag{4}$$

where $\sqsubseteq$ is the partial order defined as:

$$a \sqsubseteq b \overset{\text{def}}{\longleftrightarrow} a + b = b \tag{5}$$

The partial order is reflexive, anti-symmetric and transitive.

Boolean Algebra has the following added axioms:

$a + 1 = 1$	Identity Property
$ab = ba$	Commutative
$aa = a$	Idempotent
$a + ab = a$	Absorption
$a(a + b) = a$	Absorption
$a\overline{a} = 0$	Complement
$a + \overline{a} = 1$	Complement
$\overline{\overline{a}} = a$	Involution
$\overline{a} + \overline{b} = \overline{ab}$	De Morgan Law
$\overline{a}\overline{b} = \overline{a + b}$	De Morgan Law

With KAT we can encode the functions if, then, else and while, do. [3]

The encodings are:

$$\begin{gathered}
\text{skip} \rightarrow 1 \\
p;\, q \rightarrow pq \\
\text{if } b \text{ then } p \text{ ifend} \rightarrow (bp + \overline{b}) \\
\text{if } b \text{ then } p \text{ else } q \text{ ifend} \rightarrow (bp + \overline{b}q) \\
\text{while } b \text{ do } p \text{ done} \rightarrow (bp)^*\overline{b}
\end{gathered}$$

As an example, let's encode the earlier code:

```
if (x == 39)
    then if (y == 5)
        then z := 42;
    ifend
ifend
```

Where b is `x == 39`, c is `y == 5` and p is `z := 42`.

We will get: $(b(cp + \overline{c}) + \overline{b})$. Or written out: $bcp + b\overline{c} + \overline{b}$

2.4 Relational Algebra

Kleene Algebra with Tests [2] can create a model of Relational Algebra. Note that this is only a subset of Relational Algebra [4].

Relational Algebra is the Algebra of binary relations. Binary relations, like functions, have an input and an output. But, unlike functions, not all inputs need to have an output and an input can have multiple outputs. This is why Relational Algebra is also used as a model for programs. The binary relations are the code of the program and the input and output are the states between code.

In the model with KAT, the code p, q, r are still the binary relation. But the tests $b, c, d\dots$ would also have to be relations. Instead they are a relation like the identity function ι with restricted input and output.

A more formal definition: Let X be a set of states. Let $\mathcal{K}$ and $\mathcal{B}$ be binary relations in $X \times X$ and $\circ$ be the relational composition. All binary relations in set $\mathcal{B}$ are subsets of the identity relation ι. $*$ is the reflexive transitive closure of a relation.

Then the relational Kleene Algebra is a structure in the form:

$$(\mathcal{K}, \mathcal{B}, \cup, \circ, *, ^{-}, \varnothing, \iota)$$

We need to add an Kleene Algebra variable $\top$ to reason with Incorrectness Logic. It is the everywhere-true binary relation from Relational Algebra. Intuitively, $\top$ acts as a clean slate where everything is true again. There is no code equivalent for $\top$.

Unfortunately it is an open problem if the $\top$ from Relational Algebra can be added to Kleene Algebra. There is no complete axiomatisation of $\top$ in Kleene Algebra. For example, the equation $\top p \top p \top = \top p \top$ holds in Relational Algebra. No proof has been found in Kleene Algebra, so we don't know if it holds there.

We use the following equations of $\top$ for our proof. We will use that $\top$ is the "largest" element in $\mathcal{K}$:

$$\forall x \in \mathcal{K} : x \sqsubseteq \top \tag{6}$$

An implication of $\top$ that we will use, is:

$$\top a = 0 \rightarrow a = 0 \tag{7}$$

3 Translation from PHL to KAT

3.1 Definition of the Translation

Hoare Logic can be translated into KAT with the two equivalent equations [1]:

$$\{b\}p\{c\} \text{ holds iff } bp\overline{c} = 0$$

Which means intuitively that code p with the pre-condition b and post-condition $\overline{c}$ will never terminate.

$$\{b\}p\{c\} \text{ holds iff } bp = bpc$$

Which means intuitively that checking the post-condition c for the code p with the pre-condition b is redundant.

