A General Framework for Sound and Complete Floyd-Hoare Logics

ROB ARTHAN, URSULA MARTIN, ERIK A. MATHIESEN and PAULO OLIVA
Queen Mary, University of London

This paper presents an abstraction of Hoare logic to traced symmetric monoidal categories, a very general framework for the theory of systems. Our abstraction is based on a traced monoidal functor from an arbitrary traced monoidal category into the category of pre-orders and monotone relations. We give several examples of how our theory generalises usual Hoare logics (partial correctness of while programs, partial correctness of pointer programs), and provide some case studies on how it can be used to develop new Hoare logics (run-time analysis of while programs and stream circuits).

Categories and Subject Descriptors: F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

General Terms: Hoare logic, traced monoidal categories

Additional Key Words and Phrases: Stream circuits

1. INTRODUCTION

Under the general label of *Hoare logic*, the early work of Floyd [1967] and Hoare [1969] on axiom systems for flowcharts and while programs has been applied to various other domains, such as recursive procedures [Apt 1981], pointer programs [Reynolds 2002], and higher-order languages [Berger et al. 2005]. 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* [Cook 1978]. This is achieved via an abstraction of Hoare logic to the theory of traced symmetric monoidal categories [Joyal et al. 1996], 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 feedback (unbounded iteration). In fact, traced symmetric monoidal categories are closely related to Bainbridge's work [1976] on the duality between flowcharts and networks. The scope of traced symmetric monoidal categories, how-

This research was supported by EPSRC grants GR/M98340, GR/L48256 and EP/F02309X/1 and by an EPSRC studentship to the third author. Oliva also gratefully acknowledges support of the Royal Society under grant 516002.K501/RH/kk.

Authors' address: Department of Computer Science, Queen Mary, University of London, Mile End Road, London E1 4NS, UK.

ever, is much broader, being actively used, for instance, for modelling computation (e.g. [Simpson and Plotkin 2000]) and in connection with Girard's geometry of interaction (e.g. [Haghverdi and Scott 2005]).

The main feature of Hoare logic is the use of assertions P, Q specifying the input-output behaviour of a program. If A is a program, then the triple $\{P\}\ A\ \{Q\}$ states that on inputs satisfying P the program A, if it terminates, will produce a result which satisfies property Q. An inherent ordering among assertions, given by the consequence relation, is also available, allowing for the properties of input/output to be refined or relaxed. In particular, if $\{P\}\ A\ \{Q\}$ holds, and Q logically implies Q', then we must also have that $\{P\}\ A\ \{Q'\}$ holds. Abstractly, assertions can be viewed as objects of a *pre-ordered set*, with the logical implication as an instance of an ordering relation. The derived Hoare triple relation $\{\cdot\}\ A\ \{\cdot\}$ can then be viewed as *monotone* binary relation between the points of the pre-order.

More precisely, let $\mathcal{H}$ be the category of pre-ordered sets and monotone relations (see Section 3 for formal definition). We first identify a particular class of functors – which we call *verification functors* – between traced symmetric monoidal categories and subcategories of $\mathcal{H}$. 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 [1969] (respectively, O'Hearn and Reynolds logic [2002]). In order to illustrate the generality of our framework, we also derive a new sound and complete Hoare logic for the verification of running-time (and hence termination) of programs, and a sound Hoare logic for the verification of linear dynamical systems (modelled via stream circuits).

The chief contributions of this paper are as follows: *(i)* The definition of the concept of a verification functor, between traced symmetric monoidal categories and the category $\mathcal{H}$ (Section 3). *(ii)* An abstraction of Hoare triples in terms of verification functors (Definition 3.3). 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 [Cook 1978] and loop invariants, have a clear cut correspondence to some of our abstract notions. *(iii)* Sound and complete rules for our abstract notion of Hoare triples, over a fixed verification functor (Theorem 3.4). *(iv)* Four concrete instances of our abstraction, namely: partial correctness of while programs, partial correctness of pointer programs (separation logic), run-time analysis of while programs, and stream circuits (Section 4). In Section 6, we discuss the link between our work and other abstractions of Hoare logic.

2. SYSTEMS IN THE ABSTRACT

In this section we describe an abstraction of "system" to be used in our abstraction of the Floyd-Hoare logic [Floyd 1967; Hoare 1969]. Consider first the case of flowcharts, or programming languages in general. The essential construct in this case is the unbounded iteration. For instance, in the Figure 1 we have depicted a flowchart where the factorial of y is calculated on the variable x via an iterative process. As suggested in the picture, the iteration corresponds to a feedback loop from the (*yes*) branch of the boolean test, to the next step in the computation of the factorial. Note that the code inside the loop ($x := xy; y := y - 1$) can be seen as a local process which at each iteration is gearing the computation to achieve the desired result – the computation of the factorial of y.

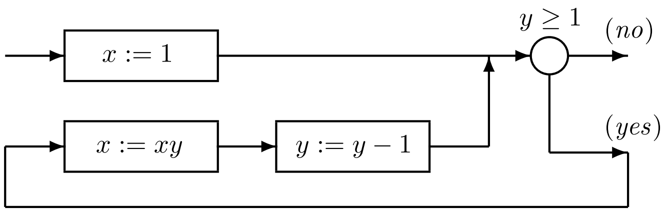


Fig. 1. Example of feedback for flowcharts

In the case of continuous or discrete dynamical systems, the feedback operation is essential for allowing the correction of minor errors and achieving stability. Consider a mechanical system comprising a cart of mass m attached to a wall by a spring of spring constant k and acted on by a force f. Newtonian mechanics gives us the differential equation $m\ddot{x} + kx - f = 0$, where x is the extension of the spring, as an implicit description of the behaviour of this system. In control engineering, we might take an intensional view of the physical system as a an input/output relation with f as input and x as output. With appropriate conventions for initial values, the differential equation may be viewed as an integral equation, $mx + k \int \int x - f = 0$, leading to the control law diagram of Figure 2. This diagrammatic expression of the system might serve, for example, as the design for an analogue computer to simulate the mechanical system. The spring here acts inside the feedback loop ($-k/m$) as a control element, trying to keep the cart within a particular region.

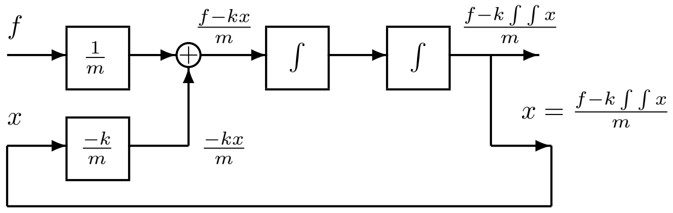


Fig. 2. Example of feedback for dynamic systems

As exemplified above, systems of very different kinds are in general often described via *block diagrams*, with a fixed block-diagram language $\mathcal{L}$. Systems in $\mathcal{L}$ are built from a set $\mathcal{L}^b$ of basic building blocks via sequential and parallel composition, and feedback (see Figure 3). Let us adopt the convention of using A, B, C for the syntactic block diagrams. Formally, the BNF for building block diagrams is

$$A, B \ ::\equiv\ \mathsf{Basic} \ \mid\ A; B \ \mid\ \mathsf{Par}(A, B) \ \mid\ \mathsf{Fb}(A)$$

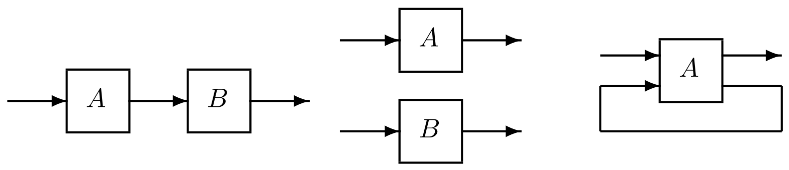


Fig. 3. Block diagram constructors: sequential composition $A; B$, parallel composition $\mathsf{Par}(A, B)$, and feedback $\mathsf{Fb}(A)$, respectively.

where Basic is a basic building block.

On closer inspection, one will notice that the main properties of the sequential composition are captured by the basic structure of a category[1]. In order to capture also the intrinsic properties of the parallel composition $\mathsf{Par}(A, B)$ and feedback $\mathsf{Fb}(A)$, a particular class of categories has been singled out, so-called *traced symmetric monoidal categories*.

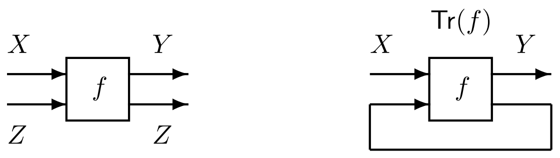


Fig. 4. Trace diagrammatically

Recall that a pair comprising a category $\mathcal{S}$ and a covariant bifunctor $\otimes$ 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 [Mac Lane 1998, chapter 11]). 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 [Joyal et al. 1996], the notion of *traced symmetric monoidal category* is introduced[2], 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 [Joyal et al. 1996] for details). We will normally omit the decoration in $\mathsf{Tr}^Z_{X,Y}$ whenever it is clear over which objects 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 4. In Sections 4 and 5 we will give the formal definition

[1]For the rest of this article we will assume some basic knowledge of category theory. For a readable introduction see [Mac Lane 1998]. 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 $g : Y \to Z$.

[2]In fact, [Joyal et al. 1996] 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.

of TMCs based on disjoint union (flowcharts) and cartesian product (networks). Before that we define the necessary concepts for our abstract Hoare logic system.

In our work, we are interested in traced monoidal categories which arise as the semantics of a syntactic block diagram language $\mathcal{L}$. More precisely, for each block diagram language $\mathcal{L}$ we will consider (possibly different) semantic mappings $[\![\cdot]\!]$ into particular traced monoidal categories $\mathcal{S}$. We will use f, g, h for the semantic functions denoted by these block diagrams. We assume that the semantic mapping is compositional, i.e.

$$\begin{aligned}
[\![A; B]\!] &= [\![B]\!] \circ [\![A]\!] \\
[\![\mathsf{Par}(A, B)]\!] &= [\![A]\!] \otimes [\![B]\!] \\
[\![\mathsf{Fb}(A)]\!] &= \mathsf{Tr}_{\mathcal{S}}([\![A]\!]).
\end{aligned}$$

Note that the objects and morphisms in the range of such a semantic mapping form a sub-TMC of $\mathcal{S}$ and we can view $\mathcal{L}$ as a typed language in which the types identify hom-sets in this sub-TMC and where the constructors combine the types of their operands as follows:

$$\begin{aligned}
(\cdot\,;\cdot) &\quad : \quad (\alpha \to \beta) \times (\beta \to \gamma) \Rightarrow \alpha \to \gamma \\
\mathsf{Par}(\cdot\,,\cdot) &\quad : \quad (\alpha \to \beta) \times (\gamma \to \delta) \Rightarrow (\alpha \otimes \gamma) \to (\beta \otimes \delta) \\
\mathsf{Fb}(\cdot) &\quad : \quad (\alpha \otimes \gamma) \to (\beta \otimes \gamma) \Rightarrow \alpha \to \beta.
\end{aligned}$$

3. ABSTRACT HOARE LOGIC

Recall that a pre-order is a pair $(X, \leq)$ consisting of a set X and a binary relation $\leq$ on X which is reflexive and transitive. For instance, the set of formulas of a fixed first-order theory under the consequence relation (i.e. $A \vdash B$) forms a pre-order, or any set of sets under subset inclusion $\subseteq$.

Let A be a while program, P, Q be assertions (formulas) over the program variables of A, and $\{P\}\ A\ \{Q\}$ denote the usual Hoare triple for while programs. The only property of the binary relation $\{\cdot\}\ A\ \{\cdot\}$ which does not depend on the structure of A, and hence is intrinsic to the Hoare triples themselves, is the monotonicity stated in the *consequence rule*:

if $\{P\}\ A\ \{Q\}$ holds and $P' \to P$ and $Q \to Q'$ then $\{P'\}\ A\ \{Q'\}$ also holds.

The following definition captures this basic property of assertions and Hoare triples. When r is a binary relation we write $x\ \underline{r}\ y$ as a shorthand for $\langle x, y\rangle \in r$.

Definition 3.1 *(Hoare category).* A relation $r : X \times Y$ between two pre-ordered sets X, Y is called *monotone* if $P\ \underline{r}\ Q$ and $P' \sqsubseteq_X P$ and $Q \sqsubseteq_Y Q'$ implies $P'\ \underline{r}\ Q'$. Let $\mathcal{H}$ denote the category of pre-ordered sets and monotone relations. We call $\mathcal{H}$ the *Hoare category.*

Since monotone relations are closed under composition, $\mathcal{H}$ is clearly a category. It is also easy to see that $\mathcal{H}$ can be considered a symmetric monoidal category, with the monoidal operation as cartesian product, since the cartesian product of two pre-ordered sets X, Y forms again a pre-ordered set with the order on $X \times Y$ defined coordinatewise, i.e., $\langle x, y\rangle \sqsubseteq \langle x', y'\rangle$ iff $x \sqsubseteq x'$ and $y \sqsubseteq y'$. It is also easily

$$\begin{tikzcd}\mathcal{L} \arrow[r, "\llbracket \cdot \rrbracket"] & \mathcal{S} \arrow[r, "H"] & \mathcal{H}\end{tikzcd}$$

Fig. 5. Block diagram language $\mathcal{L}$, traced monoidal semantic $\mathcal{S}$, and Hoare category $\mathcal{H}$. Recall that $\mathcal{S}$ is assumed to be a traced monoidal category.

verified that the trace based on cartesian product, i.e.

$$x \; \underline{\mathsf{Tr}(r)} \; y \;\; :\equiv \;\; \exists z(\langle x, z\rangle \; \underline{r} \; \langle y, z\rangle), \tag{1}$$

gives a trace on $\mathcal{H}$, making it into a TMC. Given a block diagram language $\mathcal{L}$, with its respective trace monoidal semantics, we define an abstract notion of Hoare logic using certain functors from the semantic domains into $\mathcal{H}$:

Definition 3.2 *(Verification functor)*. Let $\mathcal{L}$ be a fixed block diagram language with semantics $\llbracket \cdot \rrbracket : \mathcal{L} \to \mathcal{S}$. A mapping $H : \mathcal{S} \to \mathcal{H}$ is called a *verification functor* if it behaves like a strict traced monoidal functor[3] on the image[4] (a sub-TMC of $\mathcal{S}$) of the semantic mapping $\llbracket \cdot \rrbracket$ (see Figure 5).

