Hoare Logic in the Abstract

Ursula Martin, Erik A. Mathiesen, and Paulo Oliva

Department of Computer Science
Queen Mary, University of London
Mile End Road
London E1 4NS, UK
{uhmm, erikarne, pbo}@dcs.qmul.ac.uk

Abstract. We present an abstraction of Hoare logic to traced symmetric monoidal categories, a very general framework for the theory of systems. We first identify a particular class of functors – which we call 'verification functors' – between traced symmetric monoidal categories and subcategories of Preord (the category of preordered sets and monotone mappings). We then give an abstract definition of Hoare triples, parametrised by a verification functor, and prove a single soundness and completeness theorem for such triples. In the particular case of the traced symmetric monoidal category of while programs we get back Hoare's original rules. We discuss how our framework handles extensions of the Hoare logic for while programs, e.g. the extension with pointer manipulations via separation logic. Finally, we give an example of how our theory can be used in the development of new Hoare logics: we present a new sound and complete set of Hoare-logic-like rules for the verification of linear dynamical systems, modelled via stream circuits.

1 Introduction

A few years ago one of the authors spent some time with an aeronautical engineer learning how flight control systems are designed. Standard engineering tools like Simulink represent the differential equations that model the system as a box-and-arrow diagram, incorporating composition and feedback. The engineer's informal explanation of how an input signal is transformed to an output signal as it passes through each component of the diagram was strikingly similar to a Hoare logic presentation of a program proof, and inspired the development of a sound, if ugly, ad-hoc Hoare-like logic for a fragment of linear differential equations [7].

So obvious questions (for us, if not the engineer) are how it is that such disparate things as programs and differential equations can both support Hoare-like logics, and whether there is a principled way of developing such logics in new situations. This paper provides the answer: both are instances of a particular kind of traced symmetric monoidal category, and for any such instance a sound and complete set of Hoare-logic-like rules can be read off from the categorical structure.

Under the general label of *Hoare logic*, the early work of Floyd [10] and Hoare [12] on axiom systems for flowcharts and while programs has been applied to various other domains, such as recursive procedures [2], pointer programs [19], and

higher-order languages [4]. Our goal in this paper is to identify a minimal structure supporting soundness and (relative) completeness results, in the manner of Cook's presentation for *while programs* [8]. This is achieved via an abstraction of Hoare logic to the theory of traced symmetric monoidal categories [13], a very general framework for the theory of systems. Traced symmetric monoidal categories precisely capture the intrinsic structure of both flowcharts and dynamical systems, namely: sequential and parallel 'composability', and infinite behaviour. In fact, traced symmetric monoidal categories are closely related to Bainbridge's work [3] on the duality between flowcharts and networks. The scope of traced symmetric monoidal categories, however, is much broader, being actively used, for instance, for modelling computation (e.g. [21]) and in connection with Girard's geometry of interaction (e.g. [11]).

The main feature of Hoare logic is the use of pre- and post-conditions to specify the behaviour of a program, and the order relation between pre- and post-condition given by logical implication. Let Preord be the category of preordered sets and monotone mappings. We first identify a particular class of functors – which we call *verification functors* – between traced symmetric monoidal categories and subcategories of Preord. We then give an abstract definition of Hoare triples, parametrised by a verification functor, and prove a single soundness and completeness (in the sense of Cook) theorem for such triples. In the particular case of the traced symmetric monoidal category of while programs (respectively, pointer programs) this embedding gives us back Hoare's original logic [12] (respectively, O'Hearn and Reynolds logic [19]). In order to illustrate the generality of our framework, we also derive new sound and complete Hoare-logic-like rules for the verification of linear dynamical systems, in our case, modelled via stream circuits.

In general, our abstraction of Hoare logic provides a 'categorical' recipe for the development of new (automatically) sound and complete Hoare-logic-like rules for any class of systems having the underlying structure of a traced symmetric monoidal category. Moreover, Hoare logic notions such as expressiveness conditions, relative completeness [8] and loop invariants, have a clear cut correspondence to some of our abstract notions.

The chief contributions of this paper are as follows: (i) The definition of a verification functor, between traced symmetric monoidal categories and the category Preord (Section 3). (ii) An abstraction of Hoare triples in terms of verification functors (Definition 3). (iii) Sound and complete rules for our abstract notion of Hoare triples, over a fixed verification functor (Theorem 1). (iv) Three concrete instances of our abstraction, namely: while programs, pointer programs, and stream circuits (Section 4). In Section 5, we discuss the link between our work and other abstractions of Hoare logic.

For the rest of this article we will assume some basic knowledge of category theory. For a readable introduction see [15]. Given a category $\mathcal{S}$, we will denote its objects by $\mathcal{S}_o$ and its morphisms by $\mathcal{S}_m$. Composition between two morphisms will be denoted as usual by $(g \circ f) : X \to Z$, if $f : X \to Y$ and $g : Y \to Z$.

Fig. 1. Trace diagrammatically

2 Traced Symmetric Monoidal Categories

A pair of a category $\mathcal{S}$ and a functor $\otimes$ is called a *monoidal category* if for some particular object of $\mathcal{S}$ (the identity element of the monoid) $\otimes$ satisfies the monoidal axioms of associativity and identity (for details, see Chapter 11 of [15]). A monoidal category is called symmetric if there exists a family of natural isomorphisms $c_{X,Y} : X \otimes Y \to Y \otimes X$ satisfying the two braiding axioms plus the symmetry axiom $c_{X,Y} \circ c_{Y,X} = \mathsf{id}_{X \otimes Y}$.

In [13], the notion of *traced symmetric monoidal category* is introduced[1], for short *tmc*. A symmetric monoidal category is traced if for any morphism $f : X \otimes Z \to Y \otimes Z$ there exists a morphism $\mathsf{Tr}^Z_{X,Y}(f) : X \to Y$ satisfying the trace axioms (see [13] for details). We will normally omit the decoration in $\mathsf{Tr}^Z_{X,Y}$ whenever it is clear over which object the trace is being applied. Morphisms of a tmc can be represented diagrammatically as input-output boxes, so that, if $f : X \otimes Z \to Y \otimes Z$ then $\mathsf{Tr}(f) : X \to Y$ corresponds to a *feedback* over the 'wire' Z, as shown in Figure 1.