With the code now a variable, there is no need for the Assignment Axiom in KAT. We instead tread the assignment as an hypothesis. Thus in general the implication in Hoare Logic where every $\{b_i\}p_i\{c_i\}$ is an assignment

$$\frac{\{b_1\}p_1\{c_1\}, \{b_2\}p_2\{c_2\}, \ldots, \{b_n\}p_n\{c_n\}}{\{b\}p\{c\}}$$

will be translated into KAT as the Horn formula:

$$b_1p_1\overline{c_1} = 0 \wedge b_2p_2\overline{c_2} = 0 \wedge \cdots \wedge b_np_n\overline{c_n} = 0 \rightarrow bp\overline{c} = 0$$

3.2 Soundness and Completeness

To prove that a translation holds, we need to prove that it is sound and complete. Soundness means that every statement that can be proven in Hoare Logic can also be proven in KAT. For that we only need to check that every rule in Hoare Logic will also hold after the translation. Completeness means that if an translation of an assertion holds in KAT, it will also hold in Hoare Logic.

The translation is sound. The Composition Rule, the Consequence Rule, the Conditional Rule and the While Rule all hold in KAT [5].

The completeness proof is harder to construct. We would need to prove that for every implication in KAT of the form:

$$b_1p_1\overline{c_1} = 0 \wedge b_2p_2\overline{c_2} = 0 \wedge \cdots \wedge b_np_n\overline{c_n} = 0 \rightarrow bp\overline{c} = 0$$

the implication in Hoare Logic also holds:

$$\frac{\{b_1\}p_1\{c_1\}, \{b_2\}p_2\{c_2\}, \ldots, \{b_n\}p_n\{c_n\}}{\{b\}p\{c\}}$$

This is impossible to do. The fact that the implication is true, does not give the steps for a proof of the implication in Hoare Logic.

Instead they prove that KAT is complete for all universal Horn formulas that are relational valid of the form:

$$r_1 = 0 \wedge r_2 = 0 \wedge \cdots \wedge r_n = 0 \rightarrow p = q$$

They do this by eliminating the hypothesis. They prove that the formula $r = 0 \rightarrow p = q$ is equivalent with an equation $p' = q'$. Since KAT is complete for all valid equations, the translation is also complete.

4 Translation from Incorrectness to KAT

We will be focusing only one triples of the form $[b]p[ok : c]$. We ignore the error part for now.
The reason for that is that this triple is simpler. The intention was that after the translation of this form was proven, the translation would be extended with the error part. This didn't happen.

4.1 Finding the Translation

Hoare Logic and Incorrectness Logic can be encoded [1] in Relational Algebra as the formulas:

$$\{b\}p\{c\} \text{ holds iff } \top; \mathit{assume}(b); p \sqsubseteq \top; \mathit{assume}(c)$$

$$[b]p[c] \text{ holds iff } \top; \mathit{assume}(b); p \sqsupseteq \top; \mathit{assume}(c)$$

Where $\mathit{assume}(p)$ is defined as the binary relation:

$$\text{If } b \subseteq \Sigma \text{ then let } \mathit{assume}(b) = \{(\sigma, \sigma) \mid \sigma \in b\}$$

where Σ is the set of states σ. $\top$ is the everywhere-true binary relation and $\sqsubseteq$ and $\sqsupseteq$ are the order operations.

Relations that will become Booleans, have the form: $\{(\sigma, \sigma) \mid \sigma \in a\}$. Thus $\top; \mathit{assume}(b); p$ should be read in KAT as: $\top bp$.

We will first check if the Hoare Logic formula is correct. Translating the Relational Algebra into Kleene Algebra, we get:

$$\begin{aligned}
\top bp &\sqsubseteq \top c && \text{multiply both sides with } \bar{c} \\
\top bp\bar{c} &\sqsubseteq \top c\bar{c} && \text{using axiom } a\bar{a} = 0 \text{ and } p0 = 0 \\
\top bp\bar{c} &\sqsubseteq 0 && \text{by the axiom: } a \sqsubseteq b \text{ is equivalent with } a + b = b \\
\top bp\bar{c} + 0 &= 0 && \text{using axiom } a + 0 = a \\
\top bp\bar{c} &= 0 && \text{by (7)} \\
bp\bar{c} &= 0
\end{aligned}$$

Now we have the given formula.

We can get the KAT formula for the Incorrectness Logic using the same method. Unfortunately the $\top$ gets in the way and therefore we can not multiply on the left side. Translating the Relational Algebra into Kleene Algebra, we get:

$$\begin{aligned}
\top bp &\sqsupseteq \top c && \text{using the axiom: } a \sqsubseteq b \text{ is equivalent with } a + b = b \\
\top bp + \top c &= \top bp
\end{aligned}$$

Thus we get the formula: $\top bp + \top c = \top bp$.