Let a verification functor $H : \mathcal{S} \to \mathcal{H}$ be fixed. Each object $X \in \mathcal{S}_o$ corresponds to a pre-ordered set $H(X)$, and each morphism f in $\mathcal{S}$ corresponds to a monotone relation $H(f)$. Under the assumption that the semantic mapping $\llbracket \cdot \rrbracket$ is compositional, the verification functor condition is equivalent to:

(SC1) $H(\llbracket A; B \rrbracket) = H(\llbracket B \rrbracket) \circ H(\llbracket A \rrbracket)$

(SC2) $H(\llbracket \mathsf{Par}(A, B) \rrbracket) = H(\llbracket A \rrbracket) \times H(\llbracket B \rrbracket)$

(SC3) $H(\llbracket \mathsf{Fb}(A) \rrbracket) = \mathsf{Tr}_{\mathcal{H}}(H(\llbracket A \rrbracket))$

for all block diagrams A, B in the block diagram language $\mathcal{L}$. Relational equality will be needed for soundness and completeness of the corresponding Hoare logic. As we will see, if only soundness is required only relational inclusion is needed, i.e.

(S1) $H(\llbracket A; B \rrbracket) \supseteq H(\llbracket B \rrbracket) \circ H(\llbracket A \rrbracket)$

(S2) $H(\llbracket \mathsf{Par}(A, B) \rrbracket) \supseteq H(\llbracket A \rrbracket) \times H(\llbracket B \rrbracket)$

(S3) $H(\llbracket \mathsf{Fb}(A) \rrbracket) \supseteq \mathsf{Tr}_{\mathcal{H}}(H(\llbracket A \rrbracket))$

Definition 3.3 *(Abstract Hoare Triples)*. Let A be a concrete block diagram, whose meaning is a morphism $\llbracket A \rrbracket : X \to Y$ in $\mathcal{S}$. Moreover, let $P \in H(X)$ and $Q \in H(Y)$. Define *abstract Hoare triples* as

$$\{P\} \; A \; \{Q\} \;\; :\equiv \;\; P \; \underline{H(\llbracket A \rrbracket)} \; Q \tag{2}$$

[3] I.e. a functor that preserves both the trace structure and the monoidal structure.

[4] H only needs to behave well on the image of the semantics mapping, since at the end we are only interested in the properties of actual programs.

$$\frac{}{\{P\}\ A\ \{Q\}}\ (\mathsf{Ax})\ (\dagger) \qquad \frac{\{P\}\ A\ \{Q\}}{\{P'\}\ A\ \{Q'\}}\ (\mathsf{Con})\ (\ddagger)$$

$$\frac{\{P\}\ A\ \{Q\} \quad \{R\}\ B\ \{S\}}{\{\langle P, R\rangle\}\ \mathsf{Par}(A, B)\ \{\langle Q, S\rangle\}}\ (\mathsf{Par})$$

$$\frac{\{P\}\ A\ \{Q\} \quad \{Q\}\ B\ \{R\}}{\{P\}\ A; B\ \{R\}}\ (\mathsf{Seq}) \qquad \frac{\{\langle P, Q\rangle\}\ A\ \{\langle R, Q\rangle\}}{\{P\}\ \mathsf{Fb}(A)\ \{R\}}\ (\mathsf{Fb})$$

Fig. 6. The system $\mathbf{HL}(\mathcal{L}, [\![\cdot]\!], H)$ with $[\![\cdot]\!] : \mathcal{L} \to \mathcal{S}$ and $H : \mathcal{S} \to \mathcal{H}$.

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 A then, if A terminates, Q holds afterwards* will be one of the special cases of our general theory. See Sections 4 and 5 for other meanings of $\{P\}\ A\ \{Q\}$.

Let a specific block diagram language be fixed, and let $\mathcal{S}$ be the traced monoidal category denoting the programs of the given language. Moreover, let $H : \mathcal{S} \to \mathcal{H}$ be a fixed verification functor. We will denote by $\mathbf{HL}(\mathcal{L}, [\![\cdot]\!], H)$ the set of rules shown in Figure 6, where the side conditions are:

($\dagger$) $A \in \mathcal{L}^b$ and $P\ \underline{H([\![A]\!])}\ Q$

($\ddagger$) $P' \sqsubseteq P$ and $Q \sqsubseteq Q'$

The formal system $\mathbf{HL}(\mathcal{L}, [\![\cdot]\!], H)$ should be viewed as a *syntactic* axiomatisation[5] of the ternary relation $\{P\}\ A\ \{Q\}$. The verification functor H gives the *semantics* of the Hoare triples (and rules). By soundness and completeness of the system $\mathbf{HL}(\mathcal{L}, [\![\cdot]\!], H)$ we mean that syntax corresponds precisely to semantics, i.e. a syntactic Hoare triple $\{P\}\ A\ \{Q\}$ is provable in $\mathbf{HL}(\mathcal{L}, [\![\cdot]\!], H)$ if and only if $P\ \underline{H([\![A]\!])}\ Q$ is true in $\mathcal{H}$.

THEOREM 3.4 (SOUNDNESS AND COMPLETENESS). *The system* $\mathbf{HL}(\mathcal{L}, [\![\cdot]\!], H)$ *is sound and complete.*

Proof. Soundness is trivially true for the axioms. The consequence rule is sound by the monotonicity of the relation $H([\![A]\!])$. Soundness of the composition rule also uses the fact that H respects composition, i.e.

$$P\ \underline{H([\![A; B]\!])}\ R \quad \Leftrightarrow \quad P\ \underline{H([\![B]\!]) \circ H([\![A]\!])}\ R.$$

[5]Note that assertions P, Q, R could potentially be semantic objects (e.g. sets), which is harmless since the only structure required from assertions is the ability to form a pair $\langle P, Q\rangle$ out of two assertions P and Q. Moreover, the fact that logical axioms have a semantic side condition ($\dagger$) is not too restrictive since this is only assumed for the basic block diagrams, and these are normally manageable computationally.

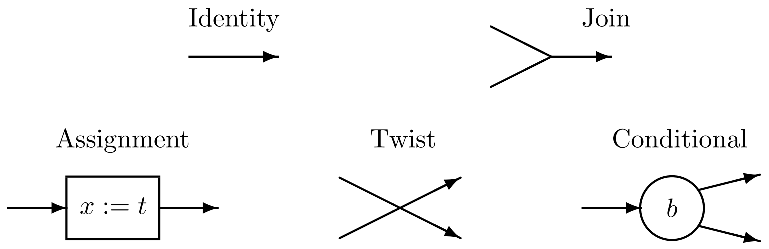


Fig. 7. Basic flowcharts.

Soundness of the cartesian product rule uses that the functor H is monoidal, i.e.

$$\langle P, Q\rangle\ \underline{H([\![A \otimes B]\!])}\ \langle R, S\rangle \quad\Leftrightarrow\quad \langle P, Q\rangle\ \underline{H([\![A]\!]) \times H([\![B]\!])}\ \langle R, S\rangle.$$

For the soundness of the trace rule, assume $\langle P, Q\rangle\ \underline{H([\![A]\!])}\ \langle R, Q\rangle$. By the definition of the trace on $\mathcal{H}$ we have $P\ \underline{\mathsf{Tr}(H([\![A]\!]))}\ R$. Finally, by the fact that H is traced we have $P\ \underline{H([\![\mathsf{Fb}(A)]\!])}\ R$. We argue now about completeness. By side condition (†) it follows that all true statements of the form $\{P\}\ A\ \{Q\}$, for basic diagrams A, are provable. It remains to show that if $\{P\}\ A\ \{Q\}$ is true, for an arbitrary block diagram A, then there exists a premise of the corresponding rule (depending on the structure of A) which is also true. If $\{P\}\ A; B\ \{R\}$ is true then, since H respects composition, there must exists a Q such that both

$$P\ \underline{H([\![A]\!])}\ Q \quad \text{and} \quad Q\ \underline{H([\![B]\!])}\ R$$

are true. If $\{\langle P, R\rangle\}\ A \otimes B\ \{\langle Q, S\rangle\}$ is true then so it is $\{P\}\ A\ \{Q\}$ and $\{R\}\ B\ \{S\}$, by the fact that H is monoidal. Finally, if $\{P\}\ \mathsf{Fb}(A)\ \{R\}$ is true, then so is

$$\langle P, Q\rangle\ \underline{H([\![A]\!])}\ \langle R, Q\rangle,$$

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 $\mathcal{H}$. As we will see, the laborious work is pushed into showing that H is a verification functor. That will be clear in Section 4.3, where we build a verification functor appropriate for while programs, using Cook's expressiveness condition [Cook 1978].

4. VERIFICATION FUNCTORS FOR FLOWCHARTS

The traced monoidal categories that correspond to classical Hoare logics for imperative programs are based on the category of sets and relations with the monoidal structure arising from the disjoint union operator $\uplus$. In this case, the relations arise from the input-output relations of flowcharts. We will write $\mathsf{inl} : X_0 \to X_0 \uplus X_1$ and $\mathsf{inr} : X_1 \to X_0 \uplus X_1$ for the injections of the summands into a disjoint union. In this TMC, the trace of a relation $f : X \uplus Z \to Y \uplus Z$ is then defined as follows:

$$x\ \underline{\mathsf{Tr}(f)}\ y\ :\equiv\ \exists z_0 \ldots z_n(\mathsf{inl}\, x\ \underline{f}\ \mathsf{inr}\, z_0 \wedge \ldots \mathsf{inr}\, z_i\ \underline{f}\ \mathsf{inr}\, z_{i+1} \ldots \wedge \mathsf{inr}\, z_n\ \underline{f}\ \mathsf{inl}\, y)$$

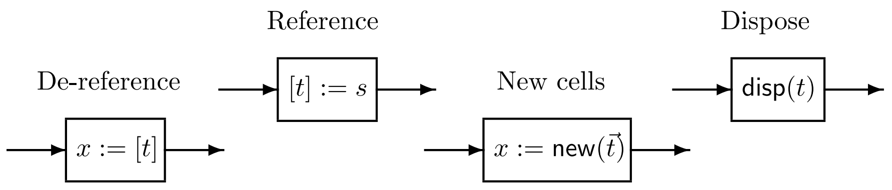


Fig. 8. Basic pointer programs.

This trace based on disjoint union lets us view feedback loops as representing iterative processes.

First we will consider systems (block diagrams) where disjoint union is used for the monoidal structure in the semantic domains. These are normally called "flowcharts", and two instances of these are (refinements of) while programs and pointer programs. The instantiations will produce the original Hoare logic [Hoare 1969], separation logic and a new Hoare logic for running time analysis.

4.1 The programming language: flowcharts

In the case of simple diagrammatic while programs the basic constructs of our language $\mathcal{L}_{\mathbb{FC}}$ are shown in Figure 7, where x stands for a program variable, t for an numerical expression over program variables, and b for a boolean expression over program variables. These can be put together via sequential composition, parallel composition and feedback, as shown in Figure 3. We will consider an extension $\mathcal{L}_{\mathbb{PP}}$ of this programming language with pointers by adding the basic pointer programs shown in Figure 8, where x denotes a program variables, and t, s denote numerical expressions over program variables.

4.2 The semantics

In this section we define the semantic categories $\mathcal{S}$ to be used later. The categories $\mathcal{S}$ will be defined inductively and the semantic mappings $[\![\cdot]\!] : \mathcal{L} \to \mathcal{S}$ will be (implicitly) given by obvious recursive definitions.