The two most common examples of traced symmetric monoidal categories are the category of sets and relations where $\otimes$ is either disjoint union, with trace defined as

$$x \,\mathsf{Tr}(f)\, y \;:\equiv\; \exists \boldsymbol{z}(\langle 0, x\rangle f \langle 1, z_0\rangle \wedge \ldots \langle 1, z_i\rangle f \langle 1, z_{i+1}\rangle \ldots \wedge \langle 1, z_n\rangle f \langle 0, y\rangle)$$

or cartesian product, with trace defined as

$$x \,\mathsf{Tr}(f)\, y \;:\equiv\; \exists z(\langle x, z\rangle f \langle y, z\rangle)$$

Note that we abbreviate a tuple of variables $z_0, \ldots, z_n \in Z$ by $\boldsymbol{z}$.

Definition 1 (Basic morphisms). *Let $M \subseteq \mathcal{S}_m$ be a subset of the morphisms of a traced symmetric monoidal category $\mathcal{S}$. We define $\mathsf{cl}(M)$, the* traced symmetric monoidal closure of M*, as the smallest subset of $\mathcal{S}_m$ containing M which is closed under sequential composition, monoidal closure, and trace. If M is such that $\mathsf{cl}(M) = \mathcal{S}_m$ then M is called a set of* basic morphisms *for the category $\mathcal{S}$.*

[1] In fact, [13] introduces the theory of traces for a more general class of monoidal categories, so-called balanced monoidal categories, of which symmetric monoidal categories are a special case.

Intuitively, we view each morphism in a tmc as representing a particular system. The categorical composition models the sequential composition of systems, while the monoidal operation is related to the 'parallel' composition of two systems. Finally, the trace corresponds to feedback, allowing infinite behaviour. The basic morphisms represent the basic systems, out of which complex systems are built. Note that the whole set of morphisms is trivially a set of basic morphisms. We are interested, however, in tmc in which the set of basic morphisms (systems) forms a strict subset of the set of all morphisms, as in the following three examples.

The categories we describe below are purely *semantical*. Nevertheless, we build the categories in such a way that each *syntactic* while program or stream circuit is denoted by a morphism in the category, and *vice versa* each morphism has a representation as a syntactic while program or stream circuit.

2.1 Example 1: Flowcharts

The first example we consider is that of *flowcharts*. Let Store be the set of stores, i.e. mappings $\rho : \mathsf{Var} \to \mathbb{Z}$ from program variables $\mathsf{Var} = \{x, y, \ldots\}$ to integers. Let $\mathcal{F}_o$ be the set (of sets) containing the empty set $\emptyset$ and Store, and closed under *disjoint union*, i.e. $\mathcal{F}_o \equiv \{\emptyset, \mathsf{Store}, \mathsf{Store} \uplus \mathsf{Store}, \ldots\}$. Consider also the following family of functions $\mathcal{F}_b$ between sets in $\mathcal{F}_o$:

- *Skip*, $\mathsf{id} : \mathsf{Store} \to \mathsf{Store}$
  $\mathsf{id}(\rho) :\equiv \rho$
- *Assignment*, $(x := t) : \mathsf{Store} \to \mathsf{Store}$
  $(x := t)(\rho) :\equiv (\rho)[t_\rho/x]$
- *Joining*, $\Delta : \mathsf{Store} \uplus \mathsf{Store} \to \mathsf{Store}$
  $\Delta(\rho) :\equiv \mathsf{proj}(\rho)$
- *Forking*, $\nabla_b : \mathsf{Store} \to \mathsf{Store} \uplus \mathsf{Store}$
  $$\nabla_b(\rho) :\equiv \begin{cases} \mathsf{inj}_0(\rho) & \text{if } \neg b_\rho \\ \mathsf{inj}_1(\rho) & \text{otherwise} \end{cases}$$

The conditional forking (∇_b) and the assignment ($x := t$) are parametrised by functions b and t (intuitively, expressions) from Store to the boolean lattice $\mathbb{B}$ and $\mathbb{Z}$, respectively, so that b_ρ and t_ρ denote their value on a given store ρ. We use inj_0 and inj_1 for left and right injections into $\mathsf{Store} \uplus \mathsf{Store}$, while proj is the projection. We then close the set of basic functions $\mathcal{F}_b$ under sequential composition of functions, disjoint union of functions and the standard trace for disjoint union, to form the set of partial functions $\mathcal{F}_m$. It is easy to see that $\mathcal{F} \equiv (\mathcal{F}_o, \mathcal{F}_m, \uplus, \mathsf{Tr})$ forms a tmc, if we add to the set of basic morphisms a family of functions $c_{X,Y} : X \uplus Y \to Y \uplus X$, which simply flip the flag of the disjoint union. Note that $\mathcal{F}_b$ is by construction a set of basic morphisms for $\mathcal{F}$.

2.2 Example 2: Pointer programs

The second example of tmc we consider is an extension of $\mathcal{F}$ to a category of programs that manipulate both stores and heaps, which we will refer to as the traced symmetric monoidal category of *pointer programs* $\mathcal{P}$. Let $\mathsf{State} :\equiv (\mathsf{Store} \times \mathsf{Heap}) \cup \{\mathsf{abort}\}$, where Store is as in the previous section and Heap is the set of partial functions (from $\mathbb{N}$ to $\mathbb{Z}$) with finite domain. We view the

elements of the heap as pairs consisting of a function $h : \mathbb{N} \to \mathbb{Z}$ and a finite set $d \in \mathcal{P}_{\mathsf{fin}}(\mathbb{N})$ describing the valid domain of h. We then form the set (of sets) $\mathcal{P}_o$ as the set containing the empty set $\emptyset$ and State, and closed under *disjoint union*, i.e. $\mathcal{P}_o \equiv \{\emptyset, \mathsf{State}, \mathsf{State} \uplus \mathsf{State}, \ldots\}$. Each of the basic functions of $\mathcal{F}_b$ can be lifted to the extended type structure, by simply ignoring the 'heap' component. The set of functions $\mathcal{P}_b$ is an extension of the set of (the lifting of the) functions $\mathcal{F}_b$ with the following family of functions:

