Hoare Logics for Time Bounds

A Study in Meta Theory

Maximilian P. L. Haslbeck and Tobias Nipkow*

Technische Universität München
haslbema@in.tum.de
http://www.in.tum.de/~{haslbema,nipkow}

Abstract. We study three different Hoare logics for reasoning about time bounds of imperative programs and formalize them in Isabelle/HOL: a classical Hoare like logic due to Nielson, a logic with potentials due to Carbonneaux *et al.* and a *separation logic* following work by Atkey, Chaguérand and Pottier. These logics are formally shown to be sound and complete. Verification condition generators are developed and are shown sound and complete too. We also consider variants of the systems where we abstract from multiplicative constants in the running time bounds, thus supporting a big-O style of reasoning. Finally we compare the expressive power of the three systems.

Keywords: Hoare Logic, Algorithm Analysis, Program Verification

1 Introduction

This paper is about Hoare logics for proving upper bounds on running times and about the formalized (in a theorem prover) study of their meta theory. The paper is not about the automatic analysis of running times but about fundamental questions like soundness and completeness of logics and of verification condition generators (VCGs). The need for such a study becomes apparent when browsing the related literature (e.g. [1, 6, 7]): (formalized) soundness results are of course provided, but completeness of logics and VCGs is missing.

We study multiple different Hoare logics because we are interested in different aspects of the logics. One aspect is the difference between precise upper bounds and order-of-magnitude upper bounds that abstract from multiplicative constants. In the latter case we speak of "big-O style" logics.

A second aspect is modularity. We would like to combine verified results about subprograms in order to show correctness and running time for larger programs. Therefore we also study a separation logic for running time analysis.

Overall we study the meta theory of three different kinds of Hoare logics that have emerged in the literature. Our main contributions are:

* Supported by DFG GRK 1480 (PUMA) and Koselleck Grant NI 491/16-1

- Based on the simple imperative language IMP (Section 2), we formalize three logics for time bounds from the literature (Section 3); we show their soundness and completeness w.r.t. IMP's semantics, discuss specific weaknesses and strengths and study their interrelations (Section 4).
- The first logic we study is a big-O style logic due to Nielson [23] (Section 3.1). We improve, formalize and verify this logic and extend it with a VCG whose soundness and completeness we also verify.
- In Section 3.2 we formalize a quantitative Hoare logic following ideas by Carbonneaux *et al.* [4, 6] and extend their work as follows: we prove completeness of the logic and design a sound and complete VCG. Additionally we extend the logic to a big-O style logic.
- Following ideas of Atkey [1] and Charguéraud and Pottier [7] we formalize a logic similar to separation logic (Section 3.3) for reasoning about concrete running times. We formally prove soundness and completeness.
- All proofs have been formalized in Isabelle/HOL [19, 18] and are available online [9].

2 Basics

We consider the simple deterministic imperative language IMP. Its formalization is standard and can be found elsewhere [18]. IMP's commands are built up from SKIP, assignment, sequential composition, conditional and While-loop. Program states are functions from variables to values. By default c is a command and s a state. Evaluation of a boolean or arithmetic expression e in state s is denoted by $[\![e]\!]_s$.

We have defined a big-step semantics that counts the consumed time during execution: SKIP, assignment and evaluation of boolean expressions require one time unit. The precise definition of the semantics is routine. We write $(c, s) \overset{t}{\Rightarrow} s'$ to mean that starting command c in state s terminates after time t in state s'.

Given a pair (c, s), $\downarrow(c, s)$ means that the computation of c starting from s terminates, $\downarrow_S(c, s)$ then denotes the final state, and $\downarrow_T(c, s)$ the execution time.

3 Hoare Logics for Time Bounds

In this section we study and extend three different Hoare logics: a classical one based on [23], one using potentials [4] and one based on separation logic with time credits [1].

3.1 Nielson style

Nielson and Nielson [23] present a Hoare logic to prove the "order of magnitude of the execution time" of a program (which we call "big-O style"). They reason about triples of the form $\{P\}c\{e \Downarrow Q\}$ where P and Q are assertions and e is a time bound. The intuition is the following: if the execution of command c is

started in a state satisfying P then it terminates in a state satisfying Q after $O(e)$ time units, i.e. the execution time has order of magnitude e. Note that e is evaluated in the state before executing c.

Throughout the paper we rely on what is called a *shallow* embedding of assertions and time bounds: there is no concrete syntactic representation of assertions and time bounds but they are merely functions in HOL, our ambient logic. They map states to truth values and natural numbers.

A complication in reasoning about execution time comes from the fact that one needs to combine time bounds that refer to different points in the execution, for example when adding time bounds in a sequential composition. This difficulty can be overcome with *logical variables* that enable us to transport time bounds from the prestate to the poststate of a command. We formalize logical variables by modelling assertions as functions of two states, the state of the logical variables (typically l) and the state of the program variables (typically s).

The validity of Nielson's triples is formally defined as follows:

$$\models_1 \{P\}c\{e \Downarrow Q\} \equiv (\exists k.\forall l\ s.\ P\ l\ s \longrightarrow (\exists t\ s'.\ (c,s) \stackrel{t}{\Rightarrow} s' \wedge Q\ l\ s' \wedge t \leq k \cdot e\ s))$$

The Hoare logic below needs to generate "fresh" logical variables. Thus we need to express which logical variables are already used. This is called the *support* of an assertion. Because assertions are merely functions, the support is defined semantically:

$$\mathit{support}\ Q\ \equiv\ \{x \mid \exists l_1\ l_2\ s.\ (\forall y.\ y \neq x \longrightarrow l_1\ y = l_2\ y) \wedge Q\ l_1\ s \neq Q\ l_2\ s\}$$

Our Hoare logic is shown in Figure 1. It is largely a formalization of the system in [23, Table 10.4] but with two important changes: we have simplified rule *While* (details below) and we have replaced the consequence rule by conseq_K, an adaptation of Kleymann's stronger consequence rule [15]; rules *conseq* and *const* are derived from it. Note that the latter two rules suffice for a sound and complete Hoare logic, but our proof of completeness of the VCG needs conseq_K.

Now we discuss the rules in Figure 1. Rules *Skip*, *Assign*, *If* and *conseq* are straightforward. Note that $\mathbf{1}$ is the time bound $\lambda s.1$ and $+$ is lifted to time bounds pointwise. The notation $s[a/x]$ is short for "s with x mapped to $[\![a]\!]_s$".