Semantic category $\mathcal{S}_{\mathbb{FC}}$ for $\mathcal{L}_{\mathbb{FC}}$. Let Store be the set of mappings $\rho : \mathsf{Var} \to \mathbb{Z}$ assigning an integer value to each of the program variables $\mathsf{Var} = \{x, y, \ldots\}$. Moreover, let $n\mathsf{Store}$ denote the coproduct of n copies of Store, i.e. $\mathsf{Store} \uplus \ldots \uplus \mathsf{Store}$. The objects of $\mathcal{S}_{\mathbb{FC}}$ are taken to be $n\mathsf{Store}$ for each natural number n. Consider also the following family of functions between the objects of $\mathcal{S}_{\mathbb{FC}}$:

- *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*, $\nabla : \mathsf{Store} \uplus \mathsf{Store} \to \mathsf{Store}$

  $\nabla(\mathsf{inl}(\rho)) :\equiv \rho$

  $\nabla(\mathsf{inr}(\rho)) :\equiv \rho$

- *Twist*, $c : \mathsf{Store} \uplus \mathsf{Store} \to \mathsf{Store} \uplus \mathsf{Store}$
  $\mathsf{twist}(\langle v, \rho \rangle) :\equiv \langle \overline{v}, \rho \rangle$

- *Forking*, $\Delta_b : \mathsf{Store} \to \mathsf{Store} \uplus \mathsf{Store}$

$$\Delta_b(\rho) :\equiv \begin{cases} \mathsf{inl}(\rho) & \mathsf{if}\ \neg b_\rho \\ \mathsf{inr}(\rho) & \mathsf{otherwise} \end{cases}$$

The conditional forking (Δ_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 inl and inr for left and right injections into $\mathsf{Store} \uplus \mathsf{Store}$.

We then close the set of basic functions in $\mathcal{S}_{\mathbb{FC}}$ under sequential composition of functions, disjoint union of functions and the standard trace for disjoint union, to form the set of partial functions which are the morphisms of $\mathcal{S}_{\mathbb{FC}}$. As such, $\mathcal{S}_{\mathbb{FC}}$ is the smallest traced monoidal category containing the finite coproducts of Store and the above morphisms.

Semantic category $\mathcal{S}^T_{\mathbb{FC}}$ for $\mathcal{L}_{\mathbb{FC}}$. In $\mathcal{S}_{\mathbb{FC}}$ above, each program A is mapped to a partial function on Store. Therefore, programs with equal input-output behaviour will be equated under this semantics. In Section 4.5 we will need a finer semantics, which distinguishes programs with different running time (even if input-output behaviour is similar). The running time of a program A on input ρ is defined as the number of primitive statements 'assignment', 'conditional' and 'joining' executed until termination (or ∞ if program diverges). The objects of $\mathcal{S}^T_{\mathbb{FC}}$ will be the same as in $\mathcal{S}_{\mathbb{FC}}$. Morphisms $f : n\mathsf{Store} \to m\mathsf{Store}$ from $\mathcal{S}_{\mathbb{FC}}$, however, will give rise to a family of morphisms, $\{f\} \times (n\mathsf{Store} \to \mathbb{N}^\infty)$, where the component $n\mathsf{Store} \to \mathbb{N}^\infty$ records the running time of the program A (for $[\![A]\!] = f$). For each morphism f ($= [\![A]\!]$) let us denote the first component by f_{io} (input-output behaviour) and the second as f_{rt} (running time). We now argue that $\mathcal{S}^T_{\mathbb{FC}}$ is also a traced monoidal category. On the first component of f, composition, coproduct and trace are as defined above. For the component $(\cdot)_{\mathsf{rt}}$ we let composition be defined as

$$(f \circ g)_{\mathsf{rt}}^{n\mathsf{Store} \to \mathbb{N}^\infty} :\equiv \lambda \rho^{n\mathsf{Store}} \begin{cases} g_{\mathsf{rt}}(\rho) + f_{\mathsf{rt}}(g_{\mathsf{io}}(\rho)) & \mathsf{if}\ g_{\mathsf{io}}(\rho) \downarrow \\ \infty & \mathsf{if}\ g_{\mathsf{io}}(\rho) \uparrow, \end{cases}$$

with identity $\lambda\rho.0$. Disjoint union for the running time component is defined as

$$(f \uplus g)_{\mathsf{rt}}^{(m+n)\mathsf{Store} \to \mathbb{N}^\infty} :\equiv \lambda \rho^{(m+n)\mathsf{Store}}. \begin{cases} f_{\mathsf{rt}}(\rho) & \mathsf{if}\ \rho \in m\mathsf{Store} \\ g_{\mathsf{rt}}(\rho) & \mathsf{if}\ \rho \in n\mathsf{Store}. \end{cases}$$

Finally, using the abbreviation

$$A(\rho, \vec{z}, y) :\equiv \mathsf{inl}\,\rho \;\underline{f}\; \mathsf{inr}\, z_0 \wedge \ldots \mathsf{inr}\, z_i \;\underline{f}\; \mathsf{inr}\, z_{i+1} \ldots \wedge \mathsf{inr}\, z_k \;\underline{f}\; \mathsf{inl}\, y$$

the trace for the running time component is defined as

$$x(\mathsf{Tr}(f))_1^{n\mathsf{Store} \to \mathbb{N}^\infty} y :\equiv \lambda \rho^{(m+n)\mathsf{Store}}. \begin{cases} f_1(\rho) + \Sigma_{i=0}^k f_1(z_i) & \mathsf{if}\ \exists \vec{z}, y A(\rho, \vec{z}, y) \\ \infty & \mathsf{otherwise.} \end{cases}$$

This satisfies the trace axioms, since the the trace axiom only involve operations that do not change the running time of the program (such as sliding and braiding).

Semantic category $\mathcal{S}_{\mathbb{PP}}$ for $\mathcal{L}_{\mathbb{PP}}$. In the case of pointer programs, we consider an adaptation of $\mathcal{S}_{\mathbb{FC}}$ 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* $\mathcal{S}_{\mathbb{PP}}$. Let $\mathsf{State} :\equiv \mathsf{Store} \times \mathsf{Heap}$, where Store is as above and Heap is the set of partial functions from $\mathbb{N}$ to $\mathbb{Z}$ with finite domain. Define also for any set X a new set X_{a} as $X \cup \{\mathsf{abort}\}$. 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. Similarly to above, we denote by $n\mathsf{State}$ the coproduct of n copies of State. The objects of $\mathcal{S}_{\mathbb{PP}}$ consist of $(n\mathsf{State})_{\mathsf{a}}$ for each natural number n, i.e. $\emptyset, \mathsf{State}_{\mathsf{a}}, (\mathsf{State} \uplus \mathsf{State})_{\mathsf{a}}, \ldots$. It is easy to check that $\mathcal{S}_{\mathbb{PP}}$ is also a TMC. Each of the basic functions of $\mathcal{S}_{\mathbb{FC}}$ can be lifted to the extended type structure, by simply ignoring the 'heap' component, or by propagating abort when receiving an abort as input. The set of basic functions of $\mathcal{S}_{\mathbb{PP}}$ is an extension of the set of (the lifting of the) basic functions of $\mathcal{S}_{\mathbb{FC}}$ with the following family of functions:

—*Look up*, $(x := [t]) : \mathsf{State}_{\mathsf{a}} \to \mathsf{State}_{\mathsf{a}}$

$$(x := [t])(\rho, h, d) :\equiv \begin{cases} (\rho[h(t_\rho)/x], h, d) & t_\rho \in d \\ \mathsf{abort} & \mathsf{otherwise} \end{cases}$$

—*Mutation*, $([t] := s) : \mathsf{State}_{\mathsf{a}} \to \mathsf{State}_{\mathsf{a}}$

$$([t] := s)(\rho, h, d) :\equiv \begin{cases} (\rho, h[t_\rho \mapsto s_\rho], d) & t_\rho \in d \\ \mathsf{abort} & \mathsf{otherwise} \end{cases}$$

—*Allocation*, $x := \mathsf{new}(\vec{t}) : \mathsf{State}_{\mathsf{a}} \to \mathsf{State}_{\mathsf{a}}$

$$(x := \mathsf{new}(\vec{t}))(\rho, h, d) :\equiv (\rho[x \mapsto i], h[i + j \mapsto t_j]_{j \in \{0,\ldots,n\}}, d \uplus \{i, \ldots, i + n\})$$

—*Deallocation*, $\mathsf{disp}(t) : \mathsf{State}_{\mathsf{a}} \to \mathsf{State}_{\mathsf{a}}$

$$(\mathsf{disp}(t))(\rho, h, d) :\equiv \begin{cases} (\rho, h, d \backslash \{t_\rho\}) & t_\rho \in d \\ \mathsf{abort} & \mathsf{otherwise,} \end{cases}$$

where $\vec{t} \equiv t_0, \ldots, t_n$, and i is the smallest number such that $i + k \notin d$, for $0 \leq k \leq n$.

As done in the case of flowcharts above, we can then close the set of basic functions under sequential composition, disjoint union and trace, to form the set of partial functions which are the morphisms of $\mathcal{S}_{\mathbb{PP}}$. 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 $\mathcal{S}_{\mathbb{PP}} \equiv (\mathcal{S}_{\mathbb{PP}}, \uplus, \mathsf{Tr})$, with the standard trace for disjoint union, forms another example of a TMC with the extra basic morphisms look up, mutation, allocation and deallocation.

4.3 Hoare logic for partial correctness

In this section we present a verification embedding of the TMC of flowcharts $\mathcal{S}_{\mathbb{FC}}$. This will give us soundness and (relative) completeness of Hoare's original verification logic [Hoare 1969] for partial correctness (using forward reasoning).

Let us now define the monoidal functor $H_{\mathbb{FC}} : \mathcal{S}_{\mathbb{FC}} \to \mathcal{H}$. Let a Cook-expressive[6] first-order theory be fixed. On the objects $X \in \mathcal{S}_{\mathbb{FC}}$ we let $H_{\mathbb{FC}}(X)$ be a pre-ordered

[6]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. [Cook 1978]).

set of first-order formulas over the expression language of our given programming language. The ordering $P \sqsubseteq R$ on the elements of $H_{\mathbb{FC}}(X)$ is taken to be $P \to R$ in the fixed theory. We assume that pairs of formulas are also considered formulas, with the connectives defined pointwise. We now define the functor $H_{\mathbb{FC}}$ on the image of the semantic embedding $[\![\cdot]\!]$. For each flowchart A we let $H_{\mathbb{FC}}([\![A]\!])$ be the following monotone relation

$$P \; \underline{H_{\mathbb{FC}}([\![A]\!])} \; Q \;:\equiv\; \mathsf{SPC}(A, P) \to Q$$

where $\mathsf{SPC}(A, P)$ is a formula expressing the strongest post-condition of A under P. Such formula exists by our assumption that the theory is Cook-expressive. Moreover, note that the denotation of $\underline{H_{\mathbb{FC}}([\![A]\!])}$, since it only depends on A via $\mathsf{SPC}(A, \cdot)$, does not depend on the particular syntax of A, but only on the input-output behaviour of A, i.e. $[\![A]\!]$. For this reason we often write $\mathsf{SPC}(A, \cdot\,)$ as $\mathsf{SPC}([\![A]\!], \cdot\,)$. It is also easy to see what SPC is for the basic morphisms of $\mathcal{S}_{\mathbb{FC}}$

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

Since mapping H preserves composition:

$$\begin{aligned}
P \; \underline{H_{\mathbb{FC}}([\![B]\!] \circ [\![A]\!])} \; Q \quad &\Leftrightarrow \quad P \; \underline{H_{\mathbb{FC}}([\![A; B]\!])} \; Q\\
&\Leftrightarrow \quad \mathsf{SPC}(A; B, P) \to Q\\
&\Leftrightarrow \quad \exists R((\mathsf{SPC}(A, P) \to R) \wedge (\mathsf{SPC}(B, R) \to Q))\\
&\Leftrightarrow \quad \exists R((P \; \underline{H_{\mathbb{FC}}([\![A]\!])} \; R) \wedge (R \; \underline{H_{\mathbb{FC}}([\![B]\!])} \; Q))
\end{aligned}$$

it is clearly a functor. Moreover, H is also 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. $H_{\mathbb{FC}}(X \uplus Y)$ is isomorphic to $H_{\mathbb{FC}}(X) \times H_{\mathbb{FC}}(Y)$. For sake of completeness we include the isomorphism here

(i) $f \;:\; H(X_0 \uplus X_1) \to H(X_0) \times H(X_1)$

$$f(P(z^{X_0 \uplus X_1})) :\equiv \langle P(\mathsf{inl}(x^{X_0})), P(\mathsf{inr}(y^{X_1}))\rangle$$

(ii) $g \;:\; H(X_0) \times H(X_1) \to H(X_0 \uplus X_1)$

$$g(P_0(x^{X_0}), P_1(y^{X_1})) :\equiv \forall x(z = \mathsf{inl}(x) \to P_0(x)) \wedge \forall y(z = \mathsf{inr}(y) \to P_1(y)).$$

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 $H_{\mathbb{FC}}(f) \times H_{\mathbb{FC}}(g)$.

We argue now in two steps that $H_{\mathbb{FC}}$ is also a verification functor. The main task is to show that $H_{\mathbb{FC}}$ respects the trace structure, i.e.

$$P \; \underline{H_{\mathbb{FC}}(\mathsf{Tr}([\![A]\!]))} \; R \quad \Leftrightarrow \quad P \; \underline{\mathsf{Tr}(H_{\mathbb{FC}}([\![A]\!]))} \; R.$$

By the definition of trace on the $\mathcal{H}$ we get

$$P\ \underline{H_{\mathbb{FC}}(\mathsf{Tr}([\![A]\!]))}\ R \quad\Leftrightarrow\quad \exists Q\ (\langle P,Q\rangle\ \underline{H_{\mathbb{FC}}([\![A]\!])}\ \langle R,Q\rangle)$$

and by the definition of $H_{\mathbb{FC}}$ above:

$$\mathsf{SPC}(\mathsf{Tr}([\![A]\!]),P)\to R \quad\Leftrightarrow\quad \exists Q\ (\mathsf{SPC}([\![A]\!],\langle P,Q\rangle)\to\langle R,Q\rangle).$$

The next lemma proves the left to right implication. The implication from right to left is proven in Theorem 4.2.