This equation means intuitively that the results from the post-condition c are included in the results of applying the pre-condition b on the code p.

4.2 Soundness of the Translation

The assertion $[b]p[ok : c]$ of Incorrectness Logic can be encoded to KAT with the following translation:

$$\top bp + \top c = \top bp \tag{8}$$

The rules of Incorrectness Logic translated with (8) look like this:
Consequence Rule:

$$b \sqsubseteq b' \wedge \top bp + \top c = \top bp \wedge c' \sqsubseteq c \rightarrow \top b'p + \top c' = \top b'p \tag{9}$$

Disjunction Rule:

$$\top bp + \top c = \top bp \wedge \top dp + \top e = \top dp \rightarrow \top(b+d)p + \top(c+e) = \top(b+d)p \tag{10}$$

Sequencing Rule:

$$\top bp + \top c = \top bp \wedge \top cq + \top d = \top cq \rightarrow \top bpq + \top d = \top bpq \tag{11}$$

Iterate Non-zero Rule:

$$\top bp^*p + \top c = \top bp^*p \rightarrow \top bp^* + \top c = \top bp^* \tag{12}$$

Choice Rule:

$$\top bp + \top c = \top bp \vee \top bq + \top c = \top bq \rightarrow \top b(p+q) + \top c = \top b(p+q) \tag{13}$$

While Rule:

$$\top bcp + \top c = \top bcp \rightarrow \top c(bp)^*\bar{b} + \top \bar{b}c = \top c(bp)^*\bar{b} \tag{14}$$

Conditional Rule:

$$\top bcp + \top d = \top bcp \wedge \top \bar{b}cq + \top d = \top \bar{b}cq \rightarrow \top c(bp + \bar{b}q) + \top d = \top c(bp + \bar{b}q) \tag{15}$$

Now I will prove that the rules of Incorrectness Logic will hold, using these translations.

Theorem 1. *The formulas (9) - (15) are theorems in KAT.*

Proof. The first rule to prove is the Consequence Rule (9).
Assuming the premises:

$$b \sqsubseteq b' \tag{16}$$

$$\top bp + \top c = \top bp \tag{17}$$

$$c' \sqsubseteq c \tag{18}$$

Because of the axiom that $a \sqsubseteq b$ is equivalent with $a + b = b$, we get for (16) and (18):

$$b + b' = b' \tag{19}$$

$$c + c' = c \tag{20}$$

Now the proof:

$$\begin{aligned}
\top b'p &= \top(b+b')p && \text{by (19)}\\
&= \top bp + \top b'p && \text{by distributivity}\\
&= \top bp + \top c + \top b'p && \text{by (17)}\\
&= \top bp + \top(c+c') + \top b'p && \text{by (20)}\\
&= \top bp + \top c + \top c' + \top b'p && \text{by distributivity}\\
&= \top bp + \top c' + \top b'p && \text{by (17)}\\
&= \top(b+b')p + \top c' && \text{by distributivity}\\
&= \top b'p + \top c' && \text{by (19)}
\end{aligned}$$

Thus the implication (9) holds.

Next is the Disjunction Rule (10). Assuming the premises:

$$\top bp + \top c = \top bp \tag{21}$$

$$\top dp + \top e = \top dp \tag{22}$$

Using those rules, we can now prove:

$$\begin{aligned}
\top(b+d)p &= \top bp + \top dp && \text{by distributivity}\\
&= \top bp + \top c + \top dp + \top e && \text{by (21) and (22)}\\
&= \top(b+d)p + \top(c+e) && \text{by distributivity}
\end{aligned}$$

Thus the implication (10) is true.

After that is the Sequencing Rule (11). Assuming the premises:

$$\top bp + \top c = \top bp \tag{23}$$

$$\top cq + \top d = \top cq \tag{24}$$

Then the proof:

$$\begin{aligned}
\top bpq &= (\top bp + \top c)q && \text{by (23)}\\
&= \top bpq + \top cq && \text{by distributivity}\\
&= \top bpq + \top cq + \top d && \text{by (24)}\\
&= (\top bp + \top c)q + \top d && \text{by distributivity}\\
&= \top bpq + \top d && \text{by (23)}
\end{aligned}$$

Thus the implication (11) is proven.