- *Look up*, $(x := [t]) : \mathsf{State} \to \mathsf{State}$
  $$(x := [t])(\rho, h, d) :\equiv \begin{cases} (\rho[h(t_\rho)/x], h, d) & t_\rho \in d \\ \mathsf{abort} & \mathsf{otherwise} \end{cases}$$
- *Mutation*, $([t] := s) : \mathsf{State} \to \mathsf{State}$
  $$([t] := s)(\rho, h, d) :\equiv \begin{cases} (\rho, h[t_\rho \mapsto s_\rho], d) & t_\rho \in d \\ \mathsf{abort} & \mathsf{otherwise} \end{cases}$$
- *Allocation*, $x := \mathsf{new}(\boldsymbol{t}) : \mathsf{State} \to \mathsf{State}$
  $$(x := \mathsf{new}(\boldsymbol{t}))(\rho, h, d) :\equiv (\rho[x \mapsto i], h[i + j \mapsto t_j], d \uplus \{i, \ldots, i + n\})$$
- *Deallocation*, $\mathsf{disp}(t) : \mathsf{State} \to \mathsf{State}$
  $$(\mathsf{disp}(t))(\rho, h, d) :\equiv \begin{cases} (\rho, h, d \backslash \{t_\rho\}) & t_\rho \in d \\ \mathsf{abort} & \mathsf{otherwise} \end{cases}$$

As done in the case of flowcharts above, we can then close the set of functions $\mathcal{P}_b$ under sequential composition, disjoint union and trace, to form the set of partial functions $\mathcal{P}_m$. We are assuming that the functions are strict with respect to abort, i.e. on the abort state all programs will return abort. The category $\mathcal{P} \equiv (\mathcal{P}_o, \mathcal{P}_m, \uplus, \mathsf{Tr})$, with the standard trace for disjoint union, forms another example of a tmc with the extra basic morphisms look up, mutation, allocation and deallocation.

2.3 Example 3: Stream circuits

Finally, we give an example of a tmc (the category of *stream circuits*) based on cartesian product, rather than disjoint union. Let Σ denote the set of streams[2] of real numbers, i.e. all functions $\mathbb{N} \to \mathbb{R}$. Let $\mathcal{C}_o$ be the set containing the singleton set $\{\varepsilon\}$ and Σ, and closed under cartesian product, i.e. $\mathcal{C}_o \equiv \{\{\varepsilon\}, \Sigma, \Sigma \times \Sigma, \ldots\}$. Consider the following set $\mathcal{C}_b$ of functions between the sets in $\mathcal{C}_o$:

- *Wire*, $\mathsf{id} : \Sigma \to \Sigma$
  $\mathsf{id}(\sigma) :\equiv \sigma$
- *Copy*, $\mathsf{c} : \Sigma \to \Sigma \times \Sigma$
  $\mathsf{c}(\sigma) :\equiv \langle \sigma, \sigma \rangle$
- *Scalars*, $(a\times) : \Sigma \to \Sigma$
  $(a\times)(\sigma) :\equiv [a\sigma_0, a\sigma_1, \ldots]$
- *Sum*, $(+) : \Sigma \times \Sigma \to \Sigma$
  $(+)\langle \sigma, \sigma' \rangle :\equiv [\sigma_0 + \sigma'_0, \sigma_1 + \sigma'_1, \ldots]$
- *Register*, $R : \Sigma \to \Sigma$
  $R(\sigma) :\equiv [0, \sigma_0, \sigma_1, \ldots]$

[2] As shown in [9], much of mathematical analysis can be developed co-inductively using streams: the stream corresponding to an analytic function f is just $[f(0), f'(0), f''(0), \ldots]$ which is related to the coefficients of the Taylor series expansion of f.

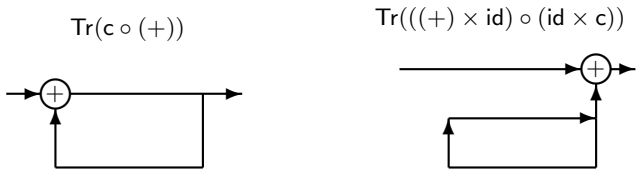

Fig. 2. Stream circuits not defining functions

We then form the set of *relations* $\mathcal{C}_m$ by viewing the basic functions $\mathcal{C}_b$ above as relations (via their graph) and closing this set under relational composition, cartesian product of relations, and the standard trace for the cartesian product (cf. Section 2). The diagrams in Figure 2 illustrate two stream circuits which do not define a function (but define a relation). In the circuit on the left there is no fixed point for the input stream $\sigma_a :\equiv [a, a, \ldots]$, for $a \neq 0$. On the other hand, in the circuit on the right any stream τ is a fixed point of the trace, so that any input stream σ is related to an arbitrary (output) stream. In order to avoid such pathological circuits (i.e. circuits not defining a function), one normally puts a further restriction on the formation of stream circuits. A stream circuit $f : X \to Y$ is called *valid* if in every sub-circuit of the form $\mathsf{Tr}^Z_{X',Y'}(f') : X' \to Y'$ the path from X' to the output Z goes through a register. This requirement for forming loops is standard in stream circuits (cf. [20]). Although $\mathcal{C}$ is a category of relations, valid circuits are denoted by *total* functions, since the fixed points of a valid circuit exist and are unique.

It is again easy to see that $\mathcal{C} \equiv (\mathcal{C}_o, \mathcal{C}_m)$ forms a category. The category $\mathcal{C}$ is symmetric monoidal, with the monoidal structure of cartesian product, if we add to the set of basic morphisms the family of relations $c_{X,Y} : X \times Y \to Y \times X$, which simply flip the two components of the input pair, i.e. $\langle x, y\rangle c_{X,Y} \langle y, x\rangle$. Finally, with the standard family of trace relations (defined in Section 2), $\mathcal{C}$ forms a traced symmetric monoidal category.

3 Abstract Hoare Logic

In this section we present an abstraction of Hoare logic, including pre- and post-conditions and Hoare triples as normally used in Hoare logic for while programs. Our goal here is to isolate the minimal structure needed from these pre- and post-conditions, for the development of a sound and complete Hoare logic.

Recall that a mapping $f : X \to Y$ between two preordered sets X, Y is called *monotone* if $P \sqsubseteq_X Q$ implies $f(P) \sqsubseteq_Y f(Q)$. Let Preord denote the category of preordered sets and monotone mappings. It is easy to see that Preord can be considered a symmetric monoidal category, with the monoidal operation as cartesian product, since the cartesian product of two preordered sets X, Y forms again a preordered set with the order on $X \times Y$ defined coordinatewise.

For the rest of this section, let $\mathcal{S}$ be a fixed tmc. A subset $\mathcal{S}'_m$ of the morphisms $\mathcal{S}$ is called *subsystem closed* if $f, g \in \mathcal{S}'_m$, whenever $\mathsf{Tr}(f)$, $f \circ g$ or $f \otimes g$ is in $\mathcal{S}'_m$.