Lemma 4.1. *Let A be a block diagram such that $[\![A]\!] : X\uplus Z\to Y\uplus Z$. Assume $\vec{z}$ are all the program variables of A. Moreover, let $P\in H_{\mathbb{FC}}(X)$ and $R\in H_{\mathbb{FC}}(Y)$ be fixed formulas. If $\mathsf{SPC}([\![\mathsf{Fb}(A)]\!],P)\to R$ then $\mathsf{SPC}([\![A]\!],\langle P,Q\rangle)\to\langle R,Q\rangle$, for some formula Q.*

Proof. The corresponding lemma for while programs is proven in [Cook 1978]. In here we reformulate Cook's proof in our abstract setting of traced monoidal categories. We construct a formula Q such that

(i) $\mathsf{SPC}([\![A]\!],\langle P,Q\rangle)\leftrightarrow\langle R',Q\rangle$, for some formula R',

and then argue that (i) implies

(ii) $\mathsf{SPC}([\![\mathsf{Fb}(A)]\!],P)\leftrightarrow R'$.

By our hypothesis $\mathsf{SPC}([\![\mathsf{Fb}(A)]\!],P)\to R$ it will then follow that R' implies R, as desired. The formula Q is essentially the strongest loop invariant.

Construction of Q. Given the block diagram A we build a new block diagram $A' : X\uplus Z\to Y\uplus Z\uplus Z$ where the internal states of Z can be observed, even after the feedback is applied. Let $A' :\equiv A;\mathsf{Par}(\mathsf{id},\Delta_{\vec{z}=\vec{y}})$ (see Figure 9) where $\vec{z}$ is the

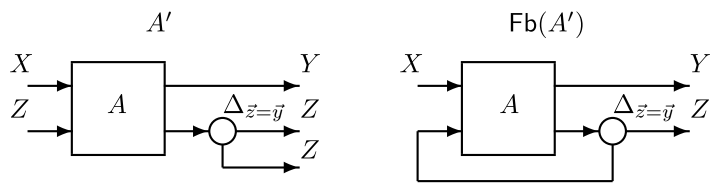


Fig. 9. Cook's construction (forward reasoning)

finite sequence of variables mentioned in the description of A, and $\vec{y}$ is a fresh tuple of variables of same length. Notice that the block diagram $\mathsf{Fb}^{Z}_{X,Y\uplus Z}(A')$ behaves in the same way as $\mathsf{Fb}^{Z}_{X,Y}(A)$ except that the block diagram $\mathsf{Fb}(A')$ will 'terminate' earlier (meaning that the fixed point sequence is shorter) if the state $\vec{z}$ matches $\vec{y}$. Let $\mathsf{SPC}([\![\mathsf{Fb}(A')]\!],P)=\langle Q_0,Q_1(\vec{y})\rangle$, and define $Q[\vec{z}] :\equiv \exists\vec{y}\,Q_1(\vec{y})[\vec{z}]$.

Proof of (i). Let $\langle R',Q'\rangle :\equiv \mathsf{SPC}([\![A]\!],\langle P,Q\rangle)$. We must show that $Q'[\vec{z}]\Leftrightarrow Q[\vec{z}]$. Unwinding the definition of $Q[\vec{z}]$, using the definition of strongest post condition, we obtain

$$\exists\vec{y}\exists\rho_0\in P\exists\rho_1,\ldots,\rho_{n-1}((\mathsf{inl}\rho_0\,A\,\mathsf{inr}\rho_1)\wedge\textstyle\bigwedge_{i=1}^{n-2}(\mathsf{inr}\rho_i\,A\,\mathsf{inr}\rho_{i+1})\wedge(\mathsf{inr}\rho_{n-1}\,A\,\mathsf{inr}\vec{z})).$$

On the other hand, unwinding the definition of $Q'[\vec{z}]$ gives

$$\exists \rho_0 \in P(\mathsf{inl}\rho_0 \, A \, \mathsf{inr}\vec{z}) \vee \exists \rho_n \in Q(\mathsf{inr}\rho_n \, A \, \mathsf{inr}\vec{z}),$$

which can be seen as an equivalent way of writing $Q[\vec{z}]$, as both formulas say that $\vec{z}$ is a reachable (internal) state during the run of $\mathsf{Fb}(A)$ from an initial state satisfying P.

Proof of (ii). What we have shown in point (i) is that Q characterises the internal states of $\mathsf{Fb}(A)$ on inputs satisfying P. Similarly, $R''[\vec{z}] :\equiv \mathsf{SPC}([\![\mathsf{Fb}(A)]\!], P)[\vec{z}]$ expresses that

$$\exists \rho_0 \in P \exists \rho_1, \ldots, \rho_{n-1}((\mathsf{inl}\rho_0 \, A \, \mathsf{inr}\rho_1) \wedge \textstyle\bigwedge_{i=1}^{n-2}(\mathsf{inr}\rho_i \, A \, \mathsf{inr}\rho_{i+1}) \wedge (\mathsf{inr}\rho_{n-1} \, A \, \mathsf{inl}\vec{z}))$$

whereas the definition of R' gives

$$\exists \rho_0 \in P(\mathsf{inl}\rho_0 \, A \, \mathsf{inl}\vec{z}) \vee \exists \rho_n \in Q(\mathsf{inr}\rho_n \, A \, \mathsf{inl}\vec{z}).$$

Both characterise the output states from a terminating run starting from a state satisfying P. $\Box$

THEOREM 4.2. *$H_{\mathbb{FC}} : \mathcal{S}_{\mathbb{FC}} \to \mathcal{H}$, as defined above, is a verification functor for morphisms arising from block diagrams in $\mathcal{L}_{\mathbb{FC}}$.*

Proof. By Lemma 4.1, it remains to be shown that whenever

(i) $\mathsf{SPC}([\![A]\!], \langle P, Q\rangle) \to \langle R, Q\rangle$,

for a formula Q, then $\mathsf{SPC}([\![\mathsf{Fb}(A)]\!], P) \to R$. Assume (i) and $\mathsf{SPC}([\![\mathsf{Fb}(A)]\!], P)(\rho)$, 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

$$\mathsf{inl}\rho' \; \underline{[\![A]\!]} \; \mathsf{inr}\rho_0, \ldots, \mathsf{inr}\rho_i \; \underline{[\![A]\!]} \; \mathsf{inr}\rho_{i+1}, \ldots, \mathsf{inr}\rho_n \; \underline{[\![A]\!]} \; \mathsf{inl}\rho.$$

By a simple induction, using the assumption (i), we get that all ρ_i satisfy Q and that ρ satisfies R, as desired. $\Box$

Given the semantic embedding $[\![\cdot]\!] : \mathcal{L}_{\mathbb{FC}} \to \mathcal{S}_{\mathbb{FC}}$, the system $\mathbf{HL}(\mathcal{L}_{\mathbb{FC}}, [\![\cdot]\!], H_{\mathbb{FC}})$ obtained via our embedding $H_{\mathbb{FC}}$ is a refinement of the system given by Hoare [1969], i.e. Hoare's rules are derivable from ours. See, for instance, the case of the while loop rule in Figure 10, given that a while loop $\mathsf{while}_b(A)$ can be represented in $\mathcal{L}_{\mathbb{FC}}$ as $\mathsf{Fb}(\nabla; \Delta_b; \mathsf{Par}(\mathsf{id}, A))$. Moreover, the soundness and (relative) completeness of the Hoare logic rules for while programs follow easily from Theorems 3.4 and 4.2.

4.4 Separation logic

In the case of the extended flowchart language $\mathcal{L}_{\mathbb{PP}}$ (pointer programs) and its respective semantics $\mathcal{S}_{\mathbb{PP}}$ (see Section 4.2), we can define a verification functor $H_{\mathbb{PP}}$ as follows. On the objects $X \in \mathcal{S}_{\mathbb{PP}}$ we let $H_{\mathbb{PP}}(X)$ be a pre-ordered set of first-order formulas over the language of pointer programs. We assume the language has enough primitives to describe the weakest pre-condition of programs (i.e. Cook expressive), but without abort as an atomic formula, since we want abort not to be expressible in the language. The ordering $P \sqsubseteq R$ on the elements of $H_{\mathbb{PP}}(X)$ is again taken to be $P \to R$ in the fixed theory. For each pointer program A we let $H_{\mathbb{PP}}([\![A]\!])$ be the following monotone relation

$$\cfrac{\cfrac{\cfrac{\cfrac{(\nabla \in \mathcal{L}_0^b)}{\{\langle P, P\rangle\}\ \nabla\ \{P\}} \qquad \cfrac{\cfrac{(\Delta_b \in \mathcal{L}_0^b)}{\{P\}\ \Delta_b\ \{\langle P \wedge \neg b, P \wedge b\rangle\}} \quad \cfrac{\cfrac{(\mathsf{id} \in \mathcal{L}_0^b)}{\{P \wedge \neg b\}\ \mathsf{id}\ \{P \wedge \neg b\}} \quad \{\mathbf{P} \wedge \mathbf{b}\}\ \mathbf{A}\ \{\mathbf{P}\}}{\{\langle P \wedge \neg b, P \wedge b\rangle\}\ \mathsf{Par}(\mathsf{id}, A)\ \{\langle P \wedge \neg b, P\rangle\}}}{\{P\}\ \Delta_b; \mathsf{Par}(\mathsf{id}, A)\ \{\langle P \wedge \neg b, P\rangle\}}}{\{\langle P, P\rangle\}\ \nabla; \Delta_b; \mathsf{Par}(\mathsf{id}, A)\ \{\langle P \wedge \neg b, P\rangle\}}\ (\mathsf{Fb})}{\{P\}\ \mathsf{Fb}(\nabla; \Delta_b; \mathsf{Par}(\mathsf{id}, A))\ \{P \wedge \neg b\}}\ (\mathrm{def})}{\{\mathbf{P}\}\ \mathsf{while}_{\mathbf{b}}(\mathbf{A})\ \{\mathbf{P} \wedge \neg \mathbf{b}\}}$$

Fig. 10. Derivation of Hoare's while loop rule in $\mathbf{HL}(\mathcal{L}_{\mathbb{FC}}, [\![\cdot]\!], H_{\mathbb{FC}})$

$$P\ \underline{H_{\mathbb{PP}}([\![A]\!])}\ Q\ :\equiv\ P \rightarrow \mathsf{WPC}(A, Q)$$

where $\mathsf{WPC}(A, P)$ is a formula expressing the weakest liberal pre-condition (no guarantee of termination) of program A under post-condition Q. As in Reynolds [2002], well-specified programs do not abort, since formulas in $H_{\mathbb{PP}}(X)$ never hold true for abort.

It has been shown in [O'Hearn et al. 2001; Reynolds 2002], that the weakest liberal pre-conditions for the new basic statements can be concisely expressed in *separation logic* as (see [Reynolds 2002] for notation)

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

In order to show that $H_{\mathbb{PP}}$ as defined above is a verification functor, we need a lemma similar to Lemma 4.1.

Lemma 4.3. *Let A be a block diagram such that $[\![A]\!] : X \uplus Z \rightarrow Y \uplus Z$. Assume $\vec{z}$ are all the program variables of A. Moreover, let $P \in H_{\mathbb{PP}}(X)$ and $R \in H_{\mathbb{PP}}(Y)$ be fixed formulas. If $P \rightarrow \mathsf{WPC}([\![\mathsf{Fb}(A)]\!], R)$ then $\langle P, Q\rangle \rightarrow \mathsf{WPC}([\![A]\!], \langle R, Q\rangle)$, for some formula Q.*

Proof. The proof is actually easier than that of Lemma 4.1. We construct a formula Q such that

(i) $\mathsf{WPC}([\![A]\!], \langle R, Q\rangle) \leftrightarrow \langle P', Q\rangle$, for some formula P',

and then argue that (i) implies

(ii) $\mathsf{WPC}([\![\mathsf{Fb}(A)]\!], R) \leftrightarrow P'$.

By our hypothesis $P \rightarrow \mathsf{WPC}([\![\mathsf{Fb}(A)]\!], R)$ it will then follow that P implies P', as desired. The formula Q is essentially the weakest loop invariant.

Construction of Q. Given the block diagram A we build a new block diagram $A' : X \uplus Z \uplus Z \rightarrow Y \uplus Z$ where the internal states of Z can be observed, even after

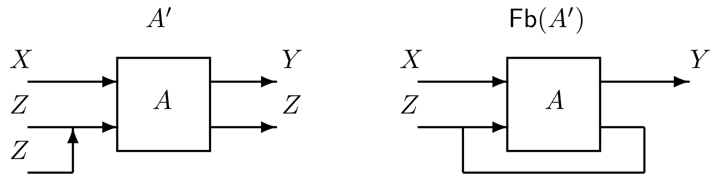


Fig. 11. Cook's construction (backward reasoning)

the feedback is applied. Let $A' :\equiv \mathsf{Par}(\mathsf{id}, \nabla); A$ (see Figure 11) Notice that the block diagram $\mathsf{Fb}^{Z}_{X \uplus Z, Y}(A')$ behaves in the same way as $\mathsf{Fb}^{Z}_{X,Y}(A)$ except that the block diagram $\mathsf{Fb}(A')$ might have new inputs fed into the internal loop through the additional input wire. Let $\langle Q_0, Q_1 \rangle :\equiv \mathsf{WPC}(\llbracket \mathsf{Fb}(A') \rrbracket, R)$, and define $Q[\vec{z}] :\equiv Q_1[\vec{z}]$.