The next proof is the Iteration Non-zero Rule (12). With the premises:

$$\top bp^*p + \top c = \top bp^*p \tag{25}$$

Then:

$$\begin{aligned}
\top bp^* &= \top b(1 + p^*p) && \text{by the Kleene Identity (1)}\\
&= \top b + \top bp^*p && \text{by distributivity}\\
&= \top b + \top bp^*p + \top c && \text{using (25)}\\
&= \top b(1 + p^*p) + \top c && \text{by distributivity}\\
&= \top bp^* + \top c && \text{by the Kleene Identity (1)}
\end{aligned}$$

Thus (12) holds generally.

For the Choice Rule (13), the premises are either:

$$\top bp + \top c = \top bp \tag{26}$$

Or:

$$\top bq + \top c = \top bq \tag{27}$$

Then

$$\begin{aligned}
\top b(p+q) &= \top bp + \top bq && \text{by distributivity}\\
&= \top bp + \top bq + \top c && \text{by (26) or (27)}\\
&= \top b(p+q) + \top c && \text{by distributivity}
\end{aligned}$$

Thus the implication (13) holds.

Next is the While Rule (14). Assume:

$$\top bcp + \top c = \top bcp \tag{28}$$

Then:

$$\begin{aligned}
\top c(bp)^*\overline{b} &= \top c(1 + (bp)^*bp)\overline{b} && \text{by the Kleene Identity (1)}\\
&= \top c(bp)^*bp\overline{b} + \top c\overline{b} && \text{by distributivity}\\
&= \top c(bp)^*bp\overline{b} + \top c\overline{b} + \top c\overline{b} && \text{by Kleene axiom } a + a = a\\
&= \top c(bp)^*bp\overline{b} + \top c\overline{b} + \top\overline{b}c && \text{by Boolean axiom } ab = ba\\
&= \top c(1 + (bp)^*bp)\overline{b} + \top\overline{b}c && \text{by distributivity}\\
&= \top c(bp)^*\overline{b} + \top\overline{b}c && \text{by the Kleene Identity (1)}
\end{aligned}$$

Note that we didn't use (28).

Then the last rule is the Conditional Rule (15). Assume the premises:

$$\top bcp + \top d = \top bcp \tag{29}$$

$$\top\overline{b}cq + \top d = \top\overline{b}cq \tag{30}$$

Then the proof:

$$\begin{aligned}
\top c(bp + \overline{b}q) &= \top cbp + \top c\overline{b}q && \text{by distributivity}\\
&= \top bcp + \top\overline{b}cq && \text{by Boolean axiom } ab = ba\\
&= \top bcp + \top\overline{b}cq + \top d && \text{by (29) or (30)}\\
&= \top cbp + \top c\overline{b}q + \top d && \text{by Boolean axiom } ab = ba\\
&= \top c(bp + \overline{b}q) + \top d && \text{by distributivity}
\end{aligned}$$

Thus the rules hold.

□

4.3 Alternative translation attempts

We got the formula: $\top bp + \top c = \top bp$. Although this equation is correct, it is hard to prove completeness with it. Here are some ideas that I explored.

Attempt 1

First, we can not use the same trick that Hoare Logic uses and multiply both sides with $\overline{c}$. We will get:

$$\begin{aligned}
\top bp &\sqsupseteq \top c && \text{multiply both sides with } \overline{c}\\
\top bp\overline{c} &\sqsupseteq \top c\overline{c} && \text{by axiom } a\overline{a} = 0\\
\top bp\overline{c} &\sqsupseteq \top 0 && \text{by axiom } a0 = 0\\
\top bp\overline{c} &\sqsupseteq 0 && \text{using the axiom: } a \sqsubseteq b \text{ is equivalent with } a + b = b\\
\top bp\overline{c} + 0 &= \top bp\overline{c}
\end{aligned}$$

$\top bp\overline{c} \sqsupseteq 0$ might seem like a nice equation, but it is actually always true and therefore useless to us.

Attempt 2

You could multiply both sides with c to get:

$$\begin{aligned} \top bp &\sqsupseteq \top c && \text{multiply both sides with } c \\ \top bpc &\sqsupseteq \top cc && \text{by Boolean axiom } aa = a \\ \top bpc &\sqsupseteq \top c \end{aligned}$$

The formula $\top bpc \sqsupseteq \top c$ is not that interesting. It is however still sound: all the proofs will use the same steps as for $\top bp \sqsupseteq \top c$. This is left as an exercise for the reader.