Definition 2 (Verification functor). *Let $\mathcal{S}'_m$ be a subsystem closed subset of $\mathcal{S}_m$. A strict monoidal functor $H : \mathcal{S} \to \mathsf{Preord}$ is called a* verification functor *for $\mathcal{S}'_m$ if*

$$H(\mathsf{Tr}(f))(P) \sqsubseteq R \;\;\Leftrightarrow\;\; \exists Q \in H(Z)(H(f)\langle P, Q\rangle \sqsubseteq \langle R, Q\rangle) \tag{1}$$

for all $f : X \otimes Z \to Y \otimes Z$ such that $\mathsf{Tr}(f) \in \mathcal{S}'_m$, and $P \in H(X)$, $R \in H(Y)$. If $\mathcal{S}'_m = \mathcal{S}_m$ we simply call H a verification functor.

Let a verification functor $H : \mathcal{S} \to \mathsf{Preord}$ for $\mathcal{S}'_m$ be fixed. For simplicity assume $\mathcal{S}'_m = \mathcal{S}_m$ (see Remark 1). Each object $X \in \mathcal{S}_o$ corresponds to a preordered set $H(X)$, and each morphism $f \in \mathcal{S}_m$ corresponds to a monotone mapping $H(f)$.

Definition 3 (Abstract Hoare Triples). *Let $f : X \to Y$ be a morphism (system) in $\mathcal{S}$, $P \in H(X)$ and $Q \in H(Y)$. Define* abstract Hoare triples *as*

$$\{P\}\ f\ \{Q\} \;\;:\equiv\;\; H(f)(P) \sqsubseteq_{H(Y)} Q \tag{2}$$

Although we use the same notation as the standard Hoare triple, it should be noted that the meaning of our abstract Hoare triple can only be given once the verification functor H is fixed. The usual Hoare logic meaning of *if P holds before the execution of f then, if f terminates, Q holds afterwards* will be one of the special cases of our general theory. See Section 4 for other meanings of $\{P\}\ f\ \{Q\}$.

Let a tmc $\mathcal{S}$ and verification functor $H : \mathcal{S} \to \mathsf{Preord}$ be fixed. Moreover, let $\mathcal{S}_b$ be a set of basic morphisms for $\mathcal{S}$. We will denote by $\mathbf{H}(\mathcal{S}, H)$ the following set of rules:

$$\frac{\{P\}\ f\ \{Q\} \quad \{Q\}\ g\ \{R\}}{\{P\}\ g \circ f\ \{R\}}\ (\circ) \qquad \frac{}{\{P\}\ f\ \{H(f)(P)\}}\ (f \in \mathcal{S}_b)$$

$$\frac{\{P\}\ f\ \{Q\} \quad \{R\}\ g\ \{S\}}{\{\langle P, R\rangle\}\ f \otimes g\ \{\langle Q, S\rangle\}}\ (\otimes) \qquad \frac{\{P\}\ f\ \{Q\}}{\{P'\}\ f\ \{Q'\}}\ \begin{pmatrix} P' \sqsubseteq P \\ Q \sqsubseteq Q' \end{pmatrix}$$

$$\frac{\{\langle P, Q\rangle\}\ f\ \{\langle R, Q\rangle\}}{\{P\}\ \mathsf{Tr}_{\mathcal{S}}(f)\ \{R\}}\ (\mathsf{Tr})$$

The formal system $\mathbf{H}(\mathcal{S}, H)$ should be viewed as a *syntactic* axiomatisation of the ternary relation $\{P\}\ f\ \{Q\}$. The verification functor H gives the *semantics* of the Hoare triples (and rules). By soundness and completeness of the system $\mathbf{H}(\mathcal{S}, H)$ we mean that syntax corresponds precisely to semantics, i.e. a syntactic Hoare triple $\{P\}\ f\ \{Q\}$ is provable in $\mathbf{H}(\mathcal{S}, H)$ if and only if the inequality $H(f)(P) \sqsubseteq Q$ is true in Preord.

Theorem 1 (Soundness and completeness). *The system* $\mathbf{H}(\mathcal{S}, H)$ *is sound and complete.*

Proof. Soundness is trivially true for the axioms. The consequence rule is sound by the monotonicity of $H(f)$ and transitivity of $\sqsubseteq$. Soundness of the composition rule also uses the monotonicity of $H(g)$, the transitivity of $\sqsubseteq$ and the fact that H respects composition, i.e. $H(g)(H(f)(P)) = H(g \circ f)(P)$. Soundness of the cartesian product rule uses that the functor H is monoidal, i.e. $H(f \otimes g)\langle P, Q\rangle = (H(f) \times H(g))\langle P, Q\rangle = \langle H(f)(P), H(g)(Q)\rangle$. For the soundness of the trace rule, assume $H(f)\langle P, Q\rangle \sqsubseteq \langle R, Q\rangle$. By the definition of a verification functor we have $H(\mathsf{Tr}(f))(P) \sqsubseteq R$. We argue now about completeness. By the consequence rule and the axioms it is easy to verify that all true statements of the form $\{P\}\ f\ \{Q\}$, for $f \in \mathcal{S}_b$, are provable. It remains to show that if $\{P\}\ f\ \{Q\}$ is true, for an arbitrary morphism f, then there exists a premise of the corresponding rule (depending on the structure of f) which is also true. If $\{P\}\ g \circ f\ \{R\}$ is true then so it is $\{P\}\ f\ \{H(f)(P)\}$, by the reflexivity of the ordering, and $\{H(f)(P)\}\ g\ \{R\}$, by the fact that H respects composition. If $\{\langle P, R\rangle\}\ f \otimes g\ \{\langle Q, S\rangle\}$ is true then so it is $\{P\}\ f\ \{Q\}$ and $\{R\}\ g\ \{S\}$, by the fact that H is monoidal. Finally, if $\{P\}\ \mathsf{Tr}(f)\ \{R\}$ is true, then so is $\{\langle P, Q\rangle\}\ f\ \{\langle R, Q\rangle\}$, for some Q, by the definition of a verification functor. □

The abstract proof of soundness and completeness presented above is both short and simple, using only the assumption of a verification functor and basic properties of the category Preord. As we will see, the laborious work is pushed into showing that H is a verification functor. That will be clear in Section 4.1, where we build a verification functor appropriate for while programs, using Cook's expressiveness condition [8].

Remark 1. It is easy to see that all arguments in the proof of Theorem 1 will go through if we restrict ourselves to a particular subsystem closed set of morphisms $\mathcal{S}'_m$. This follows from the fact that in the completeness proof we simply use that any morphism can be 'generated' from the basic morphisms, which is also true for morphisms in $\mathcal{S}'_m$. If H is a verification functor for $\mathcal{S}'_m \subset \mathcal{S}_m$, the rules of $\mathbf{H}(\mathcal{S}, H)$ should be restricted to morphisms in $\mathcal{S}'_m$. Similarly, the soundness and completeness theorems apply only to systems in $\mathcal{S}'_m$. See Section 4.3 for an instantiation of the general framework where a special class of systems $\mathcal{S}'_m$ is singled out.