Now consider rule *Seq*. Given $\{P\}c_1\{e_1 \Downarrow Q\}$ and $\{Q\}c_2\{e_2 \Downarrow R\}$ one may want to conclude $\{P\}c_1;c_2\{e_1 + e_2 \Downarrow R\}$. Unfortunately, $e_1 + e_2$ does not lead to the correct result, as c_1 could have altered variables e_2 depends on. In order to adapt e_2 for the changes that occur in c_1, we use a shifted time bound e_2', and leave as a proof goal to show that the value of e_2' in the prestate is an upper bound on e_2 in the poststate of c_1. Rule *Seq* relates e_2' and e_2 through a fresh logical variable u that is equated with the value of e_2' in the prestate of c_1. The time bound e in the conclusion must be an upper bound of $e_1 + e_2'$.

In the *const* rule, the time bound can be reduced by a constant factor. Note that we split up Nielson's cons_e rule into *conseq* and *const*.

Our rule *While* is a simplification of the one in [23]. The latter is an extension with time of the "standard" While-rule for total correctness where a variable

$$\frac{}{\vdash_1 \{P\}\text{SKIP}\{\mathbf{1} \Downarrow P\}}\,Skip \qquad \frac{}{\vdash_1 \{\lambda l\ s.\ P\ l\ (s[a/x])\}x := a\{\mathbf{1} \Downarrow P\}}\,Assign$$

$$\frac{\vdash_1 \{\lambda l\ s.\ P\ l\ s \wedge [\![b]\!]_s\}c_1\{e \Downarrow Q\} \quad \vdash_1 \{\lambda l\ s.\ P\ l\ s \wedge \neg[\![b]\!]_s\}c_2\{e \Downarrow Q\}}{\vdash_1 \{P\}\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2\{e + \mathbf{1} \Downarrow Q\}}\,If$$

$$\frac{\begin{array}{c}\vdash_1 \{\lambda l\ s.\ P\ l\ s \wedge e_2'\ s = l\ u\}c_1\{e_1 \Downarrow \lambda l\ s.\ Q\ l\ s \wedge e_2\ s \le l\ u\} \quad \vdash_1 \{Q\}c_2\{e_2 \Downarrow R\} \\ (\forall l\ s.\ P\ l\ s \Longrightarrow e_1\ s + e_2'\ s \le e\ s) \quad u \notin support\ P \quad u \notin support\ Q\end{array}}{\vdash_1 \{P\}c_1; c_2\{e \Downarrow R\}}\,Seq$$

$$\frac{\begin{array}{c}\vdash_1 \{\lambda l\ s.\ I\ l\ s \wedge [\![b]\!]_s \wedge e'\ s = l\ u\}c\{e'' \Downarrow \lambda l\ s.\ I\ l\ s \wedge e\ s \le l\ u\} \\ (\forall l\ s.\ I\ l\ s \wedge [\![b]\!]_s \longrightarrow e\ s \ge 1 + e'\ s + e''\ s) \\ (\forall l\ s.\ I\ l\ s \wedge \neg[\![b]\!]_s \longrightarrow e\ s \ge 1) \quad u \notin support\ I\end{array}}{\vdash_1 \{I\}\text{WHILE } b \text{ DO } c\{e \Downarrow \lambda l\ s.\ I\ l\ s \wedge \neg[\![b]\!]_s\}}\,While$$

$$\frac{\begin{array}{c}\forall l\ s.\ P'\ l\ s \longrightarrow P\ l\ s \\ \vdash_1 \{P\}c\{e \Downarrow Q\} \\ \forall l\ s'.\ Q\ l\ s' \longrightarrow Q'\ l\ s'\end{array}}{\vdash_1 \{P'\}c\{e' \Downarrow Q'\}}\,conseq \qquad \frac{\begin{array}{c}\exists k.\forall l\ s.\ P\ l\ s \longrightarrow e\ s \le k \cdot e'\ s \\ \vdash_1 \{P\}c\{e \Downarrow Q\}\end{array}}{\vdash_1 \{P\}c\{e' \Downarrow Q\}}\,const$$

$$\frac{\begin{array}{c}\exists k.\forall l\ s.\ P'\ l\ s \longrightarrow (e\ s \le k \cdot e'\ s \wedge (\forall s'.\exists l'.\ P\ l's \wedge (Q\ l's' \longrightarrow Q'\ l\ s'))) \\ \vdash_1 \{P\}c\{e \Downarrow Q\}\end{array}}{\vdash_1 \{P'\}c\{e' \Downarrow Q'\}}\,conseq_K$$

Fig. 1: Hoare logic for reasoning about order of magnitude of execution time

decreases with each loop iteration. However, once you have time, you no longer need that variable and we removed it. The key constraint in rule *While* is $e \ge 1 + e' + e''$. It can be explained by unfolding the loop once. The time e to execute the whole loop must be an upper bound for the time e'' to execute the loop body plus the time e' to execute the remaining loop iteration; the 1+ accounts for evaluation of b. The time e' to execute the remaining loop iterations is obtained from e by (intuitively) an application of rule *Seq*: in the first premise a fresh logical variable u is used to pull e back over c, resulting in e'. The rest of rule *While* is standard.

Soundness of the calculus can be shown by induction on the derivation of $\vdash_1 \{P\}c\{e \Downarrow Q\}$:

Theorem 1 (Soundness of $\vdash_1$). $\vdash_1 \{P\}c\{e \Downarrow Q\} \Longrightarrow \models_1 \{P\}c\{e \Downarrow Q\}$

Our completeness proof follows the general pattern for Hoare logics: define a weakest precondition operator *wp* and show that the triple $\{wp\ c\ Q\}c\{Q\}$ is derivable. In our setting *wp* is defined like this

$$wp\ c\ Q\ \equiv\ (\lambda l\ s.\ \exists t\ s'.\ (c, s) \overset{t}{\Rightarrow} s' \wedge Q\ l\ s')$$

and we show derivability of the following triple that also takes time into account:

Lemma 1. $finite(support\ Q) \Longrightarrow \vdash_1 \{wp\ c\ Q\}c\{\lambda s.\ \downarrow_T(c, s) \Downarrow Q\}$

As we need fresh logical variables for rules *Seq* and $While$, we assume that the set of logical variables Q depends on is finite.