Attempt 3

You could add $\top\overline{c}$ to both sides to get:

$$\begin{aligned} \top c &\sqsubseteq \top bp && \text{add } \top\overline{c} \text{ to both sides} \\ \top c + \top\overline{c} &\sqsubseteq \top bp + \top\overline{c} && \text{by distributivity} \\ \top(c + \overline{c}) &\sqsubseteq \top bp + \top\overline{c} && \text{by Boolean axiom } a + \overline{a} = 1 \\ \top 1 &\sqsubseteq \top bp + \top\overline{c} && \text{by axiom } a1 = a \\ \top &\sqsubseteq \top bp + \top\overline{c} \end{aligned}$$

Combined with the fact that $\forall x : x \sqsubseteq \top$ and the anti-symmetry of $\sqsubseteq$, we get: $\top = \top bp + \top\overline{c}$.

This implies that the result of $\top bp$ and the result of $\top\overline{c}$ together cover all possible states. $\overline{c}$ could be seen as all unreachable states of bp.

Attempt 4

We can play with the fact that $\forall x : x \sqsubseteq \top$.
If we take $\top p \sqsubseteq \top$ and $\top b \sqsubseteq \top$ and we combine that with with the found equation, we get:

$$\begin{aligned} &\top bpc \sqsubseteq \top pc \sqsubseteq \top c \sqsubseteq \top bp && \text{by transitivity of } \sqsubseteq \\ &\top bpc \sqsubseteq \top bp \end{aligned}$$

The formula $\top bpc \sqsubseteq \top bp$ is equivalent with the formula $0 \sqsubseteq \top bp\overline{c}$ in Attempt 1, and thus also useless.

Attempt 5

If we multiply the formula $\top bpc \sqsubseteq \top pc \sqsubseteq \top c \sqsubseteq \top bp$ of Attempt 4 with c, we get:

$$\begin{aligned} &\top bpc \sqsubseteq \top pc \sqsubseteq \top c \sqsubseteq \top bp && \text{multiply by } c \\ &\top bpcc \sqsubseteq \top pcc \sqsubseteq \top cc \sqsubseteq \top bpc && \text{by Boolean axiom } cc = c \\ &\top bpc \sqsubseteq \top pc \sqsubseteq \top c \sqsubseteq \top bpc && \text{by anti-symmetry of } \sqsubseteq \\ &\top bpc = \top pc = \top c \end{aligned}$$

If we could remove the $\top$, we would get $bpc = pc$.

The formula $bpc = pc$ is equivalent with the formula $\overline{b}pc = 0$ and is one of the formulas I found on my own. I have found the soundness proofs with these two formulas. The translation would also be complete, because $\overline{b}pc = 0$ is of the form $r = 0$. Unfortunately, this is just Hoare Logic where both the pre-condition and the post-condition are negated.

5 Challenges with proving Completeness

5.1 Eliminating Hypothesis

Proving that the translation is complete is hard the normal way. The proof of completeness of Hoare Logic could be reused for Incorrectness Logic if the equation $\top bp + \top c = \top bp$ could be rewritten.
It is possible to eliminate the hypothesis if it is in any of these forms [6]:

- in the form $e = 0$ where e is an expression
- in the form $a \leq 1$ where a is a letter
- in the form $1 = w$ or $a = w$ where a is a letter and w is a word with extra properties
- in the form $S = 1$ where S is a sum of letters

I focused on the first option, because we are not working with letters and we don't have extra properties. But I couldn't find any equivalent equation in the form $e = 0$. The $\top$ prevents this.

5.2 Eliminating the $\top$

Another problem is that I added the new symbol $\top$ to KAT. Earlier I said that is an open problem whether we can do this. It would be better if we could replace $\top$.

Replace with $\forall$

We could replace $\top bp + \top c = \top bp$ with $\forall x : xbp + xc = xbp$, but then we get into problems.
Take for example $x = \overline{b}$ and then:

$$\begin{aligned} xbp + xc &= xbp && \text{take } x = \overline{b} \\ \overline{b}bp + \overline{b}c &= \overline{b}bp && \text{by axiom } a\overline{a} = 0 \\ 0p + \overline{b}c &= 0p && \text{by axiom } 0a = 0 \\ \overline{b}c &= 0 \end{aligned}$$

We have removed the $\top$, but added the restriction $\overline{b}c = 0$. This restriction is too severe: it means that the pre-condition b needs to include the post-condition c. Thus the simple triple $[x = 0]x := x + 1[ok : x = 1]$ would not be accepted in this model. I am not sure, but I suspect this is the same as the formulas $bpc = pc$ and $\overline{b}pc = 0$ from earlier.