Proof of (i). Let $\langle P', Q' \rangle :\equiv \mathsf{WPC}(\llbracket A \rrbracket, \langle R, Q \rangle)$. We must show that $Q'[\vec{z}] \Leftrightarrow Q[\vec{z}]$. Unwinding the definition of $Q[\vec{z}]$, using the definition of weakest pre condition, we obtain that either program $\mathsf{Fb}(A')$ diverges on input $\mathsf{inr}\vec{z}$ or

$$\exists \rho_1, \ldots, \rho_{n-1} \exists \rho_n \in R((\mathsf{inr} z \, A \, \mathsf{inr} \rho_1) \wedge \textstyle\bigwedge_{i=1}^{n-2} (\mathsf{inr} \rho_i \, A \, \mathsf{inr} \rho_{i+1}) \wedge (\mathsf{inr} \rho_{n-1} \, A \, \mathsf{inl} \rho_n)).$$

On the other hand, unwinding the definition of $Q'[\vec{z}]$ gives

$$\exists \rho_n \in R(\mathsf{inr} z \, A \, \mathsf{inl} \rho_n) \vee \exists \rho_0 \in Q(\mathsf{inr} \vec{z} \, A \, \mathsf{inr} \rho_0),$$

which can be seen as equivalent ways of saying that starting the loop $\mathsf{Fb}(A)$ with internal state $\vec{z}$ if the loop terminates the final state has property R.

Proof of (ii). Similarly, defining $P'' :\equiv \mathsf{WPC}(\mathsf{Tr}(\llbracket A \rrbracket), R)$, simply by looking at the definitions of P' and P'' we can see that $P'' \leftrightarrow P'$, as both characterise the initial states for which a run will terminate and the last state is in R . □

Also, similarly to Theorem 4.2, we have:

THEOREM 4.4. *$H_{\mathbb{PP}} : \mathcal{S}_{\mathbb{FC}} \to \mathcal{H}$, as defined above, is a verification functor for morphisms arising from block diagrams in $\mathcal{L}_{\mathbb{FC}}$.*

Proof. By Lemma 4.3, it remains to be shown that whenever

(i) $\langle P, Q \rangle \to \mathsf{WPC}(\llbracket A \rrbracket, \langle R, Q \rangle)$,

for a formula Q, then $P \to \mathsf{WPC}(\llbracket \mathsf{Fb}(A) \rrbracket, R)$. Assume (i) and $P(\rho)$, for some store value ρ. We must show $\mathsf{WPC}(\llbracket \mathsf{Fb}(A) \rrbracket, R)(\rho)$. If $\mathsf{Fb}(A)$ does not terminate on input ρ the result is trivial. Taking ρ as an initial state, assume the program $\mathsf{Fb}(A)$ terminates with sequence of internal states $\rho_1, \ldots, \rho_n, \rho'$. By a simple induction, using (i) we have that each $\rho_i \in Q$ and $\rho' \in R$. So, $\mathsf{WPC}(\llbracket \mathsf{Fb}(A) \rrbracket, R)(\rho)$. □

The system $\mathbf{HL}(\mathcal{L}_{\mathbb{PP}}, \llbracket \cdot \rrbracket, H_{\mathbb{PP}})$, which we then obtain from our abstract approach is basically the one presented in Reynolds [2002], for *global backward reasoning*, using the logical theory of bunched implications as an 'oracle' for the consequence rule.

4.5 Hoare logic for running time analysis and termination

Finally, we conclude this section with a new Hoare logic for *running time* analysis, and hence *termination*, of programs over the language $\mathcal{L}_{\mathbb{FC}}$ (this can easily be extended to $\mathcal{L}_{\mathbb{PP}}$ as well). This Hoare logic will be obtained from a verification functor $H^T_{\mathbb{FC}} : \mathcal{S}^T_{\mathbb{FC}} \to \mathcal{H}$ which we now define. To each object $n\mathsf{Store}$ of $\mathcal{S}^T_{\mathbb{FC}}$ the functor $H^T_{\mathbb{FC}}$ associates the pre-ordered set $\mathsf{Clock}^n :\equiv n\mathsf{Store} \to \mathbb{N}^\infty$, ordered pointwise (using the ordering of $\mathbb{N}^\infty$). The reason for the notation Clock^n is that $n\mathsf{Store} \to \mathbb{N}^\infty$ is isomorphic to

$$(\mathsf{Store} \to \mathbb{N}^\infty) \times \ldots \times (\mathsf{Store} \to \mathbb{N}^\infty),$$

so $H^T_{\mathbb{FC}}$ is monoidal on objects. We now define $H^T_{\mathbb{FC}}$ on the morphisms of $\mathcal{S}_{\mathbb{FC}}$. Given a flowchart A (assume $[\![A]\!] : (n\mathsf{Store} \to m\mathsf{Store}) \times (n\mathsf{Store} \to \mathbb{N}^\infty))$ and P, Q elements of Clock^n and Clock^m, respectively, we define the following monotone relation

$$P \; \underline{H^T_{\mathbb{FC}}([\![A]\!])} \; Q \; :\equiv P \geq \mathsf{RRT}(A, Q)$$

where $\mathsf{RRT}(A, Q) \in \mathsf{Clock}^n$ ($\equiv n\mathsf{Store} \to \mathbb{N}^\infty$) calculates the *relative running time* of A with respect to Q. The precise definition of RRT is given as follows:

$$\mathsf{RRT}(A, Q) :\equiv \lambda\rho. \begin{cases} \infty & \text{if } A(\rho) \uparrow \\ \mathsf{RunTime}(A, \rho) + Q([\![A]\!](\rho)) & \text{otherwise,} \end{cases}$$

where $\mathsf{RunTime}(A, \rho)$ is stored semantically as the second component of $[\![A]\!]$. Therefore, once again the use of the syntactic program A in the definition of the relation $P \; \underline{H^T_{\mathbb{FC}}([\![A]\!])} \; Q$ is just an abbreviation, since all the information we use from A is contained in its semantics $[\![A]\!]$.

Finally, note that for basic programs, RRT can be calculated explicitly as

$$\begin{array}{lll} \mathsf{RRT}(\mathsf{id}, Q) & :\equiv & Q \\ \mathsf{RRT}(x := t, Q) & :\equiv & Q[t/x] + 1 \\ \mathsf{RRT}(\Delta_b, \langle P, Q\rangle) & :\equiv & (b)?(P, Q) + 1 \\ \mathsf{RRT}(\mathsf{twist}, \langle P, Q\rangle) & :\equiv & \langle Q, P\rangle \\ \mathsf{RRT}(\nabla, Q) & :\equiv & \langle Q + 1, Q + 1\rangle \end{array}$$

where $(b)?(P, Q)$ is an abbreviation for "if b then P else Q".

As usual, in order to obtain a completeness result we assume that the language over which we are defining our pre- and post-conditions is rich enough to express RRT. Given that $\mathsf{RRT}(A, 0)(\rho) = \infty$ implies the program A does not terminate on input ρ, we can conclude that any expressive language will need to include non-computable functions. This is to be expected, since an expressive language where each function is computable would in this case allow us to solve the halting problem.

As in Definition 3.3, the functor $H^T_{\mathbb{FC}}$ gives rise to a Hoare triple $\{P\} \; A \; \{Q\}$. The relation $\{P\} \; A \; \{Q\}$ holds if and only if on each input ρ such that $P(\rho) \neq \infty$ then $Q \neq \infty$ and A on ρ runs in time $P(\rho) - Q([\![A]\!](\rho))$ (this in particular implies that the program terminates).

Theorem 4.5. *$H^T_{\mathbb{FC}} : \mathcal{S}_{\mathbb{FC}} \to \mathcal{H}$ as defined above is a verification functor.*

Proof. First of all, it is easy to check that $H^T_{\mathbb{FC}}$ indeed maps objects and morphisms of $\mathcal{S}_{\mathbb{FC}}$ into pre-orders and monotone relations (objects and morphisms of

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\Delta_b \in \mathcal{L}_0^b)}{\{(b)?(P,Q)+1\}\ \Delta_b\ \{\langle P,Q\rangle\}} \quad \dfrac{\{\mathbf{P}\}\ \mathbf{A}\ \{(\mathbf{b})?(\mathbf{P},\mathbf{Q})+\mathbf{1}\}}{\{\langle P,Q\rangle\}\ \mathsf{Par}(\mathsf{id},A)\ \{\langle P,(b)?(P,Q)+1\rangle\}}}{\{(b)?(P,Q)+1\}\ \Delta_b;\mathsf{Par}(\mathsf{id},A)\ \{\langle P,(b)?(P,Q)+1\rangle\}}}{\{\langle (b)?(P,Q)+1,(b)?(P,Q)+1\rangle\}\ \nabla;\Delta_b;\mathsf{Par}(\mathsf{id},A)\ \{\langle P,(b)?(P,Q)+1\rangle\}}\ (\mathsf{Fb})}{\{(b)?(P,Q)+1\}\ \mathsf{Fb}(\nabla;\Delta_b;\mathsf{Par}(\mathsf{id},A))\ \{P\}}\ (\mathrm{def})}{\{(\mathbf{b})?(\mathbf{P},\mathbf{Q})+\mathbf{1}\}\ \mathsf{while}_{\mathbf{b}}(\mathbf{A})\ \{\mathbf{P}\}}
$$

Fig. 12. While loop rule for running time

$\mathcal{H}$). $H^T_{\mathbb{FC}}$ also trivially satisfies conditions (SC1) and (SC2), i.e. $H^T_{\mathbb{FC}}$ is a monoidal functor. One direction of condition (SC3) is similar to the proof of Lemma 4.1. Given a flowchart A we construct a new flowchart A' (see Figure 11) in such way that $\mathsf{RRT}(\mathsf{Fb}(A'), R)$ gives us the (greatest) fixed point of A. The other direction of condition (SC3) is similar to the proof of Theorem 4.2. □

Therefore, this gives rise to a Hoare logic system $\mathbf{HL}(\mathcal{L}_{\mathbb{FC}}, [\![\cdot]\!], H^T_{\mathbb{FC}})$ which can be used to prove upper bounds on the running time of a program. As a simple example, consider the program calculating the factorial of y (on variable x) described in Section 2 (Figure 1):

$$
\dfrac{\{3y+2\}\ x := 1\ \{3y+1\} \qquad \dfrac{\dfrac{\{3y\}\ x := xy\ \{3y-1\} \quad \{3y-1\}\ y := y-1\ \{3y+1\}}{\{3y\}\ x := xy; y := y-1\ \{3y+1\}}}{\{3y+1\}\ \mathsf{while}_b(x := xy; y := y-1)\ \{3y\}}}{\{3y+2\}\ (x := 1); \mathsf{while}_b(x := xy; y := y-1)\ \{3y\}}
$$

where $b \equiv (y \geq 1)$. Given that $y = 0$ at the end, we have a proof of

$$\{3y+2\}\ (x := 1); \mathsf{while}_b(x := xy; y := y-1)\ \{0\},$$

which gives an upper bound of $3y+2$ steps on the running time of the program.

5. VERIFICATION FUNCTORS FOR NETWORKS

We will now apply our approach to Hoare logic to systems described by network diagrams that give a visual representation for sets of equations. An important motivating example is the notion of signal flow graph in control theory which forms the basis of systems such as Simulink. Network diagrams of this sort can be given several different formal semantics, e.g., in terms of differential equations or in terms of recurrence relations.

To apply our approach to network diagrams, our starting point is again the category of sets and relations, but now using the monoidal structure given by the cartesian product. The *cartesian trace*, $\mathsf{Tr}(r)$, of a relation $r : X \times Z \to Y \times Z$ is as defined in equation (1). In a network diagram, r above will be thought of as the input-output relation of a subsystem, with z representing a control signal that is fed back from output to input to form the overall system. The input-output relation of the overall system is then the relation between X and Y determined by solving the system of equations denoted by r for the value of the control signal, z.

Algebraic structure will be important to us. Fortunately, the cartesian trace gives rise to a wide range of interesting TMCs with nice algebraic properties. In the sequel, we will first show that the cartesian trace applies to many interesting categories of relations enjoying useful algebraic or other structure. We then present a diagrammatic language for stream circuits and give it a semantics in terms of an algebraic TMC $\mathcal{S}_{\mathbb{SC}}$ of relations representing a formal abstraction of the differential equation semantics. We use a syntactic criterion to identify a useful class of "valid circuits", whose semantic values are total functions. The subcategory formed by these circuits admits a sound verification functor and so this gives a sound Hoare logic for the valid circuits.

5.1 Traced Monoidal Categories of Relations

In this section, we give a general way of constructing TMCs whose morphisms are relations. The resulting TMCs will often enjoy good algebraic, analytic or gemoetrical properties. For simplicity and brevity, we formulate the general construction in terms of concrete categories. As we remark at the end of this section, the construction can in fact be generalised further but we do not need the extra generality in the present work.