It is instructive to observe that for this proof, only the Hoare rules *Skip* to *conseq* are needed. Neither *const* nor $conseq_K$ are used. Lemma 1 thus expresses that it always is possible to derive a triple with the precise execution time as a time bound. Only as a last step an abstraction of multiplicative constants and over-approximation of the time bound is necessary. This shows that for every valid triple one can first deduce a correct upper bound for the running time, only to get rid of a multiplicative constant in a final application of the *const* rule one. In the end, Lemma 1 implies completeness:

Theorem 2 (Completeness of $\vdash_1$).
$finite\ (support\ Q) \Longrightarrow \models_1 \{P\}c\{e \Downarrow Q\} \Longrightarrow \vdash_1 \{P\}c\{e \Downarrow Q\}$

In particular we can now apply the above observation about the shape of derivations of valid triples to provable ones, by soundness: in any derivation one can pull out all applications of *const* and combine them into a single one at the very root of the proof tree. We will observe the very same principle when studying the quantitative Hoare logic in Section 3.2.

Verification Condition Generator Showing validity of $\{P\}c\{e \Downarrow Q\}$ now boils down to applying the correctly instantiated rules of the Hoare logic and proving their side conditions. The former is a mechanical task, which is routinely automated by a verification condition generator, while the latter is left to an automatic or interactive theorem prover.

We design a VCG that collects the side conditions for an annotated program. While for classical Hoare logic it suffices to annotate a loop with an invariant I, for reasoning about execution time we introduce two more annotations for the following reason.

Consider rule *Seq* in Figure 1. When applying the rule to a proof goal $\vdash_1 \{P\}c_1;c_2\{e \Downarrow R\}$ we need to instantiate the variables P, Q, e_1, e_2, and e_2'. As for classical Hoare logic, Q is chosen to be the weakest preconditions of c_2 w.r.t. R, which can be calculated if the loops in c_2 are annotated by invariants. (Analogously for P being the weakest precondition of c_1 w.r.t. Q). Similarly, when annotating the loops in c_1 and c_2 with time bounds E, time bounds e_1 and e_2 can be constructed. Finally, e_2' can be determined if the evolution of e_2 through c_1 is known. For straight line programs, this can be deduced, only for loops a state transformer S has to be annotated. An annotated loop then has the form $\{I, S, E\}$ WHILE b DO C where I is the invariant and S and E are as above.

For our completeness proof of the VCG we also need annotations that correspond to applications of rule $conseq_K$ and record information that cannot be inferred automatically. For that purpose we introduce a new annotated command $Conseq\ \{P', Q, e'\}\ C$ where P', Q and e' are as in rule $conseq_K$.

We use capital letters, e.g. C, to denote annotated commands and $\overline{C}$ is the unannotated version of C stripped of all annotations.

We use three auxiliary functions *pre*, *post* and *time*. Their definitions are shown in Figure 2.

$$
\begin{array}{ll}
pre\ \text{SKIP}\ Q = Q & post\ \text{SKIP}\ s = s \\
pre\ (x := a)\ Q = (\lambda ls.\ Q\ l\ (s[a/x])) & post\ (x := a)\ s = s[a/x] \\
pre\ (C_1; C_2)\ Q = pre\ C_1\ (pre\ C_2\ Q) & post\ (C_1; C_2)\ s = post\ C_2\ (post\ C_1\ s) \\
pre\ (Conseq\ \{P', \_, \_\}\ C)\ Q = P' & post\ (Conseq\ \{\_, \_, \_\}\ C) = post\ C \\
pre\ (\text{IF}\ b\ \text{THEN}\ C_1\ \text{ELSE}\ C_2)\ Q\ l\ s = & post\ (\text{IF}\ b\ \text{THEN}\ C_1\ \text{ELSE}\ C_2)\ s = \\
\ \text{if}\ [\![b]\!]_s\ \text{then}\ pre\ C_1\ Q\ l\ s\ \text{else}\ pre\ C_2\ Q\ l\ s & \ \text{if}\ [\![b]\!]_s\ \text{then}\ post\ C_1\ s\ \text{else}\ post\ C_2\ s \\
pre\ (\{I, \_, \_\}\ \text{WHILE}\ b\ \text{DO}\ C)\ Q = I & post\ (\{\_, S, \_\}\ \text{WHILE}\ b\ \text{DO}\ C) = S
\end{array}
$$

$$
\begin{array}{l}
time\ \text{SKIP}\ s = 1 \\
time\ (x := a)\ s = 1 \\
time\ (C_1; C_2)\ s = time\ C_1\ s + time\ C_2\ (post\ C_1\ s) \\
time\ (Conseq\ \{\_, \_, \_\}\ C) = time\ C \\
time\ (\text{IF}\ b\ \text{THEN}\ C_1\ \text{ELSE}\ C_2)\ s = \\
\ \text{if}\ [\![b]\!]_s\ \text{then}\ time\ C_1\ s\ \text{else}\ time\ C_2\ s \\
time\ (\{\_, \_, E\}\ \text{WHILE}\ b\ \text{DO}\ C) = E
\end{array}
$$

Fig. 2: Functions *pre*, *post* and *time*

The VCG reduces proving a triple $\{P\}\overline{C}\{e \Downarrow Q\}$ to checking that the annotations really are invariants, upper bounds and correct state transformers. The VCG traverses C and collects all the verification conditions for the loops into a big conjunction. The most interesting case is the loop itself:

$$
\begin{array}{l}
vc\ (\{I, S, E\}\ \text{WHILE}\ b\ \text{DO}\ C)\ Q = vc\ C\ I\ \wedge \\
\quad (\forall l\ s.\ (I\ l\ s \wedge [\![b]\!]_s \longrightarrow pre\ C\ I\ l\ s \\
\qquad\qquad\qquad\qquad \wedge E\ s \geq 1 + E(post\ C\ s) + time\ C\ s \\
\qquad\qquad\qquad\qquad \wedge S\ s = S(post\ C\ s)) \\
\qquad \wedge\ (I\ l\ s \wedge \neg [\![b]\!]_s \longrightarrow Q\ l\ s \wedge E\ s \geq 1 \wedge S\ s = s))
\end{array}
$$

First, verification conditions are recursively generated from the loop body C and the invariant I as desired post condition. The invariant and the loop guard must imply preservation of the invariant, the recurrence inequation for the time bound and that the state transformer S obeys the fixpoint equation for loops. When exiting the loop, the post condition must hold, E has to pay for the last test of the loop guard, and S needs to be the identity.