Replace with $\exists$

It would be correct to replace $\top bp + \top c = \top bp$ with $\exists x : xbp + xc = xbp$. Unfortunately this has the trivial solution $x = 0$ or the less trivial solution where $xc = 0$. In addition, the proofs assume that all $\top$ are the same, but we will lose that guarantee. Look at the adjusted Sequencing Rule:

$$\exists x : xbp + xc = xbp \wedge \exists y : ycq + yd = ycq \rightarrow \exists z : zbpq + zd = zbpq \tag{31}$$

The given proof only works if $z = xy = yx$, and constructing a different proof for $xy \neq yx$ seems impossible.
We could restrict in such a way that all x need to be the same. The adjusted Sequencing Rule is now:

$$\exists x : (xbp + xc = xbp \wedge xcq + xd = xcq \rightarrow xbpq + xd = xbpq) \tag{32}$$

The proofs will all be valid with a slight modification. This will eliminate the $\top$, but it brings it own share of problems. The equation is only meaningful if $xc \neq 0$ and $xd \neq 0$. In addition, x is not a Boolean. If we find x, it wouldn't necessarily tell us what inputs would result in an error.

Earlier we had the formula $\top pc \sqsubseteq \top bp$. Here both sides have p, so the x replacing $\top$ could be a Boolean. So maybe this formula works:

$$\exists x \in \mathcal{B} : (xbp + xpc = xbp) \tag{33}$$

But with this formula the Sequence Rule cannot be proven.

Still, we are now moving the goalposts. To prove that the triple $[b]p[ok : c]$ has a bug, we would need to find an x such that:

- $xc \neq 0$.
- for every i, the result of x is an input for both $b_i p_i$ and for c_i.
- $xc \neq 0$ implies that for every i, $xc_i \neq 0$.
- x is not an input (Boolean Algebra), but has to be a program (Kleene Algebra).

To construct such a x or to prove that such an x exist, is more difficult that simply checking all paths.

Creation from within

We could create $\top$ for KAT by taking the transitive closure of $\mathcal{K}$. This definition of $\top$ would be bigger than all elements in $\mathcal{K}$.

We cannot take the reflexive transitive closure of an infinite set, so we use a workaround. Let Σ be the carrier of the set $\mathcal{K}$. In other words, Σ is a finite set that recursively generated $\mathcal{K}$. This set is finite, because hopefully the program that the set is based on is finite. We can define $\top$ as the universal expression:

$$\top = (p_1 + p_2 + \cdots + p_n)^*$$

where $\Sigma = \{p_1, p_2, \ldots, p_n\}$.

It holds that $\forall x \in \mathcal{K} : x \sqsubseteq \top$.

The proof uses x', which is variable x without Booleans. It's true that $\forall x : x \sqsubseteq x'$, because for all Boolean holds: $b \sqsubseteq 1$. Then $\forall x' : x' \sqsubseteq \top$ follows trivially from the definition.

The problem is that this $\top$ may not be the same $\top$ from Relational Algebra.

6 Conclusion

I found the translation from Incorrectness Logic to KAT and I have proven the soundness of the translation. I couldn't prove the completeness, but that was a more difficult task.

The author of Incorrectness Logic notes that with correctness reasoning you may forget information on a path, but you must remember all paths. For incorrectness reasoning you must remember the information, but you may forget some of the paths. What does this mean in the context of KAT? Is forgetting information the act of multiplying with $\bar{c}$ to get $bp\bar{c} = 0$ and does that mean that the alternative formulas for Incorrectness are useless, because I multiplied a constant?

References

[1] Peter W. O'Hearn. Incorrectness logic. *Proceedings of the ACM on Programming Languages*, 4, POPL, Article 10 (January 2020):1–32, 2020.

[2] Dexter Kozen Frederick Smith. Kleene algebra with tests: Completeness and decidability. *In: van Dalen D., Bezem M. (eds) Computer Science Logic. CSL 1996. Lecture Notes in Computer Science*, vol 1258, 2005.

[3] Dexter Kozen. On hoare logic and kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 1(1):60–76, 2000.

[4] FJ Rietman. The secrets of causality. Technical Report RUU-CS-93-29, Department of Information and Computing Sciences, Utrecht University, 1993.

[5] Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, 1997.

[6] Doumane A. Kuperberg D. Pous D. Pradic P. Kleene algebra with hypotheses. *In: Bojańczyk M., Simpson A. (eds) Foundations of Software Science and Computation Structures. FoSSaCS 2019. Lecture Notes in Computer Science*, 11425, 2019.