So let C be any concrete category, so that each object X of C has an associated *underlying set* (also written X by abuse of notation) and each C-morphism between C-objects X and Y is a function from X to Y, morphisms being composed via functional composition. Let us assume that *(i)* set-theoretic finite products and equalizers[7] also serve as finite products and equalizers in C, and *(ii)*, if f is a C-morphism then the graph of f, as a subset of $X \times Y$, is the range of some C-morphism, i.e., there is a C-object U and a C-morphism $u : U \to X \times Y$, with $\mathsf{Graph}(f) = \mathsf{ran}(u)$. These assumptions hold, for example, for any concrete category defined by a set of operations and equational laws, such as any of the usual algebraic categories: groups, vector spaces over a field, modules over a ring etc. In fact, for these categories, the converse of assumption *(ii)* holds: a function is a C-morphism iff its graph is the range of a C-morphism. Under these assumptions, define a *C-relation* between C-objects X and Y to be any relation, r, which is given,as a subset of $X \times Y$, by the range of a C-morphism, i.e., $r = \mathsf{ran}(u)$, for some C-object U and C-morphism $u : U \to X \times Y$. We then define the category $\mathsf{Rel}(C)$ to have the same objects as C and to have as morphisms between X and Y the set of all C-relations between X and Y, morphisms being composed by relational composition.

THEOREM 5.1. *Under the above assumptions on the concrete category C, $\mathsf{Rel}(C)$ together with cartesian product and cartesian trace forms a TMC having C as a subcategory. If, moreover, a function is a C-morphism iff its graph is the range of some C-morphism, then this subcategory comprises precisely those relations in*

[7]Recall that in any category, an equalizer for two morphisms $f, g : X \to Y$ is a universal arrow $h : E \to X$ making the horizontal composites in the diagram $E \xrightarrow{h} X \overset{f,g}{\rightrightarrows} Y$ equal. In the category of sets, an equalizer of any two maps $f, g : X \to Y$ is given by the inclusion of the subset $\{x : X \mid f(x) = g(x)\}$, which we will call *the* equalizer of f and g.

$\mathsf{Rel}(C)$ *which are set-theoretic functions.*

Proof. We must first verify that $\mathsf{Rel}(C)$ actually is a category. (As discussed below, this follows from more general results but we give a proof here for the convenience of readers who are only concerned with the case of concrete categories that we need in the sequel.) By assumption *(i)* if X is any C-object, then the diagonal function $\delta = x \mapsto \langle x, x\rangle$ is a C-morphism from X to $X \times X$ and $\mathsf{ran}(\delta)$ is the identity relation on X which is therefore a $\mathsf{Rel}(C)$-morphism acting as a two-sided identity for relational composition. Given $\mathsf{Rel}(C)$-morphisms $r : X \to Y$ and $s : Y \to Z$, there are C-objects U and V and C-morphisms $u : U \to X \times Y$ and $v : V \to Y \times Z$ with $r = \mathsf{ran}(u)$ and $s = \mathsf{ran}(v)$. Let us write $\pi_i : X_1 \times X_2 \to X_i$ for the projections of a binary product onto its factors and write π_{ij} for the composite $(\pi_i; \pi_j)$. Our data provide everything except the object E and the dotted morphisms in the following diagram.

$$\begin{tikzcd}
E \arrow[r, dotted, "i"] \arrow[rrd, dotted, "e"'] & U \times V \arrow[r, "u \times v"] & (X \times Y) \times (Y \times Z) \arrow[r, shift left, "\pi_{12}"] \arrow[r, shift right, "\pi_{21}"'] \arrow[d, "\pi_1 \times \pi_2"] & Y \\
 & & X \times Z &
\end{tikzcd}$$

If we take $i : E \to U$ to be the equalizer of the two horizontal composites from $U \times V$ to Y and let $e : E \to X \times Z$ be the composite $(i; u \times v; \pi_1 \times \pi_2)$ then e is a C-morphism and $\mathsf{ran}(e) = (r; s)$. Thus $\mathsf{Rel}(C)$ has identities and is closed under relational composition and hence forms a category. It is easy to verify that the cartesian product makes $\mathsf{Rel}(C)$ into a symmetric monoidal category and, using assumption *(ii)*, that C forms a subcategory, and that this subcategory coincides with the functional relations in $\mathsf{Rel}(C)$ under the stated condition. So it remains to show $\mathsf{Rel}(C)$ is closed under cartesian trace. So assume $r : X \times Z \to Y \times Z$ is a $\mathsf{Rel}(C)$-morphism, so that there is a C-object U and a C-morphism $u : U \to (X \times Z) \times (Y \times Z)$ with $r = \mathsf{ran}(u)$. This gives everything except the object E and the dotted morphisms in the following diagram.

$$\begin{tikzcd}
E \arrow[r, dotted, "i"] \arrow[rrd, dotted, "e"'] & U \arrow[r, "u"] & (X \times Z) \times (Y \times Z) \arrow[r, shift left, "\pi_{12}"] \arrow[r, shift right, "\pi_{22}"'] \arrow[d, "\pi_1 \times \pi_1"] & Z \\
 & & X \times Y &
\end{tikzcd}$$

If we take $i : E \to U$ to be the equalizer of the two horizontal composites from U to Z and let $e : E \to X \times Y$ be the composite $(i; u; \pi_1 \times \pi_1)$ then $\mathsf{Tr}(r) = \mathsf{ran}(e)$ so the cartesian trace is a $\mathsf{Rel}(C)$-morphism and the proof is complete. □

As an aside, we note that this theorem offers an alternative to the partial traced monoidal categories that have been considered by several researchers, e.g., see [Haghverdi and Scott 2005]. Rather than a partial trace on C, it gives a total trace on $\mathsf{Rel}(C)$ which contains C as a subcategory. Specific examples of partial

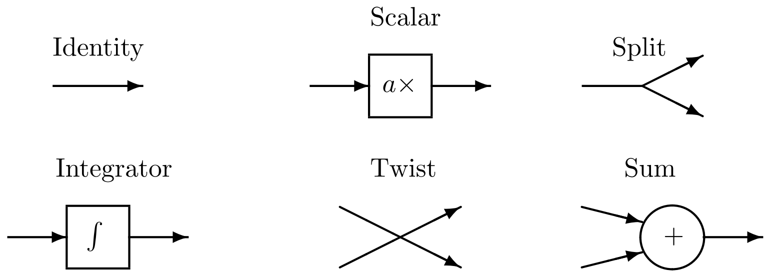


Fig. 13. Basic stream circuits.

TMCs given by Haghverdi and Scott [2005] are the category of vector spaces and the category of complete metric spaces with non-expansive continuous functions as morphisms. Both these examples are covered by the above theorem and without the restriction to finite dimensions in the case of vector spaces. The theorem also applies to both Hilbert spaces and Banach spaces with bounded operators as morphisms and to compact Hausdorff topological spaces with continuous maps.

The trace for the disjoint union monoidal structure of section 4 was motivated by consideration of iterative processes. Nonetheless it turns out to be strictly dual to the cartesian trace: the dual of the construction in the above theorem also goes through using coequalizers, coproducts and an appropriate notion of cograph. Constructions of this sort have been considered in connection with universal algebra, see [Hutchinson 1994] for details and references. The construction of the category of relations given above generalises to regular categories, see [Freyd and Scedrov 1990], and, even more generally, to categories equipped with a factorisation system, see [Pavlovic 1995]. Our method of constructing the trace also generalises giving TMC structures for these more general categories of relations.

5.2 The programming language: stream circuits

In [Boulton et al. 2003], a Hoare logic was presented for the frequency analysis of linear control systems with feedback, modelled as linear differential equations, an approach which we generalised and systematised in [Arthan et al. 2007]. In the present paper, we will model linear differential equations as stream circuits (see [Rutten 2004]). With Theorem 5.1 in mind, this model fits nicely both with the algebraic viewpoint of [Arthan et al. 2007] and with the present framework.

Our language of stream circuits comprises finite diagrams, made up from the basic circuits shown in Figure 13 (where a is a real number-valued parameter). New circuits are formed from old by connecting outputs (arrow heads) to inputs (arrow tails). An output of a circuit may be connected to an input of the same circuit, so we can create feedback loops.

Definition 5.2 *([Rutten 2004]).* We say circuit is *valid* if every closed path in the circuit passes through at least one integrator (or "register" in Rutten's terminology).

5.3 The semantics

In our semantics, stream circuits denote differential equations. As demonstrated in [Escardó and Pavlovic 1998], many computational aspects of mathematical analysis can be developed in a co-algebraic setting using infinite streams of real numbers, an analytic function f being modelled by the infinite stream $[f(0), f'(0), f''(0), \ldots]$ of its partial derivatives. For many purposes we may abstract away issues of convergence, and view infinite streams rather than functions as the main objects of interest. Following [Rutten 2004], we can view a stream circuit as a system of equations on streams; then, using appropriate conventions for initial conditions, a system of equations on streams represents a differential equation as in [Rutten 2005]: for example, the differential equation $f = f'$, or equivalently $f = (\int f) + c$, corresponds to the stream equation $f = c :: \mathsf{tl}(f)$.

To turn these ideas into a formal semantics for stream circuits, let Σ denote the set of streams of real numbers, i.e. all functions $\mathbb{N} \to \mathbb{R}$. Let the objects of $\mathcal{S}_{\mathbb{SC}}$ be all Σ^n, for natural number n. Consider the following basic morphisms between the objects of $\mathcal{S}_{\mathbb{SC}}$:

—*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*, $(+) : \Sigma \times \Sigma \to \Sigma$
$(+)\langle \sigma, \sigma' \rangle :\equiv [\sigma_0 + \sigma'_0, \sigma_1 + \sigma'_1, \ldots]$

—*Integrator*, $\mathsf{R} : \Sigma \to \Sigma$
$\mathsf{R}(\sigma) :\equiv [0, \sigma_0, \sigma_1, \ldots]$

—*Twist*, $\mathsf{t} : \Sigma \times \Sigma \to \Sigma \times \Sigma$
$\mathsf{t}\langle \sigma, \sigma' \rangle :\equiv \langle \sigma', \sigma \rangle$

The morphisms of $\mathcal{S}_{\mathbb{SC}}$ are the *relations* obtained from the basic morphisms above viewed as relations (via their graph) and closing this set under relational composition, cartesian product of relations, and the trace for the cartesian product (1). By construction, $\mathcal{S}_{\mathbb{SC}}$ is the smallest TMC of relations containing the finite products of Σ the and above basic morphisms.

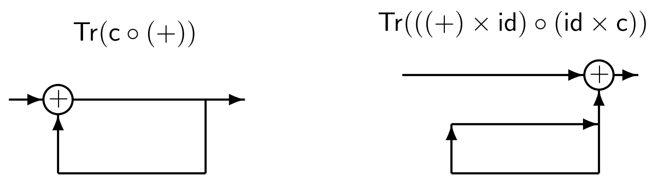
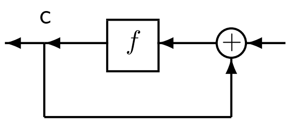


Fig. 14. Stream circuits not defining functions

The diagrams in Figure 14 illustrate two stream circuits that define relations that are not total functions. In the circuit on the left there is no fixed point for the input stream $\sigma_a :\equiv [a, a, \ldots]$ unless $a = 0$, so the circuit defines a partial function with domain $\{0\}$. 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 any output stream, so the circuit represents the "chaotic" relation $\Sigma \times \Sigma$. The notion of valid circuit gives a syntactic criterion for avoiding this pathological behaviour: while $\mathcal{S}_{\mathbb{SC}}$ is a category of relations that are partial and one-to-many in general, valid circuits denote *total*

$\mathsf{fdback}(f) :\equiv \mathsf{Tr}((\mathsf{c}) \circ (f) \circ (+))$ $\mathsf{sum}(f, g) :\equiv (+) \circ (g \uplus f) \circ (\mathsf{c})$



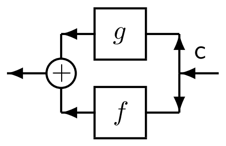


Fig. 15. Feedback and summation (semantically) in $\mathcal{S}_{\mathbb{SC}}$

functions, i.e. a valid circuit denotes a system of equations whose solutions always exist and are unique (as we shall prove shortly).