The verification conditions for *Conseq* $\{P', Q, e'\}$ C merely check the side condition of rule $conseq_K$:

$$\begin{aligned} vc\ (Conseq\ \{P', Q, e'\}\ C)\ Q' = vc\ C\ Q\ \wedge \\ \exists k.\ \forall l\ s.\ P'\ l\ s \longrightarrow time\ C\ s \leq k \cdot e'\ s \\ \wedge\ \forall t. \exists l'.\ pre\ C\ Q\ l'\ s \wedge (Q\ l'\ t \longrightarrow Q'\ l\ t) \end{aligned}$$

The remaining equations for vc are straightforward:

$$\begin{aligned} &vc\ \text{SKIP}\ Q = True \\ &vc\ (x := a)\ Q = True \\ &vc\ (C_1; C_2)\ Q = (vc\ C_1\ (pre\ C_2\ Q) \wedge vc\ C_2\ Q) \\ &vc\ (\text{IF}\ b\ \text{THEN}\ C_1\ \text{ELSE}\ C_2)\ Q = (vc\ C_1\ Q \wedge vc\ C_2\ Q) \end{aligned}$$

Theorem 3 (Soundness of vc). *Let C and Q involve only finitely many logical variables. Then $vc\ C\ Q$ together with $\exists k. \forall l\ s.\ P\ l\ s \longrightarrow pre\ C\ Q\ l\ s \wedge time\ C\ s \leq k \cdot e\ s$ imply $\vdash_1 \{P\}\ \overline{C}\{e \Downarrow Q\}$.*

That is, for proving $\vdash_1 \{P\}\ \overline{C}\{e \Downarrow Q\}$ one has to show the verification conditions, that P implies the weakest precondition (as computed by *pre*) and that the running time (as computed by *time*) is in the order of magnitude of e.

Now we come to the *raison d'être* of the stronger consequence rule $conseq_K$: the completeness proof of our VCG. The other proofs in this section only require the derived rules *conseq* and *const*. Our completeness proof of the VCG builds annotated programs that contain a *Conseq* construct for every *Seq* and *While* rule. The annotations of *Conseq* enable us to adapt the logical state; without this adaptation we failed to generate true verification conditions.

Theorem 4 (Completeness of vc). *If $\vdash_1 \{P\}\ c\{e \Downarrow Q\}$ then there is a C such that $\overline{C} = c$, $vc\ C\ Q$ is true and $\exists k. \forall l\ s.\ P\ l\ s \longrightarrow pre\ C\ Q\ l\ s \wedge time\ C\ s \leq k \cdot e\ s$.*

That is, if a triple $\vdash_1 \{P\}\ c\{e \Downarrow Q\}$ is provable then c can be annotated such that the verification conditions are true, P implies the weakest precondition (as computed by *pre*) and the running time (as computed by *time*) is in the order of magnitude of e.

Annotating loops with a correct S is troublesome, as it captures the semantics of the whole loop. Luckily S only needs to be correct for "interesting" variables, i.e. variables that occur in time bounds that need to be pulled backward through the loop body. Often these variables are not modified by a command. We implemented an optimized VCG that keeps track of which variables are of interest and requires S to be correct only on those; we also showed its soundness and completeness. Further details can be found in the formalization.

3.2 Quantitative Hoare Logic

The main idea by Carbonneaux *et al.* [4] is to generalize predicates (state $\Rightarrow \mathbb{B}$) in Hoare triples to *potentials* (state $\Rightarrow \mathbb{N}_\infty$). That is, Hoare triples are now of

the form $\{P\}c\{Q\}$ where P and Q are potentials. The resulting logic does not need logical variables. We prove soundness and completeness of that logic and design a sound and complete VCG. Then we extend the logic and VCG to big-O style reasoning.

Validity of triples involving potentials is defined as follows and is a direct generalization of validity for triples involving predicates:

$$\models_2 \{P\}c\{Q\} \equiv \forall s.\ P\ s < \infty \longrightarrow (\exists t\ s'.\ (c,s) \overset{t}{\Rightarrow} s' \wedge Q\ s' < \infty \wedge P\ s \geq t + Q\ s')$$

One may interpret the refinement from $\mathbb{B}$ to $\mathbb{N}_\infty$ as follows: infinite potentials are "impossible" and thus correspond to *False*, while finite potentials correspond to *True*. In that way "$P\ s < \infty$" corresponds to "P holds in state s". Furthermore, we interpret the difference of the prepotential P and postpotential Q as an upper bound on the actual running time. Predicates can be lifted to potentials by mapping *True* to 0 and *False* to ∞. We use the $\uparrow$ symbol for that lifting: $\uparrow P\ s \equiv$ (if $P\ s$ then 0 else ∞), and similarly for boolean expressions: $\uparrow b\ s \equiv$ (if $[\![b]\!]_s$ then 0 else ∞).

$$\frac{}{\vdash_2 \{P + \mathbf{1}\}\text{SKIP}\{P\}}\ \textit{Skip} \qquad \frac{}{\vdash_2 \{\lambda s.1 + P(s[a/x])\}x := a\{P\}}\ \textit{Assign}$$

$$\frac{\vdash_2 \{P + \uparrow b\}c_1\{Q\} \quad \vdash_2 \{P + \uparrow(\neg b)\}c_2\{Q\}}{\vdash_2 \{P + \mathbf{1}\}\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2\{Q\}}\ \textit{If} \quad \frac{\vdash_2 \{P\}c_1\{Q\} \quad \vdash_2 \{Q\}c_2\{R\}}{\vdash_2 \{P\}c_1; c_2\{R\}}\ \textit{Seq}$$

$$\frac{\vdash_2 \{I + \uparrow b\}c\{I + \mathbf{1}\}}{\vdash_2 \{I + \mathbf{1}\}\text{WHILE } b \text{ DO } c\{I + \uparrow(\neg b)\}}\ \textit{While} \quad \frac{P' \geq P \quad \vdash_2 \{P\}c\{Q\} \quad Q \geq Q'}{\vdash_2 \{P'\}c\{Q'\}}\ \textit{conseq}$$

Fig. 3: Quantitative Hoare logic

The rules in Figure 3 define the Hoare logic $\vdash_2$ corresponding to $\models_2$. Note that $P \geq Q$ is short for $\forall s.\ P\ s \geq Q\ s$.

Rules *Skip*, *Assign* and *If* are straightforward; the 1 time unit added to the prepotential pays for, respectively, SKIP, assignment and the evaluation of the boolean expression. The *conseq* rule also looks familiar, only that $\longrightarrow$ has been replaced by $\geq$. You can think of a bigger potential implying a smaller one; also remember that $False$ corresponds to ∞.

For the *While* rule, assume one can derive that having the potential I and a true guard b before the execution of c implies a postpotential one more than the invariant I (the plus one is needed for the upcoming evaluation of the guard, which incurs cost 1), then one can conclude that, starting the loop with potential $I+\mathbf{1}$ (again the plus one pays for the evaluation of the guard), the loop terminates with a potential equal to I and the negation of the guard holds in the final state. Although this rule resembles the *While* rule for partial correctness, the decreasing potential actually also ensures termination.