4 Instantiations

We will now look at some concrete examples of verification functors for the traced symmetric monoidal categories described in Section 2. For each of the instantiations of tmc $\mathcal{S}$ we have described we present a monoidal functor $H : \mathcal{S} \to \mathsf{Preord}$ and show that it satisfies condition (1).

First we will consider the traced symmetric monoidal categories of flowcharts and pointer programs. These instantiations will produce, respectively, the original Hoare logic [12], and Reynold's [19] axiomatisation based on separation logic.

We then present a new variation of Hoare logic for stream circuits, obtained through our abstraction.

4.1 Flowcharts (forward reasoning)

In this section we present a verification embedding of the tmc of flowcharts $\mathcal{F}$ (see Section 2.1). This will give us soundness and (relative) completeness of Hoare's original verification logic [12] for partial correctness (using forward reasoning).

Let us now define the monoidal functor $H : \mathcal{F} \to \mathsf{Preord}$. Let a Cook-expressive[3] first-order theory $\mathcal{L}$ be fixed. On the objects $X \in \mathcal{F}_o$ we let $H(X)$ be the preordered set of formulas of $\mathcal{L}$. The ordering $P \sqsubseteq R$ on the elements of $H(X)$ is taken to be $\mathcal{L} \vdash P \to R$. On the morphisms (flowcharts) $f \in \mathcal{F}_m$ we let $H(f)$ be the predicate transformer producing strongest post-conditions for any pre-condition P, i.e.

$$H(f)(P) :\equiv \mathsf{SPC}(f, P)$$

where $\mathsf{SPC}(f, P)$ is a formula expressing the strongest post-condition of f under P. Such formula exists by our assumption that the theory $\mathcal{L}$ is Cook-expressive. It is also easy to see what those are for the basic morphisms of $\mathcal{F}$

$$\begin{aligned}
\mathsf{SPC}(\mathsf{id}, P) &:\equiv P \\
\mathsf{SPC}(x := t, P) &:\equiv \exists v(P[v/x] \wedge x = t[v/x]) \\
\mathsf{SPC}(\nabla_b, P) &:\equiv \langle P \wedge \neg b, P \wedge b \rangle \\
\mathsf{SPC}(\Delta, \langle P, R \rangle) &:\equiv P \vee R
\end{aligned}$$

The functor H is monoidal because a formula P describing a subset of $X_0 \uplus X_1$ can be seen as a pair of formulas $\langle P_0, P_1 \rangle$ such that each P_i describes a subset of X_i, i.e. $H(X \uplus Y)$ is isomorphic to $H(X) \times H(Y)$. Similarly, there is a one-to-one correspondence between strongest post-condition transformer for a parallel composition of flowcharts $f \uplus g$ and pairs of predicate transformers $H(f) \times H(g)$. We argue now in two steps that H is also a verification functor.

Lemma 1 ([8]). *Let $f : X \uplus Z \to Y \uplus Z$ in $\mathcal{F}$ and formulas $P \in H(X)$ and $R \in H(Y)$ be fixed. If $\mathsf{SPC}(\mathsf{Tr}(f), P) \to R$ then $\mathsf{SPC}(f, \langle P, Q \rangle) \to \langle R, Q \rangle$, for some formula Q.*

Proof. We construct the formula Q such that it is a fixed point for f on P, i.e. $\mathsf{SPC}(f, \langle P, Q \rangle) \leftrightarrow \langle R', Q \rangle$, for some formula R'. We also argue that $\mathsf{SPC}(\mathsf{Tr}(f), P) \leftrightarrow R'$. By our hypothesis $\mathsf{SPC}(\mathsf{Tr}(f), P) \to R$ it follows that R' implies R, as desired. The fixed point Q is essentially the strongest loop invariant, and is constructed as follows. Given the partial function f we build a new partial function $f' : X \uplus Z \to Y \uplus Z \uplus Z$ where the internal states of Z can be observed, even after the trace is applied. Let $f' :\equiv (\mathsf{id} \uplus \nabla_{\boldsymbol{z}=\boldsymbol{y}}) \circ f$ (see Figure 3) where $\boldsymbol{z}$ is

[3] Recall that a logic is Cook-expressive if for any program f and pre-condition P the strongest post-condition of f under P is expressible by a formula in $\mathcal{L}$ (cf. [8]).

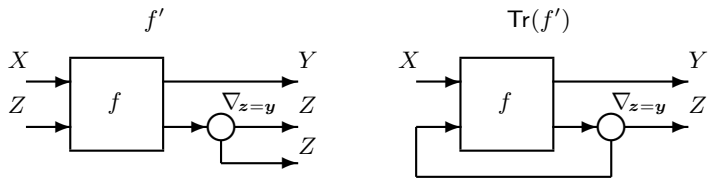

Fig. 3. Cook's construction

the finite sequence of variables mentioned in f (by construction of the category $\mathcal{F}$, each morphism only changes finitely many variables), and $\boldsymbol{y}$ is a fresh tuple of variables of same length. Notice that $\mathsf{Tr}^{Z}_{X,Y\uplus Z}(f')$ behaves almost as $\mathsf{Tr}^{Z}_{X,Y}(f)$ except that $\mathsf{Tr}(f')$ might 'terminate' earlier (meaning that the fixed point sequence is shorter) if the state $\boldsymbol{z}$ matches $\boldsymbol{y}$. Let $\mathsf{SPC}(\mathsf{Tr}(f'), P) = \langle Q_0, Q_1(\boldsymbol{y})\rangle$. It is easy to see that the formula $Q \equiv \exists \boldsymbol{y}\, Q_1(\boldsymbol{y})$ characterises the possible internal states $\boldsymbol{z}$ in the trace of f on an input satisfying P, i.e. Q is a fixed point for $H(f)$ on P, $\mathsf{SPC}(f, \langle P, Q\rangle) \leftrightarrow \langle R', Q\rangle$, for some formula R'. It also follows that $\mathsf{SPC}(\mathsf{Tr}(f), P) \leftrightarrow R'$, since Q characterises precisely the internal states of the trace $\mathsf{Tr}(f)$ on inputs satisfying P. □