In fact, $\mathcal{S}_{\mathbb{SC}}$ has several important subcategories obtained by restricting the sets of morphisms: $\mathcal{S}^F_{\mathbb{SC}}$ in which the morphisms are the total functional relations, and $\mathcal{S}^V_{\mathbb{SC}}$ comprising the denotations of valid stream circuits. To understand these subcategories it is helpful to consider some additional algebraic structure. Let $\mathsf{R} = \mathbb{R}[[x]]$ be the ring of formal power series with real coefficients, e.g., see Mac Lane and Birkhoff [1999], section VIII.9. Thus elements of R are formal expressions $f_0 + f_1x + f_2x^2 + \ldots + f_mx^m + \ldots$ where the coefficients f_i are real numbers. Note that a univariate polynomial, $p_0 + p_1x + p_2x^2 + \ldots + p_mx^m$, over $\mathbb{R}$ may be viewed as a formal power series in which all but a finite number of the coefficients are zero. Formal power series are added by adding corresponding coefficients and multiplied using the convolution product (generalising multiplication of polynomials), i.e., if $f = f_0 + f_1x + \ldots + f_mx^m + \ldots$ and $g = g_0 + g_1x + \ldots + g_mx^m \ldots$, the coefficients of the sum $f + g$ and the product fg are given by:

$$\begin{aligned} (f+g)_m &= f_m + g_m \\ (fg)_m &= f_0g_m + f_1g_{m-1} + \cdots f_{m-1}g_1 + f_mg_0 \end{aligned}$$

Under these operations R forms a commutative ring with $0 = 0 + 0x + \ldots$ and $1 = 1 + 0x + \ldots$. If we identify a stream σ with the formal power series $\sigma_0 + \sigma_1x^1 + \ldots + \sigma_mx^m + \ldots$, then R acts on the objects of $\mathcal{S}_{\mathbb{SC}}$ by multiplication, i.e., for $({}^1\sigma, \ldots, {}^n\sigma) \in \Sigma^n$ and $f \in \mathsf{R}$, we define:

$$({}^1\sigma, \ldots, {}^n\sigma)f = ({}^1\sigma f, \ldots, {}^n\sigma f)$$

Under this action, the objects of $\mathcal{S}_{\mathbb{SC}}$ become R-modules[8].

If $f \in \mathsf{R}$ and $f_0 \neq 0$, then f has a multiplicative inverse f^{-1} whose coefficients are given recursively by

$$(f^{-1})_0 = 1/f_0$$

[8]Recall that the notion of R-module is the generalisation to arbitrary rings of the notion of vector space over a field: an R-module is an additive group M such that R acts on M as a ring of linear operators. I.e., writing $x \mapsto xf$ for the action of $f \in R$ on $x \in M$, one has $(x + y)f = xf + yf$, $x1 = x$ and $x(fg) = (xf)g$. R acts itself by right multiplication and so becomes an R-module whose submodules are called *ideals*, i.e., an ideal I is an additive subgroup of R such that, for any $a \in R$, $Ia = \{xa \mid x \in I\} \subseteq I$. See, e.g., [Mac Lane and Birkhoff 1999] chapter V for the basics of module theory.

$$(f^{-1})_{m+1} = (f_1(f^{-1})_m + f_2(f^{-1})_{m-1} + \dots + f_{m+1}(f^{-1})_0)/f_0$$

as is readily checked. Clearly, any non-zero $f \in \mathsf{R}$ may be written uniquely in the form $f = x^m g$ where g is invertible, m being the *order* of f, i.e., the smallest m such that $f_m \neq 0$. It follows that the non-zero ideals of R are precisely the principal ideals generated by the powers of x, i.e., $I \neq \{0\}$ is an ideal of R iff $I = x^m\mathsf{R}$ for some m.

It is easy to check that the basic morphisms of $\mathcal{S}_{\mathbb{SC}}$ are R-module homomorphisms, so for example, if $\sigma \in \Sigma^n$ and $f \in \mathsf{R}$, one has $\mathsf{c}(\sigma f) = \mathsf{c}(\sigma)f$. Noting that the approach of [Arthan et al. 2007] generalises to R-modules[9], we find that each morphism of $\mathcal{S}_{\mathbb{SC}}$ is actually an additive relation, i.e., each morphism between objects L and M of $\mathcal{S}_{\mathbb{SC}}$ is a relation whose graph is a submodule of the product module $L \times M$. Within $\mathcal{S}_{\mathbb{SC}}$, $\mathcal{S}^T_{\mathbb{SC}}$ comprises the subcategory of morphisms which happen to be total functions. In the terminology of Section 5.1, the methods of [Arthan et al. 2007, section 5] show that $\mathcal{S}_{\mathbb{SC}}$ is the category of $\mathcal{S}^T_{\mathbb{SC}}$-relations: $\mathcal{S}_{\mathbb{SC}} = \mathsf{Rel}(\mathcal{S}^T_{\mathbb{SC}})$

Let us exploit this algebraic structure to investigate the cartesian trace applied to a functional morphism in $\mathcal{S}_{\mathbb{SC}}$. Using the vanishing law for the trace, the trace of a general morphism $\Sigma^m \times \Sigma^k \to \Sigma^n \times \Sigma^k$ can be calculated by iterating the trace on morphisms $\Sigma^{m'} \times \Sigma \to \Sigma^{n'} \times \Sigma$, so it suffices to consider the case $k = 1$. So let $F : \Sigma^m \times \Sigma \to \Sigma^n \times \Sigma$ be a functional morphism, i.e., a homomorphism of R-modules. Just as with vector spaces, we can represent F as a 2×2 block matrix, i.e., there are unique R-module homomorphisms, $A : \Sigma^m \to \Sigma^n$, $B : \Sigma \to \Sigma^n$, $C : \Sigma^m \to \Sigma$ and $D : \Sigma \to \Sigma$, such that $\langle \alpha, \sigma \rangle F = \langle \beta, \tau \rangle$ where:

$$\begin{aligned}\beta &= \alpha A + \sigma B\\ \tau &= \alpha C + \sigma D\end{aligned}$$

D here is a 1×1 matrix, i.e., under the identification of Σ with R, D is given by multiplication by some element of R say $D = (d)$. The following lemma says that under appropriate assumptions on d, the trace of F is a total function. The ideal I in the theorem may be thought of as a measure of the precision of an approximation σ to the actual fixed point σ'. For the simple existence of the fixed point, take $I = \mathsf{R}$ and pick an arbitrary σ.

LEMMA 5.3. *Assume $F : \Sigma^m \times \Sigma \to \Sigma^n \times \Sigma$ is defined by $\langle \alpha, \sigma \rangle F = \langle \beta, \tau \rangle$ where:*

$$\begin{aligned}\beta &= \alpha A + \sigma B\\ \tau &= \alpha C + \sigma D\end{aligned}$$

for some $A : \Sigma^m \to \Sigma^n$, $B : \Sigma \to \Sigma^n$, $C : \Sigma^m \to \Sigma$ and $D : \Sigma \to \Sigma$, where D is given by the 1×1 matrix (d) where $d \in \mathsf{R}$ is either 0 or has positive order, i.e.,

[9]The only difficulty is with theorem 5.8 of [Arthan et al. 2007] where the proof in the published paper uses properties of bases to define relational inverse in terms of the feedback loop operator. However, a basis-free construction can be given: in the notation of [Arthan et al. 2007], if $r : V \leftrightarrow W$, r^{-1} is the composite:

$$\begin{aligned}(0, 1_W) &: W \to V \times W;\\ \mathsf{loop}(1_{V\times W}, 1_{V\times W} - (\pi_1; r; (0, 1_W))) &: V \times W \leftrightarrow V \times W;\\ \pi_1 &: V \times W \to V\end{aligned}$$

$d = xf$ for some $f \in \mathsf{R}$. Let I be any ideal in R and assume $(\alpha, \sigma)F = (\beta, \tau)$ for some α, β, σ and τ where $\sigma - \tau \in I$, then there exist unique β' and σ' such that $(\alpha, \sigma')F = (\beta', \sigma')$ and then one has $\sigma' - \sigma \in I$.

Proof. For uniqueness, if β' and σ' satisfy

$$\begin{aligned} \beta' &= \alpha A + \sigma' B \\ \sigma' &= \alpha C + \sigma' D \end{aligned}$$

then, as $D = (d)$ and $1 - d$ is invertible, the second equation gives $\sigma' = \alpha C/(1 - d)$ fixing σ' and then the first equation fixes β'. For the existence, if $I = \{0\}$, then $\tau = \sigma$ and we may take $\beta' = \beta$ and $\sigma' = \sigma$, otherwise define sequences ${}^m\sigma$ and ${}^m\tau$ of elements of R as follows:

$$\begin{aligned} {}^0\sigma &= \sigma, & {}^0\tau &= \tau = \alpha C + {}^0\sigma d \\ {}^{m+1}\sigma &= {}^m\tau, & {}^{m+1}\tau &= \alpha C + {}^{m+1}\sigma d \end{aligned}$$

We claim that ${}^m\sigma - {}^m\tau \in Id^m$ for each m. This is true by assumption for $m = 0$, so assume inductively that it holds for some m, we then have

$$\begin{aligned} {}^{m+1}\sigma - {}^{m+1}\tau &= {}^m\tau - (\alpha C + {}^m\tau d) \\ &= {}^m\tau(1 - d) - \alpha C \\ &= {}^m\tau(1 - d) - {}^m\tau + {}^m\sigma d \\ &= ({}^m\sigma - {}^m\tau)d \in (Id^m)d = Id^{m+1} \end{aligned}$$

which proves the claim. Note that as $d = xf$ for some non-zero f, ${}^m\sigma_k = {}^{m+1}\sigma_k$ for $m > k$ since we have ${}^{m+1}\sigma - {}^m\sigma = {}^m\tau - {}^m\sigma \in Id^m = If^m x^m$. It follows that if we put $\sigma'_k = {}^{k+1}\sigma_k$, then $\sigma' - {}^m\sigma \in Id^m$ for all m and if we put $\tau' = \alpha C + \sigma' D$ then $\sigma' - \tau' \in Id^m$ for all m, but the intersection of the ideals Id^m is the zero ideal, so $\sigma' - \tau' = 0$, i.e., $\sigma' = \tau' = \alpha C + \sigma' D$; taking $\beta' = \alpha A + \sigma' B$ completes the proof. □

Remark: the solution in the case where $d = 0$, can also be given directly as $\sigma' = \alpha C$ and $\beta' = \alpha(A + CB)$.

Theorem 5.4. *Valid stream circuits denote total functions.*

By the lemma, it suffices to show that if a circuit denotes a function $F : \Sigma^m \times \Sigma \to \Sigma^n \times \Sigma$ given by $\langle \alpha, \sigma \rangle F = \langle \beta, \tau \rangle$ where:

$$\begin{aligned} \beta &= \alpha A + \sigma B \\ \tau &= \alpha C + \sigma D \end{aligned}$$

$D = (d)$ being a 1×1 matrix, and if every path in the circuit from the second input component to the second output component passes through an integrator, then $d = xf$ for some $f \in \mathsf{R}$. This is easy to verify by induction on the construction of the circuit, since passing a stream through an integrator corresponds to multiplying it by x. □

5.4 Hoare logic for stream circuits

We will now construct a functor $H_{\mathbb{SC}} : \mathcal{S}_{\mathbb{SC}} \to \mathcal{H}$ and show that its restriction to valid circuits is a *sound verification functor*, i.e. functor satisfying conditions S1, S2 and S3 of Section 3. Each object of $\mathcal{S}_{\mathbb{SC}}$ has the form $\Sigma^m = \Sigma \times \ldots \times \Sigma$ for some m. We define $H_{\mathbb{SC}}(\Sigma)$ to be the set of pairs (σ, I) where $\sigma \in \Sigma$ and $I \subseteq \Sigma$ corresponds to an ideal in R under the identification of Σ with R. $H_{\mathbb{SC}}(\Sigma)$ is ordered by defining $\langle \sigma, I\rangle \sqsubseteq \langle \tau, J\rangle$ iff $\sigma + I \subseteq \tau + J$. $H_{\mathbb{SC}}(\Sigma^m)$ is then defined to be $H_{\mathbb{SC}}(\Sigma)^m$ which we may identify with the set of all pairs $\langle \alpha, L\rangle$ where $\alpha \in \Sigma^m$ and L is what we may call a *coordinate* submodule of Σ^m, i.e., a submodule of the form $I_1 \times I_2 \times \ldots \times I_m$ where each I_k is an ideal of R under the identification of Σ with R. Note that the m-tuple $\langle I_1, I_2, \ldots, I_m\rangle$ can be recovered from the product $I_1 \times I_2 \times \ldots \times I_m$ since the ideals I_j are non-empty sets. The product ordering on $H_{\mathbb{SC}}(\Sigma)^m$ corresponds to taking $\langle \alpha, L\rangle \sqsubseteq \langle \beta, M\rangle$ iff $\alpha + L \subseteq \beta + M$. Thus we let $\langle \alpha, L\rangle$ correspond to the subset $\alpha + L$ of Σ^m and think of α as an estimate of a value in Σ^m with L giving the precision of the estimate. So for example, $\langle \alpha, \Sigma^m\rangle$ represents the whole set, or a maximally imprecise estimate, while $\langle \alpha, 0\rangle$ represents the singleton set $\{\alpha\}$, or an exact estimate.

On the morphisms (semantic values of stream circuits) $f : X \to Y$ in $\mathcal{S}_{\mathbb{SC}}$, we define the relation $H_{\mathbb{SC}}(f)$ as follows, where $\langle \alpha, L\rangle \in H_{\mathbb{SC}}(X)$ and $\langle \beta, M\rangle \in H_{\mathbb{SC}}(Y)$, and where $(A)r$ denotes the image of the set A under the relation r.

$$\langle \alpha, L\rangle \; \underline{H_{\mathbb{SC}}(f)} \; \langle \beta, M\rangle :\equiv \alpha + L \subseteq \mathsf{dom}(f) \wedge (\alpha + L)f \subseteq \beta + M$$

We now show that $H_{\mathbb{SC}}$ as defined above yields a sound verication functor for valid circuits, so that the associated Hoare logic is sound, but not necessarily complete[10].