Theorem 5 (Soundness of $\vdash_2$). $\vdash_2 \{P\}c\{Q\} \Longrightarrow \models_2 \{P\}c\{Q\}$

For proving completeness, we generalise the *weakest precondition* to the *weakest prepotential*:

$$wp\ c\ Q\ s \equiv (\text{if } \downarrow(c,s) \text{ then } \downarrow_T(c,s) + Q(\downarrow_S(c,s)) \text{ else } \infty)$$

In fact, *wp* is also a (weakest) prepotential w.r.t. provability:

Lemma 2. $\vdash_2 \{wp\ c\ Q\}c\{Q\}$

As usual, completeness follows easily from this lemma:

Theorem 6 (Completeness of $\vdash_2$). $\models_2 \{P\}c\{Q\} \Longrightarrow \vdash_2 \{P\}c\{Q\}$

Verification Condition Generator The simpler *Seq* rule (compared to $\vdash_1$) leads to a more compact VCG. Loops are simply annotated with invariants, which now are potentials. No *Conseq* annotations are required.

Function *pre C Q* determines the weakest prepotential of an annotated program C and postpotential Q. Its definition is by recursion on annotated commands and refines our earlier *pre* on predicates.

The VCG recursively traverses the command and collects the verification conditions at the loops (we omit the other cases of *vc*):

$$\begin{aligned} &vc\ (\{I\}\text{WHILE } b \text{ DO } C)\ Q = \\ &\ I + \uparrow b \geq pre\ C\ (I + \mathbf{1})\ \wedge\ I + \uparrow(\neg b) \geq Q\ \wedge\ vc\ C\ (I + \mathbf{1}) \end{aligned}$$

The two first conjuncts express invariant preservation and that the invariant "implies" the postcondition when exiting the loop. Soundness of the VCG is established by induction on the command.

Lemma 3 (Soundness of *vc*). *If we can show the verification conditions vc C Q and that we have at least as much potential as the needed prepotential ($P \geq$ pre C Q) then we can derive* $\vdash_2 \{P\}\overline{C}\{Q\}$.

Completeness of the VCG can be paraphrased like this: if we can derive the Hoare Triple $\vdash_2 \{P\}c\{Q\}$, we can find an annotation for c such that the verification conditions are true and P "implies" the prepotential.

Lemma 4 (Completeness of *vc*).
$\vdash_2 \{P\}c\{Q\} \Longrightarrow \exists C.\ \overline{C} = c\ \wedge\ vc\ C\ Q\ \wedge\ P \geq pre\ C\ Q$

Constant factors As for the Nielson system we can extend the quantitative Hoare logic to reason about the order of magnitude of execution time. We generalize our notion of validity from $\models_2$ to $\models_{2'}$:

$$\models_{2'} \{P\}c\{Q\} \equiv \exists k > 0.\forall s.\ P\ s < \infty \longrightarrow \exists t\ s'.\ \begin{cases} (c,s) \Rightarrow t \Downarrow s' \wedge Q\ s' < \infty \wedge \\ k \cdot P\ s \geq t + k \cdot Q\ s' \end{cases}$$

For intuition, assume Q is zero: then the triple is valid iff the running time t is bounded by k times the prepotential P. This amounts to O-notation.

Correspondingly we extend the set of Hoare rules $\vdash_2$ in Figure 3 to $\vdash_{2'}$ by adding the following rule:

$$\frac{\vdash_{2'} \{\lambda s.\ k \cdot P\ s\}c\{\lambda s.\ k \cdot Q\ s\} \qquad k > 0}{\vdash_{2'} \{P\}c\{Q\}}\ const$$

For re-establishing soundness we can adapt the proof of Theorem 5 by catering for constants and adding one more case for rule *const*.

Theorem 7 (Soundness of $\vdash_{2'}$). $\vdash_{2'} \{P\}c\{Q\} \Longrightarrow \models_{2'} \{P\}c\{Q\}$

For the completeness proof, nothing changes. We reuse the same *wp* and the proof of $\vdash_{2'} \{wp\ c\ Q\}c\{Q\}$ is identical to that of Lemma 2 because we extended the Hoare rules, but not the command language. In particular this means that the new *const* rule is not used in this proof. The same principle as in section 3.1 applies: the *const* rule is only used once at the end when showing completeness from $\vdash_{2'} \{wp\ c\ Q\}c\{Q\}$:

Theorem 8 (Completeness of $\vdash_{2'}$). $\models_{2'} \{P\}c\{Q\} \Longrightarrow \vdash_{2'} \{P\}c\{Q\}$

VCG with constants For the VCG we add one more annotated command $Const\ \{k\}\ C$ (where $k \in \mathbb{N}$, $k > 0$). It signals the application of a *const* rule. We reuse the old definitions of *pre* and *vc* but add new equations for *Const*:

$$vc\ (Const\ \{k\}\ C)\ Q\ s = (vc\ C\ (\lambda s.\ k \cdot Q\ s) \wedge k > 0)$$
$$pre\ (Const\ \{k\}\ C)\ Q\ s = ediv\ (pre\ C\ (\lambda s.\ k \cdot Q\ s)\ s)\ k$$

The definition of $vc\ (Const\ \{k\}\ C)\ Q$ expresses that the execution of C must leave a potential of $k \cdot Q$ instead of just Q. The definition of $pre\ (Const\ \{k\}\ C)\ Q$ expresses that we pull back a potential of $k \cdot Q$ but that in the end we renormalize the prepotential by dividing (function *ediv*) by k. More precisely, *ediv* is integer division which rounds up for non integral results and is lifted to $\mathbb{N}_\infty$.

The soundness and completeness proofs must only be adapted marginally, only some algebraic lemmas about *ediv* are needed.

To summarize this section: we have shown how to generalize conditions to potentials, thus obtaining a compositional Hoare logic; we have extended the Hoare logic to big-O style reasoning and have adapted the calculus and proofs; we also have established sound and complete VCGs for both logics.

One drawback of the quantitative Hoare logic is that it is not modular. Imagine two independent programs c_1 and c_2 that are run one after the other. When reasoning about a subprogram c_1 we need to specify a postpotential that is then used for the following program c_2. If we change c_2, resulting in a changed time consumption, also the analysis for c_1 has do be redone. What we actually would like to do, is to reason about c_1 and c_2 locally and then combine them in a final step. Separation logic addresses this issue.