Theorem 2. *$H : \mathcal{F} \to \mathsf{Preord}$, as defined above, is a verification functor.*

Proof. By Lemma 1, it remains to be shown that whenever

(i) $\mathsf{SPC}(f, \langle P, Q'\rangle) \to \langle R, Q'\rangle$,

for some formula Q', then $\mathsf{SPC}(\mathsf{Tr}(f), P) \to R$. Assume (i) and $\mathsf{SPC}(\mathsf{Tr}(f), P)(\rho)$, for some store value ρ. We must show $R(\rho)$. By the definition of the strongest post-condition there exists a sequence of stores $\rho', \rho_0, \ldots, \rho_n$ such that $P(\rho')$ and

$$f\langle 0, \rho'\rangle = \langle 1, \rho_0\rangle, \ldots, f\langle 1, \rho_k\rangle = \langle 1, \rho_{k+1}\rangle, \ldots, f\langle 1, \rho_n\rangle = \langle 0, \rho\rangle.$$

By a simple induction, using the assumption (i), we get that all ρ_k satisfy Q' and that ρ satisfies R, as desired. □

The system $\mathbf{H}(\mathcal{F}, H)$ obtained via our embedding H is a refinement of the system given by Hoare in [12], i.e. Hoare's rules are derivable from ours. See, for instance, the case of the while loop rule in Figure 4, given that a while loop $\mathsf{while}_b(f)$ can be represented in $\mathcal{F}$ as $\mathsf{Tr}((\mathsf{id}\uplus f)\circ\nabla_b\circ\Delta)$. Moreover, the soundness and (relative) completeness of the Hoare logic rules for while programs follow easily from Theorems 1 and 2.

4.2 Pointer programs (backward reasoning)

In the previous section we showed how one can obtain the standard Hoare logic for while programs as an instantiation of our general framework. We have presented a verification embedding that gives rise to the usual Hoare triple semantics

$$\dfrac{\dfrac{(\Delta \in \mathcal{F}_b)}{\{\langle P, P\rangle\}\ \Delta\ \{P\}} \qquad \dfrac{\dfrac{(\nabla_b \in \mathcal{F}_b)}{\{P\}\ \nabla_b\ \{\langle P \wedge \neg b, P \wedge b\rangle\}} \qquad \dfrac{\dfrac{(\mathsf{id} \in \mathcal{F}_b)}{\{P \wedge \neg b\}\ \mathsf{id}\ \{P \wedge \neg b\}} \quad \{\mathbf{P} \wedge \mathbf{b}\}\ \mathbf{f}\ \{\mathbf{P}\}}{\{\langle P \wedge \neg b, P \wedge b\rangle\}\ \mathsf{id} \uplus f\ \{\langle P \wedge \neg b, P\rangle\}}}{\{P\}\ (\mathsf{id} \uplus f) \circ \nabla_b\ \{\langle P \wedge \neg b, P\rangle\}}}{\dfrac{\{\langle P, P\rangle\}\ (\mathsf{id} \uplus f) \circ \nabla_b \circ \Delta\ \{\langle P \wedge \neg b, P\rangle\}}{\dfrac{\{P\}\ \mathsf{Tr}((\mathsf{id} \uplus f) \circ \nabla_b \circ \Delta)\ \{P \wedge \neg b\}}{\{\mathbf{P}\}\ \mathsf{while}_{\mathbf{b}}(\mathbf{f})\ \{\mathbf{P} \wedge \neg \mathbf{b}\}}\ (\mathsf{def})}\ (\mathsf{Tr})}$$

Fig. 4. Derivation of Hoare's while loop rule in $\mathbf{H}(\mathcal{F}, H)$

for *forward reasoning*. In this section we show how the verification functor can be changed so that one automatically gets Hoare logic rules for *backward reasoning*, from the same categorical rules presented in Section 3. We will illustrate backward reasoning using an extension of the category of flowcharts, namely the category of pointer programs $\mathcal{P}$ (see Section 2.2).

Consider the following functor $H : \mathcal{P} \to \mathsf{Preord}$. On the objects $X \in \mathcal{P}_o$ the embedding H returns the preordered set of formulas (see below for the description of the logical language used) describing subsets of $X \backslash \{\mathsf{abort}\}$. The preorder on $H(X)$ is: $R \sqsubseteq P$ iff $P \to R$, i.e. the logical equivalent of *reverse set inclusion*. On the morphisms of $\mathcal{P}$ we define

$$H(f)(P) :\equiv \mathsf{WPC}(f, P)$$

where $\mathsf{WPC}(f, P)$ is the weakest liberal pre-condition of the partial function f on post-condition P. We are assuming the dual of Cook's expressiveness condition, namely, that weakest liberal pre-conditions are expressible in the language.

According to our definition, abort is not an element of $H(X)$, which reflects the fact that Hoare triples for pointer programs ensure that programs do not crash (see [19]).

It has been shown in [17, 19], that the weakest liberal pre-conditions for the new basic statements can be concisely expressed in *separation logic* as (see [19] for notation)

$$\begin{aligned}
&\mathsf{WPC}(x := [t], P) &&:\equiv \exists v'((t \mapsto v') * ((t \mapsto v') \mathrel{-\!\!*} P[v'/x])) \\
&\mathsf{WPC}([t] := s, P) &&:\equiv (t \mapsto -) * ((t \mapsto s) \mathrel{-\!\!*} P) \\
&\mathsf{WPC}(x := \mathsf{new}(\boldsymbol{t}), P) &&:\equiv \forall i((i \mapsto \boldsymbol{t}) \mathrel{-\!\!*} P[i/x]) \\
&\mathsf{WPC}(\mathsf{disp}(t), P) &&:\equiv (t \mapsto -) * P
\end{aligned}$$

Similarly to Lemma 1 and Theorem 2, one can show that the H defined above is a verification functor. The system $\mathbf{H}(\mathcal{P}, H)$, which we then obtain from our abstract approach is basically the one presented in Reynolds [19], for *global backward reasoning*, using separation logic as an 'oracle' for the consequence rule.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\mathsf{id} \in \mathcal{F}_b)}{\{s\}\ \mathsf{id}\ \{s\}} \quad \{\mathbf{s}+\mathbf{t}\}\ \mathbf{f}\ \{\mathbf{t}\}}{\{\langle s, s+t\rangle\}\ \mathsf{id} \times f\ \{\langle s,t\rangle\}}(\times) \quad \dfrac{((+) \in \mathcal{F}_b)}{\{\langle s,t\rangle\}\ (+)\ \{s+t\}}}{\{\langle s, s+t\rangle\}\ (+) \circ (\mathsf{id} \times f)\ \{s+t\}}(\circ) \quad \dfrac{((\mathsf{c}) \in \mathcal{F}_b)}{\{s+t\}\ (\mathsf{c})\ \{\langle s+t, s+t\rangle\}}}{\{\langle s, s+t\rangle\}\ (\mathsf{c}) \circ (+) \circ (\mathsf{id} \times f)\ \{\langle s+t, s+t\rangle\}}(\circ)}{\{s\}\ \mathsf{Tr}((\mathsf{c}) \circ (+) \circ (\mathsf{id} \times f)\ \{s+t\}}(\mathsf{Tr})}{\{\mathbf{s}\}\ \mathsf{fdback}(\mathbf{f})\ \{\mathbf{s}+\mathbf{t}\}}(\text{def})$$