3.3 Separation Logic with Time Credits

Our last logic follows the idea by Atkey [1] to use separation logic in order to reason about the resource consumption of programs. This logic generalizes the quantitative Hoare logic.

The principle of "local reasoning" is addressed by separation logic for disjoint heap areas; Atkey [1] uses separation logic with time credits to reason about the amortised execution time of (imperative) programs.

In this section we follow his ideas and design a Hoare logic based on separation logic. As IMP does not have a heap to reason about, but we want to compare the logic to the two logics we already described, we treat the state of a program as a kind of heap: a *partial state ps* is a map from variable names to values, *dom ps* is the domain of *ps*, we call ps_1 and ps_2 *disjoint* ($ps_1 \perp ps_2$) if their domains are, and we can add two partial states to form their disjoint union ($ps_1 + ps_2$).

We adapt evaluation of arithmetic and boolean expressions, as well as the big-step semantics (now denoted by $\Rightarrow_p$) to partial states. If all necessary variables are in the domain of the partial state *ps*, these new constructs coincide with their counterparts on (full) states. The new big-step semantics rule for assignment for example has an additional premise. All other rules are similar.

$$\frac{vars\ a \cup \{x\} \subseteq dom\ ps}{(x := a, ps) \overset{1}{\Rightarrow}_p ps(x \mapsto [\![a]\!]_{ps})} Assign$$

The new semantics admit a frame rule: we can always add disjoint partial states, without affecting the computation.

Lemma 5. $$\frac{(c, ps_1) \overset{t}{\Rightarrow}_p ps_1' \qquad ps_1 \perp ps_2}{(c, ps_1 + ps_2) \overset{t}{\Rightarrow}_p ps_1' + ps_2}$$

In that way we treat the set of variables as resources, on which separation logic can work. Additionally, as Atkey proposes, we add time credits as resources: we consider *configurations* (ps, n) which are pairs of partial states and natural numbers. Natural numbers, viewed as resources, are always disjoint and can be added; thus they form a separation algebra [2]. A pair of separation algebras is again a separation algebra. For predicates on configurations we thus have the $*$ operator from separation algebra

$$(P * Q)(ps, n) \equiv \exists ps_1\ n_1\ ps_2\ n_2. \begin{cases} ps = ps_1 + ps_2 \wedge n = n_1 + n_2 \wedge ps_1 \perp ps_2 \wedge \\ P(ps_1, n_1) \wedge Q(ps_2, n_2) \end{cases}$$

meaning that we can split up the configuration into two disjoint configurations; one satisfying P and the other satisfying Q. Our formalization builds on an existing Isabelle/HOL theory of separation algebras [14].

The validity of a Hoare triple is defined in the following way:

$$\models_3 \{P\}c\{Q\} \equiv \forall ps\ n.\ P(ps, n) \longrightarrow \exists ps'\ n'\ t. \begin{cases} (c, ps) \overset{t}{\Rightarrow}_p ps' \wedge \\ n = n' + t \wedge Q(ps', n') \end{cases}$$

We can now state the Hoare rules for this logic, see Figure 4. Note that $\$n$ denotes the configuration of an empty partial state and n time resources, $(b \hookrightarrow B)\ ps$ is true, iff all variables in b are in the domain of ps and b evaluates to B in ps. Updating the partial state ps with value v for x is denoted by $ps(x \mapsto v)$.

$$\frac{}{\vdash \{\$1\}\text{SKIP}\{\$0\}}\,Skip$$

$$\frac{}{\vdash \{(\lambda(ps,t).\ \{x\} \cup vars\ a \subseteq dom\ ps \wedge Q\ (ps(x \mapsto [\![a]\!]_{ps}), t)) * \$1\}x := a\{Q\}}\,Assign$$

$$\frac{\begin{array}{c}\vdash \{\lambda(ps,n).\ P(ps,n) \wedge (b \hookrightarrow True)\ ps\}c_1\{Q\} \\ \vdash \{\lambda(ps,n).\ P(ps,n) \wedge (b \hookrightarrow False)\ ps\}c_2\{Q\}\end{array}}{\vdash \{(\lambda(ps,n).\ P(ps,n) \wedge vars\ b \subseteq dom\ ps) * \$1\}\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2\{Q\}}\,If$$

$$\frac{\vdash \{P\}c\{Q\}}{\vdash \{P * F\}c\{Q * F\}}\,Frame \qquad \frac{\vdash \{P\}c_1\{Q\} \quad \vdash \{Q\}c_2\{R\}}{\vdash \{P\}c_1; c_2\{R\}}\,Seq$$

$$\frac{\vdash \{\lambda(ps,n).\ I(ps,n) \wedge (b \hookrightarrow True)\ ps\}c\{I * \$1\}}{\vdash \{(\lambda(ps,n).\ I(ps,n) \wedge vars\ b \subseteq dom\ ps) * \$1\}\ \begin{array}{c}\text{WHILE } b \text{ DO } c \\ \{\lambda(ps,n).\ I(ps,n) \wedge (b \hookrightarrow False)\ ps\}\end{array}}\,While$$

$$\frac{\begin{array}{c}\forall ps\ n.\ P'(ps,n) \Longrightarrow P(ps,n) \\ \vdash \{P\}c\{Q\} \\ \forall ps\ n.\ Q(ps,n) \Longrightarrow Q'(ps,n)\end{array}}{\vdash \{P'\}c\{Q'\}}\,conseq$$

Fig. 4: Hoare logic with separation logic for reasoning about execution time

Proving soundness and completeness follows the same lines as for the quantitative Hoare logic, only complicated by the reasoning about partial states.

Theorem 9 (Soundness of $\vdash_3$). $\vdash_3 \{P\}c\{Q\} \Longrightarrow \models_3 \{P\}c\{Q\}$

This logic's weakest precondition is again defined as the right-hand side of the implication in the definition of validity:

$$wp\ c\ Q\ (ps,n) \equiv \exists ps'\ n'\ t.\ (c,s) \overset{t}{\Rightarrow}_p ps' \wedge n = n' + t \wedge\ Q\ (ps', n')$$

For completeness we first show $\vdash_3 \{wp\ c\ Q\}c\{Q\}$ by induction on the command c, and then use the definition of validity and wp to finish the proof.

Theorem 10 (Completeness of $\vdash_3$). $\models_3 \{P\}c\{Q\} \Longrightarrow \vdash_3 \{P\}c\{Q\}$

Big-O style Similar to last subsection's system we extend the Hoare logic based on Separation Logic to big-O style reasoning. We again generalize our

notion of validity (now $\models_{3'}$) and add a similar *const* rule to obtain the Hoare Logic $\vdash_{3'}$. Proving soundness and completeness of this new Hoare logic follows the same lines as in the subsection before. Similarly we come up with a simple VCG: somewhat unorthodoxly for separation logic, we use a backwards style, as well as we do not provide annotations for abstraction from multiplicative constants, as one final abstraction at the outer most position suffices to ensure completeness.

The approach inspired by Nielson to incorporate abstraction from multiplicative constants directly into the Hoare Logic in order to reason about the order of magnitude of the running time of programs shows weaknesses and seems to complicate matters. Our theoretical results show that it is always possible to reason about the exact running time and abstract away multiplicative constants in a last step.

4 Discussion

In this section we discuss the interrelations between the Hoare logics described in the last section.

First we can compare the expressibility of the logics. Nielson logic $\models_1$ and the quantitative Hoare logic $\models_{2'}$, both big-O style logics, are equivalent in the following sense:

Lemma 6. $\models_1 \{\lfloor P \rfloor_{\mathbb{B}}\} c \{\lambda s.\ \lfloor P\ s - Q(\downarrow_S (c, s))\rfloor_{\mathbb{N}} \Downarrow \lfloor Q \rfloor_{\mathbb{B}}\} \Longrightarrow \models_{2'} \{P\} c \{Q\}$ *where* $\lfloor P \rfloor_{\mathbb{B}}\ s \equiv P\ s < \infty$ *and* $\lfloor . \rfloor_{\mathbb{N}}$ *is the coercion from* $\mathbb{N}_\infty$ *to* $\mathbb{N}$*, assuming the argument is finite.*

Validity of a triple in the quantitative Hoare logic can be reduced to validity of a transformed triple in Nielson's logic. In the other direction this is only possible for assertions P and Q that do not depend on the state of their logical variables:

Lemma 7. $\models_{2'} \{\Uparrow P + e\} c \{\Uparrow Q\} \Longrightarrow \models_1 \{P\} c \{e \Downarrow Q\}$ *where* $\Uparrow P\ s \equiv (\forall l. \uparrow P\ l\ s)$

The quantitative logics support amortised resource analysis. On the face of it, Nielson's logic does not, but Lemma 6 tells us that in theory it actually does. However, automatic tools for resource analysis are mainly based on the potential method, for example [12, 5].

Furthermore, as the third system based on separation logic talks about partial states, in general it cannot be simulated by any of the other systems. This can only be done for assertions that act on complete states:

Lemma 8. $\models_{2'} \{\lfloor P \rfloor\} c \{\lfloor Q \rfloor\} \Longrightarrow \models_{3'} \{P\} c \{Q\}$*, when* P *is only true for complete partial states, with* $\lfloor P \rfloor s \equiv \inf_{n \in \mathbb{N}} \{P(\lfloor s \rfloor, n)\}$ *and* $\lfloor s \rfloor$ *is the partial state defined everywhere and returning the same results as the total state* s*.*

On the other hand any triple in the quantitative Hoare logic $\models_{2'}$ can be embedded into the separation logic $\models_{3'}$:

Lemma 9. $\models_{3'} \{\lfloor P \rfloor\} c \{\lfloor Q \rfloor\} \Longrightarrow \models_{2'} \{P\} c \{Q\}$*, where* $\lfloor P \rfloor (ps, n) \equiv (\forall s.\ n \geq P \lfloor ps \rfloor^s)$ *and* $\lfloor ps \rfloor^s$ *is the extension of the partial state* ps *by the state* s *to a total state.*

Example Let c be the IMP program that computes the discrete square root by bisection:

```
l ::= 0 ;; r::= x + 1;;  m ::= 0 ;;
(WHILE l + 1 < r DO
   m ::= (l + r) / 2;;
   (IF m * m < x THEN l ::= m ELSE r ::= m);;
   m ::= 0)
```

With the simplification that the intervals between l and r are always powers of two, we can easily show the running time to be in the order of magnitude of $1 + \log x$. Note that we can get rid of multiplicative constants, but not additive ones!

For showing $\vdash_1 \{\lambda l\ s.\ (\exists k. 1 + s\ ''x'' = 2^k)\} c \{\lambda s. \log(s\ ''x'') + 1 \Downarrow \lambda l\ s.\ True\}$ we provide the following annotations for the while loop: $I_1 = \lambda l\ s.\ s\ ''l'' \geq 0 \wedge (\exists k. s\ ''r'' - s\ ''l'' = 2^k)$, $E_1 = \lambda s.\ 1 + 5 \cdot \log(s\ ''r'' - s\ ''l'')$and $S_1 = \lambda s.\ s$; then we use our optimized VCG and prove the remaining proof obligations.

For showing $\vdash_{2'} \{(\lambda s.\ \uparrow(\exists k. 1 + s\ ''x'' = 2^k) + (\log(s\ ''x'') + 1)\} c \{\lambda\_. 0\})$, we annotate the while loop with the potential $I_{2'} = \lambda s.\ \uparrow(s\ ''l'' \geq 0 \wedge (\exists k. s\ ''r'' - s\ ''l'' = 2^k)) + 5 \cdot \log(s\ ''r'' - s\ ''l'')$.

Let us now compare the VCGs. Our VCG for Nielson's logic requires the annotation of loops with invariants I, running time bounds E and the state transformers S. In contrast, the annotations required for the VCG for the quantitative Hoare logic are uniformly potentials. In the above example, one can see that this annotated potential $I_{2'}$ exactly contains the same information as both I_1 and E_1 in the Nielson approach. The additional 1+ in E_1 is needed, as E_1 describes the running time of the whole loop, where $I_{2'}$ describes the running time from after evaluating the loop guard. Only more practical experience can tell if it is better to work with separate I, E and S or with a combined invariant potential.

In addition our annotated commands for Nielson's system may require annotations of the form $Conseq\ \{P', Q, e'\}$, whereas for the quantitative Hoare logic we managed to reduce this to $Const\ \{k\}$ annotations. It would be desirable to reduce the *Conseq* annotations similarly.

5 Related Work

Nielson [21, 22] was the first to study Hoare logics for running time analysis of programs. She proved soundness and completeness of her systems (on paper) which are based on a deep embedding of her assertion language. We base our formalization on the system given in [23] where assertions are just predicates, i.e. functions. However, our inference system differs from hers in several respects and our mechanized proofs in Isabelle/HOL are completely independent. Moreover we provide a VCG and prove it sound and complete.