Fig. 5. Derivation of feedback rule in $\mathbf{H}(\mathcal{C}, H)$

4.3 Stream circuits

In [7], a Hoare logic was suggested for the frequency analysis of linear control systems with feedback, modelled as linear differential equations. Here we propose a different approach, based on the modelling of linear differential equations as stream circuits (also called signal flow graphs, see [20]).

We now present an embedding of the tmc $\mathcal{C}$ of stream circuits, described in Section 2.3, into Preord. Let $\mathcal{C}'_m$ be the set of (functions denoting) valid stream circuits. Notice that the class of valid circuits is closed under sub-circuits. We will show that our embedding is a verification functor for $\mathcal{C}'_m$, so that for valid circuits a set of sound and complete Hoare-logic-like rules is derived.

The monoidal functor $H : \mathcal{C} \to \mathsf{Preord}$ is defined as follows. For each of the objects $X \in \mathcal{C}_o$ (X is of the form $\Sigma \times \ldots \times \Sigma$) we define $H(X)$ as the preordered set of all finite approximations (prefixes) of elements in X. We will use the variables q, r, s, t to range over elements of the objects of Preord. The preorder $t \sqsubseteq s$ on element of $H(X)$ is defined as $s \preceq t$ on $H(X)$, i.e. s is a prefix of t. The top of the preorder is the (tuple of) empty stream(s) ε. Intuitively, each element t of the preordered set corresponds to a subset of X having a common prefix t, with ε corresponding to whole set X. On the morphisms (stream circuits) $f : X \to Y$ in $\mathcal{C}_m$ and $t \in H(X)$, we define $H(f)(t)$ as

$$H(f)(t) :\equiv \mathsf{LCP}\{\sigma \ : \ (\tau f \sigma) \wedge (t \preceq \tau)\}$$

where, for a set of streams S, $\mathsf{LCP}(S)$ denotes the longest common prefix of all streams in that set. $H(f)$ is clearly a monotone function. For the basic morphisms f of $\mathcal{C}$, $H(f)$ can be easily described as:

$$\begin{aligned}
H(a\times)[t_0, \ldots, t_n] &:\equiv [at_0, \ldots, at_n] \\
H(R)[t_0, \ldots, t_n] &:\equiv [0, t_0, \ldots, t_n] \\
H(\mathsf{c})[t_0, \ldots, t_n] &:\equiv \langle [t_0, \ldots, t_n], [t_0, \ldots, t_n]\rangle \\
H(+)\langle [t_0, \ldots, t_n], [r_0, \ldots, r_m]\rangle &:\equiv [t_0 + r_0, \ldots, t_{\min\{n,m\}} + r_{\min\{n,m\}}]
\end{aligned}$$

It is easy to check that H respects the monoidal structure of $\mathcal{C}$, since the prefixes of $X \times Y$, i.e. $H(X \times Y)$, can be seen as pairs of prefixes, i.e. elements of $H(X) \times H(Y)$. We now argue that H is also a verification functor.

Lemma 2. *Let $f : X \times Z \to Y \times Z$ be such that $\mathsf{Tr}(f) \in \mathcal{C}'_m$. Moreover, let streams $t \in H(X)$ and $r \in H(Y)$ be fixed. If $r \preceq H(\mathsf{Tr}(f))(t)$ then, for some $q \in H(Z)$, $\langle r, q\rangle \preceq H(f)\langle t, q\rangle$.*

Proof. Notice that the mappings $H(f)$ are continuous, i.e. a finite prefix of the output only requires a finite prefix of the input. Moreover, by the condition that $\mathsf{Tr}(f)$ is a valid circuit, a finite prefix of the output of a trace only requires a finite prefix of the fixed points in the following sense: for a fixed $t \in H(X)$ the increasing chain

- $q_0 :\equiv \varepsilon$
- $q_{k+1} :\equiv \pi_2(H(f)\langle t, q_k\rangle)$

where π_2 denotes the second projection, is such that for some k we must have $\langle r', q_k\rangle \preceq H(f)\langle t, q_k\rangle$, where $r' = H(\mathsf{Tr}(f))(t)$. By our assumption that $r \preceq H(\mathsf{Tr}(f))(t)$ this implies $\langle r, q_k\rangle \preceq H(f)\langle t, q_k\rangle$, as desired. □

Theorem 3. *$H : \mathcal{C} \to \mathsf{Preord}$, as defined above, is a verification functor for valid circuits.*

Proof. By Lemma 2, it remains to be shown that if $(i)\, \langle r, q'\rangle \preceq H(f)\langle t, q'\rangle$, for some q', then $r \preceq H(\mathsf{Tr}(f))(t)$. Let τ be such that $t \prec \tau$. By the definition of $\mathsf{Tr}(f)$ (and the fact that f is a valid circuit) there exists a unique fixed point σ such that $\langle \tau, \sigma\rangle f\langle \nu, \sigma\rangle$. By the uniqueness of the fixed point we have that $q' \preceq \sigma$ (otherwise q' could be extended into a different fixed point). Finally, by our assumption (i) it follows that $r \preceq \nu\, (= \mathsf{Tr}(f)(t))$. □

This gives rise to a sound and complete Hoare-logic system for reasoning about *valid* stream circuits. Notice that the rules would not be sound had we not restricted ourselves to valid circuits. For instance, assuming that $a \neq 0$ we have that

$$\{\langle [a], \varepsilon\rangle\}\ \mathsf{c} \circ (+)\ \{\langle \varepsilon, \varepsilon\rangle\}$$

holds but it is not true that $\{[a]\}\ \mathsf{Tr}(\mathsf{c} \circ (+))\ \{\varepsilon\}$, as argued in Section 2.3.

In this instantiation, the Hoare triples $\{t\}\ f\ \{s\}$ stand for $s \preceq H(f)(t)$. Given that streams represent the Taylor expansion of analytic functions, in this particular example, the pre- and post-conditions in the Hoare logic range over partial sums for these Taylor expansions. We have then categorically obtained a sound and complete formal system $\mathbf{H}(\mathcal{C}, H)$, for reasoning about valid stream circuits and their input-output behaviour over classes of functions with a common partial Taylor sum. Given that a feedback circuit $\mathsf{fdback}(f)$ can be represented in $\mathcal{S}$ as $\mathsf{Tr}((\mathsf{c}) \circ (+) \circ (\mathsf{id} \times f)$, a rule for stream circuit feedbacks can then be derived (see Figure 5) in a similar fashion to the derivation of the rule for while loops (cf. Figure 4).

5 Conclusion and Related Work

In this final section, we discuss other abstractions of Hoare logic in the light of our work.

Kozen's [14] *Kleene Algebra with Test*, KAT, consists essentially of a Kleene algebra with a Boolean subalgebra. Hoare triples $\{P\}\ f\ \{Q\}$ are modelled as equations $Pf = PfQ$, using the multiplication of KAT. The rules of Hoare logic are then obtained as consequences of the equational theory of KAT. Although our work is based on similar ideas (reducing Hoare triples to preorder statements) there does not seem to be a clear cut connection between the two approaches. Whereas Kozen relies on the rich theory of KAT to derive the usual rules of Hoare logic, in our development we use a minimal theory of preordered sets for obtaining soundness and completeness.

Kozen's work is closely related to works on iteration theory [6, 16] and dynamic logic [18]. All these, however, focus on the semantics of Hoare logic over flowcharts and while programs, where the intrinsic monoidal structure is *disjoint union*. As we have shown in Section 4.3, our approach is more general including systems with an underlying *cartesian structure* as well.

Abramsky et al. [1] have also studied the categorical structure of Hoare logic, using the notion of *specification structures*. It is easy to see that a tmc $\mathcal{S}$ together with a verification functor $H : \mathcal{S} \to \mathsf{Preord}$ gives rise to a specification structure: H maps objects $X \in \mathcal{S}_o$ to sets $H(X)$, and $H(f)(P) \sqsubseteq Q$ defines a ternary relation $H(X) \times \mathcal{S}(X, Y) \times H(Y)$. The extra structure of preorder and trace, however, allows us to prove an abstract completeness theorem, which does not seem to be the focus of [1].

Blass and Gurevich [5] considered the *underlying logic of Hoare logic*. Since Cook's celebrated completeness result [8], Cook-expressive first-order logics have been used in proofs of relative completeness for Hoare logic. Blass and Gurevich have shown that existential fixed-point logic **EFL** is sufficient for proving Cook's completeness result, without the need for Cook's expressiveness condition. **EFL** contains the necessary constructions to ensure that the functor $H(f)(P)$ of Section 4.1 (producing strongest post-conditions of f on P) can be inductively built, rather than assumed to exist. The fixed-point construction is used in order to produce the fixed point Q of Lemma 1.

Much remains to be done: our main interest is in alternative verification embeddings H of the tmc $\mathcal{C}$ of stream circuits, which will allow other (hopefully more practical) Hoare logics for dynamical systems. One instance of traced symmetric monoidal categories that we have not explored here is that of finite dimensional Hilbert spaces and bounded linear mappings, where trace is defined as a generalisation of matrix trace. This seems significant given its connection with quantum computing and Girard's geometry of interaction [11].

Acknowledgements. The authors gratefully acknowledge support of the UK EPSRC grant GR/S31242/01. We also wish to thank Peter O'Hearn, Nick Benton, Rob Arthan and the anonymous referees for many relevant comments on an earlier version of this paper.

References

1. S. Abramsky, S. Gay, and R. Nagarajan. Specification structures and propositions-as-types for concurrency. In G. Birtwistle and F. Moller, editors, *Logics for Concurrency: Structure vs. Automata*, pages 5–40. Springer-Verlag, 1996.
2. K. R. Apt. Ten years of Hoare's logic: A survey – Part 1. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, October 1981.
3. E. S. Bainbridge. Feedback and generized logic. *Information and Control*, 31:75–96, 1976.
4. M. Berger, K. Honda, and N. Yoshida. A logical analysis of aliasing in imperative higher-order functions. In *ICFP 2005*, pages 280–293, 2005.
5. A. Blass and Y. Gurevich. The underlying logic of Hoare logic. *Bull. of the Euro. Assoc. for Theoretical Computer Science*, 70:82–110, 2000.
6. S. L. Bloom and Z. Ésik. Floyd-Hoare logic in iteration theories. *J. ACM*, 38(4):887–934, October 1991.
7. R. J. Boulton, R. Hardy, and U. Martin. A Hoare logic for single-input single-output continuous time control systems. *In Proceedings of the 6th International Workshop on Hybrid Systems, Computation and Control, Springer LNCS*, 2623:113–125, 2003.
8. S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7(1):70–90, February 1978.
9. M. H. Escardó and D. Pavlovic. Calculus in coinductive form. In *LICS'1998*, Indiana, USA, June 1998.
10. R. W. Floyd. Assigning meanings to programs. *Proc. Amer. Math. Soc. Symposia in Applied Mathematics*, 19:19–31, 1967.
11. E. Haghverdi and P. Scott. Towards a typed geometry of interaction. In L. Ong, editor, *CSL'2005, LNCS*, volume 3634, pages 216–231, 2005.
12. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–585, October 1969.
13. A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447–468, 1996.
14. D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic (TOCL)*, 1(1):60–76, 2000.
15. S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate texts in mathematics*. Springer, 2nd ed., 1998.
16. E. G. Manes and M. A. Arbib. *Algebraic Approaches to Program Semantics*. AKM series in theoretical computer science. Springer-Verlag, New York, NY, 1986.
17. P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In L. Fribourg, editor, *CSL'2001, LNCS*, volume 2142, pages 1–19. Springer-Verlag, 2001.
18. V. R. Pratt. Semantical considerations on Floyd-Hoare logic. In *FoCS'1976*, pages 109–121, 1976.
19. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS'2002*, pages 55–74, 2002.
20. J. J. M. M. Rutten. An application of stream calculus to signal flow graphs. *Lecture Notes in Computer Science*, 3188:276–291, 2004.
21. A. K. Simpson and G. D. Plotkin. Complete axioms for categorical fixed-point operators. In *LICS'2000*, pages 30–41, 2000.