Possibly the first example of a resource analysis logic based on potentials is due to Hofmann and Jost [11]. The idea of generalising predicates to potentials in

order to form a "quantitative Hoare logic" we borrowed from [4]: Carbonneaux *et al.* design a quantitative logic in order to reason about stack-space usage of C programs. They also formally show soundness of their logic in Coq. They employ their logic for reasoning about other resource bounds and use it as the underlying logic for an automatic tool for resource bound analysis [6, 5]. In a draft version of his dissertation [3] Carbonneaux complements his tool-focused work with a theoretical treatment of an "Invariant Logic". The relation to our logics of Section 3.2 should be studied in more detail.

Atkey [1] proposed to use separation logic with time credits to reason about the amortised running time of programs; he formalized his logic and its soundness in Coq. Similar ideas were used by Hoffmann *et al.* [10] to prove lock-freedom of concurrent programs, and by Charguéraud and Pottier [7] to verify the amortised running time of the Union-Find data structure in Coq. Guéneau, Charguéraud and Pottier [8] recently extended their framework to also obtain O results for the running time of programs. None of these works include verified VCGs.

There is also some related work that extends to probabilistic programs. Kaminski *et al.* [13] reason about the expected running time of probabilistic programs and show that their approach corresponds to Nielson's logic when restricted to deterministic algorithms. Ngo *et al.* [16] extend the idea of working with potentials to reasoning about the expected running time of probabilistic programs.

For formal treatment of program logics [17] is a good entry point. Basic concepts as well as formalizations of Hoare logics that lay the ground for our work can be found in [18].

6 Conclusion

In this paper we have studied three Hoare logics for reasoning about the running time of programs in a simple imperative language. We have formalized and verified their meta theory in Isabelle/HOL.

Further investigation is required in order to simplify the VCG for Nielson's logic and avoid the *Conseq* construct while preserving completeness of the VCG. Extending IMP with more language features is a natural next step. Adding recursive procedures should be easy (following [17]) whereas probabilistic choice (following [20]) is much more challenging and interesting. Not only is the meta theory of probabilistic programs nontrivial but even very small programs can be surprisingly hard to analyze. Although we view our work primarily as foundational, we expect that it could become a viable basis for the verification of small probabilistic programs.

Data Availability Statement and Acknowledgments. The formal proof development is available online [9]. We thank Peter Lammich for his initial help with setting up the separation logic.

References

1. Atkey, R.: Amortised resource analysis with separation logic. In: ESOP. vol. 6012, pp. 85–103. Springer (2010)
2. Calcagno, C., O'Hearn, P.W., Yang, H.: Local action and abstract separation logic. In: Logic in Computer Science, LICS 2007. pp. 366–378. IEEE (2007)
3. Carbonneaux, Q.: Modular and Certified Resource-Bound Analyses. PhD dissertation, Yale University (2017), forthcoming, draft at `http://cs.yale.edu/homes/qcar/diss/`
4. Carbonneaux, Q., Hoffmann, J., Ramananandro, T., Shao, Z.: End-to-end verification of stack-space bounds for C programs. In: O'Boyle, M.F.P., Pingali, K. (eds.) Conference on Programming Language Design and Implementation, PLDI, 2014. pp. 270–281. ACM (2014)
5. Carbonneaux, Q., Hoffmann, J., Reps, T., Shao, Z.: Automated resource analysis with Coq proof objects. In: International Conference on Computer Aided Verification, CAV, 2017. pp. 64–85. Springer (2017)
6. Carbonneaux, Q., Hoffmann, J., Shao, Z.: Compositional certified resource bounds. In: Grove, D., Blackburn, S. (eds.) Conference on Programming Language Design and Implementation, PLDI, 2015. pp. 467–478. ACM (2015)
7. Charguéraud, A., Pottier, F.: Verifying the correctness and amortized complexity of a union-find implementation in separation logic with time credits. Journal of Automated Reasoning pp. 1–35 (2017)
8. Guéneau, A., Charguéraud, A., Pottier, F.: A fistful of dollars: Formalizing asymptotic complexity claims via deductive program verification. In: European Symposium on Programming (ESOP) (2018)
9. Haslbeck, M.P.L., Nipkow, T.: Hoare logics for time bounds. Archive of Formal Proofs (Feb 2018), `https://www.isa-afp.org/entries/Hoare_Time.html`, Formal proof development
10. Hoffmann, J., Marmar, M., Shao, Z.: Quantitative reasoning for proving lock-freedom. In: Logic in Computer Science, LICS, 2013. pp. 124–133. IEEE (2013)
11. Hofmann, M., Jost, S.: Type-based amortised heap-space analysis. In: Sestoft, P. (ed.) Programming Languages and Systems, ESOP 2006. Lecture Notes in Computer Science, vol. 3924, pp. 22–37. Springer (2006)
12. Hofmann, M., Rodriguez, D.: Automatic type inference for amortised heap-space analysis. In: Felleisen, M., Gardner, P. (eds.) Programming Languages and Systems, ESOP 2013. Lecture Notes in Computer Science, vol. 7792, pp. 593–613. Springer (2013)
13. Kaminski, B.L., Katoen, J.P., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected run–times of probabilistic programs. In: European Symposium on Programming Languages and Systems. pp. 364–389. Springer (2016)
14. Klein, G., Kolanski, R., Boyton, A.: Separation algebra. Archive of Formal Proofs (May 2012), `http://isa-afp.org/entries/Separation_Algebra.html`, Formal proof development
15. Kleymann, T.: Hoare logic and auxiliary variables. Formal Aspects of Computing 11(5), 541–566 (1999)
16. Ngo, V.C., Carbonneaux, Q., Hoffmann, J.: Bounded expectations: Resource analysis for probabilistic programs. In: Conference on Programming Language Design and Implementation, PLDI, 2018 (2018)
17. Nipkow, T.: Hoare logics in Isabelle/HOL. In: Schwichtenberg, H., Steinbrüggen, R. (eds.) Proof and System-Reliability. pp. 341–367. Kluwer (2002)

18. Nipkow, T., Klein, G.: Concrete Semantics: With Isabelle/HOL. Springer (2014)
19. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
20. Olmedo, F., Kaminski, B.L., Katoen, J.P., Matheja, C.: Reasoning about recursive probabilistic programs. In: Logic in Computer Science. pp. 672–681. LICS '16, ACM (2016)
21. Riis Nielson, H.: Hoare logic's for run-time analysis of programs. Ph.D. thesis, University of Edinburgh (1984)
22. Riis Nielson, H.: A Hoare-like proof system for analysing the computation time of programs. Science of Computer Programming 9(2), 107–136 (1987)
23. Riis Nielson, H., Nielson, F.: Semantics with applications: an appetizer. Springer (2007)