THEOREM 5.5. *$H_{\mathbb{SC}} : \mathcal{S}^V_{\mathbb{SC}} \to \mathcal{H}$, as defined above, is a sound verification functor for valid circuits.*

Proof. That $H_{\mathbb{SC}}$ commutes with composition of morphisms is easy to verify, so $H_{\mathbb{SC}}$ is indeed a functor. By construction, $H_{\mathbb{SC}}(X \times Y) = H_{\mathbb{SC}}(X) \times H_{\mathbb{SC}}(Y)$ (although in our notation we are identifying $(\Sigma \times \mathbb{P}\Sigma)^m$ with a subset of $(\Sigma^m \times \mathbb{P}(\Sigma^m))$, and one may check that $H_{\mathbb{SC}}(f \times g) = H_{\mathbb{SC}}(f) \times H_{\mathbb{SC}}(g)$ when and g are morphisms, so $H_{\mathbb{SC}}$ is a monoidal functor. It remains to show that $H_{\mathbb{SC}}$ respects the trace structure. As in Theorem 5.4, it suffices to consider traces of morphisms $f : \Sigma^m \times \Sigma \to \Sigma^n \times \Sigma$. Assume that

$$\langle \alpha, L\rangle \; \underline{\mathsf{Tr}(H_{\mathbb{SC}}(f))} \; \langle \beta, M\rangle,$$

so that there is a stream σ and an ideal I such that

$$\langle\langle \alpha, \sigma\rangle, L \times I\rangle \; \underline{H_{\mathbb{SC}}(f)} \; \langle\langle \beta, \sigma\rangle, M \times I\rangle.$$

Then $\langle \alpha, \sigma\rangle \in \mathsf{dom}(f)$, so that there are γ and τ such that $\langle \alpha, \sigma\rangle f = \langle \gamma, \tau\rangle$, where $\gamma - \beta \in M$ and $\sigma - \tau \in I$. Now, just as in Theorem 5.4 the assumption that the circuit is valid lets us apply Lemma 5.3 to give unique δ and σ' such that $\langle \alpha, \sigma'\rangle f = \langle \delta, \sigma'\rangle$ such that $\delta - \gamma \in M$ and $\sigma' - \sigma \in I$, but then also $\delta - \beta \in M$ and we may check that $\langle \alpha, L\rangle \; \underline{H_{\mathbb{SC}}(\mathsf{Tr}(f))} \; \langle \beta, M\rangle$. Thus $\mathsf{Tr}(H_{\mathbb{SC}}(f)) \subseteq H_{\mathbb{SC}}(\mathsf{Tr}(f))$ completing the proof. $\square$

[10]Completeness fails because our "assertion language" is not Cook-expressive, in the sense that the strongest post-condition for an expressible pre-condition need not be expressible. E.g., consider the copying operation c with any pre-condition $\langle \alpha, L\rangle$ for $L \neq 0$.

This gives rise to a sound Hoare-logic system for reasoning about *valid* stream circuits. Notice that the rules are only sound for valid circuits. For instance, consider the circuit $\mathsf{c} \circ (+) : \Sigma \to \Sigma \times \Sigma$ used to construct the example shown on the left of Figure 14. For any α, the triple

$$\{\langle\langle \alpha, 0\rangle, \Sigma \times \Sigma\rangle\}\ \mathsf{c} \circ (+)\ \{\langle 0, \Sigma \times \Sigma\rangle\}$$

is valid, but the triple $\{\langle \alpha, \Sigma\rangle\}\ \mathsf{Tr}(\mathsf{c} \circ (+))\ \{\Sigma\}$ is only valid for $\alpha = 0$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{((+) \in \mathcal{S}_{\mathbb{FC}})}{\{\langle s,t\rangle\}\ (+)\ \{s+t\}} \quad \{\mathbf{s}+\mathbf{t}\}\ \mathbf{A}\ \{\mathbf{t}\}}{\{\langle s,t\rangle\}\ (+); A\ \{t\}}\ \text{(Seq)} \qquad \dfrac{((\mathsf{c}) \in \mathcal{S}_{\mathbb{FC}})}{\{t\}\ (\mathsf{c})\ \{\langle t,t\rangle\}}}{\{\langle s,t\rangle\}\ (+); A; (\mathsf{c})\ \{\langle t,t\rangle\}}\ \text{(Seq)}}{\{s\}\ \mathsf{Fb}((+); A; (\mathsf{c}))\ \{t\}}\ \text{(Fb)}}{\{\mathbf{s}\}\ \mathsf{fdback}(\mathbf{A})\ \{\mathbf{t}\}}\ \text{(def)}$$

Fig. 16. Derivation of feedback rule in $\mathbf{HL}(\mathcal{L}, [\![\cdot]\!], H)$

Streams can be viewed as giving the Taylor expansions of analytic functions. In this particular example, the pre- and post-conditions in the Hoare logic correspond to partial sums for the Taylor expansions. A partial sum comprising n terms corresponds to the specification $\alpha + \Sigma x^n$, where α is any stream with $\alpha_i = s_i$ for $i = 0, \ldots, n-1$. We have thus obtained from our general categorical construction a sound formal system $\mathbf{HL}(\mathcal{L}, [\![\cdot]\!], H_{\mathbb{SC}})$, for reasoning about valid stream circuits and their input-output behaviour over classes of functions with a common partial Taylor sum.

Let us define a simple (syntactic) feedback circuit $\mathsf{fdback}(A)$ as $\mathsf{Fb}((+); A; (\mathsf{c}))$, corresponding to the semantic construction shown in Figure 15. We then have the following rule, whose derivation is given in Figure 16 and is very similar to the derivation of the rule for while loops (cf. Figure 10) for while programs:

$$\dfrac{\{\mathbf{s}+\mathbf{t}\}\ \mathbf{A}\ \{\mathbf{t}\}}{\{\mathbf{s}\}\ \mathsf{fdback}(\mathbf{A})\ \{\mathbf{t}\}}$$

It should be noted that our rule has a side condition stating that $\mathsf{fdback}(A)$ is a valid circuit.

5.5 Hilbert Spaces

We have already remarked that the constructions of Section 5.1 apply to the category of Hilbert spaces and bounded operators. This gives a TMC whose objects are Hilbert spaces and whose morphisms are relations whose graphs are complete subspaces of a product space. Restricting to a specific Hilbert space Σ and its finite products Σ^m, analogues of all our results on stream circuits go through using closed discs, $I_t = \{\sigma : \Sigma \mid ||\sigma|| \leq t\}$ in place of ideals of R. The analogue of 5.3 requires D to be a contraction mapping ($||D|| = \mathsf{sup}_\sigma(||\sigma D||/||D||) < 1$) and then the fixed point can be given as an analytic limit. For the lemma to serve its purpose, the hypothesis on the approximate fixed point needs to be slightly stronger than for stream circuits, but the assumptions available where the lemma is used in Theorem 5.5 are sufficient.

The Hoare logic obtained in this way for systems where the semantic domains are Hilbert spaces way is interesting to compare with the Hoare logic for vector space semantics in general given in [Arthan et al. 2007]. In the general approach we do indeed get a very general Hoare logic but at the price of rules involving side-conditions that have a non-trivial semantic content. The approach for Hilbert spaces sketched here admits a much less general form of assertion and applies to a restricted category of spaces but those restrictions mean that it falls within the general framework of the present paper and yields rules that are much more satisfactory from a syntactic point of view.

6. CONCLUSION AND RELATED WORK

Several abstractions of the Floyd-Hoare logic can be found in the literature. In this final section we attempt to clarify the relation between the work presented here, and the previous work. Due to the vast amount of research in the area, we will only cover, however, works that we believe are closely related to ours.

Kozen's [2000] *Kleene Algebra with Test*, KAT, consists essentially of a Kleene algebra with a Boolean subalgebra. In Kozen's work Hoare triples $\{P\}\ A\ \{Q\}$ are modelled as equations $PA = PAQ$, using the multiplication available in 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 pre-order statements) there does not seem to be at the moment 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 pre-ordered sets for obtaining soundness and completeness.

Kozen's work is related to previous work on iteration theory [Bloom and Ésik 1991; Manes and Arbib 1986] and dynamic logic [Pratt 1976]. It should be stressed that all these 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 5, our approach is more general including systems with an underlying *cartesian structure* as well.

Abramsky et al. [1996] 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 $H : \mathcal{S} \to \mathcal{H}$ 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 pre-order and trace, however, allows us to prove an abstract completeness theorem, which does not seem to be the focus of [Abramsky et al. 1996].

Blass and Gurevich [2000] considered the *underlying logic of Hoare logic*. Since Cook's celebrated completeness result [Cook 1978], 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 $H_{\mathbb{FC}}(f)$ of Section 4.3 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 4.1.

Note that our treatment of infinite streams (Section 5.4) does not make use of co-induction, contrary to usual practise. The main reason is that we work with *finite prefixes* of (infinite) streams, and prove (for a given system) that an input finite prefix is related to an output finite prefix. Co-induction on the other hand is mostly used to show that a particular infinite stream has some given property.

For a given semantic mapping $[\![\cdot]\!] : \mathcal{L} \to \mathcal{S}$, verification functor $H : \mathcal{S} \to \mathcal{H}$ and program $A : X \to Y$ in $\mathcal{L}$, the binary monotone relation $\{\cdot\}\ A\ \{\cdot\}$, i.e.

$$\underline{H([\![A]\!])} \quad \subseteq \quad H(X) \times H(Y)$$

can be viewed as an order-preserving function $H(X)^{\mathrm{op}} \times H(Y) \to \mathbb{B}$, where $H(X)^{\mathrm{op}}$ denotes the pre-order $H(X)$ with the opposite ordering, and the usual ordering for the product and the booleans $\mathbb{B}$ is assumed. If we generalise pre-orders to categories, and the booleans $\mathbb{B}$ to an arbitrary category, the binary relation $\{\cdot\}\ A\ \{\cdot\}$ becomes a profunctor (cf. [Cattani and Winskel 2004]). Such profunctors also arise from a generalisation of relation, so-called *span*, used by Winskel [2005] to represent processes. A span is a diagram of the form $A \leftarrow U \to B$, where A, B and U are event structures (partial orders endowed with a binary "consistency" relation). Such a diagram is equivalent to a morphism $U \to A \times B$ as used in our definition of the category of C-relations. These observations suggest several possible links between our methods and Winskel's approach to non-determinism and concurrency.

ACKNOWLEDGMENTS

We are grateful for discussions we have had with Martin Hyland, and his suggestion of using monotone relations (rather than functions) which considerably simplified our previous work [Martin et al. 2006]. We also thank Glynn Winskel for helpful correspondence and pointers to related work. We thank Colin O'Halloran of QinetiQ and John Hall of DSTL for a discussion which stimulated our interest in applying Hoare logics to systems expressed as network diagrams. Finally, we are immensely indebted to the anonymous referees for their thorough review and the various comments which helped greatly to improve the paper.

REFERENCES

Abramsky, S., Gay, S., and Nagarajan, R. 1996. Specification structures and propositions-as-types for concurrency. In Logics for Concurrency: Structure vs. Automata, G. Birtwistle and F. Moller, Eds. *Logics for Concurrency: Structure vs. Automate*, 5–40.

Apt, K. R. 1981. Ten years of Hoare's logic: A survey – Part 1. *ACM Transactions on Programming Languages and Systems 3,* 4 (October), 431–483.

Arthan, R., Martin, U., Mathiesen, E. A., and Oliva, P. 2007. Reasoning about linear systems. In *SEFM'07*. IEEE Press.

Bainbridge, E. S. 1976. Feedback and generized logic. *Information and Control 31*, 75–96.

Berger, M., Honda, K., and Yoshida, N. 2005. A logical analysis of aliasing in imperative higher-order functions. In *ICFP 2005*. 280–293.

Blass, A. and Gurevich, Y. 2000. The underlying logic of Hoare logic. *Bull. of the Euro. Assoc. for Theoretical Computer Science 70*, 82–110.

Bloom, S. L. and Ésik, Z. 1991. Floyd-Hoare logic in iteration theories. *J. ACM 38,* 4 (October), 887–934.

Boulton, R. J., Hardy, R., and Martin, U. 2003. 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.

Cattani, G. L. and Winskel, G. 2004. Profunctors, open maps and bisimulation. Accepted for MSCS.

Cook, S. A. 1978. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput. 7,* 1 (February), 70–90.

Escardó, M. H. and Pavlovic, D. 1998. Calculus in coinductive form. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science.*

Floyd, R. W. 1967. Assigning meanings to programs. *Proc. Amer. Math. Soc. Symposia in Applied Mathematics 19*, 19–31.

Freyd, P. and Scedrov, A. 1999. *Categories, Allegories* North-Holland Mathematical Library. Vol. 39. North-Holland, Amsterdam.

Haghverdi, E. and Scott, P. 2005. Towards a typed geometry of interaction. In *CSL'05, LNCS*, L. Ong, Ed. Vol. 3634. 216–231.

Hoare, C. A. R. 1969. An axiomatic basis for computer programming. *Communications of the ACM 12,* 10 (October), 576–585.

Hutchinson, G. 1994. Relation categories and coproduct congruence categories in universal algebra. *Algebra Universalis 32*, 609–647.

Joyal, A., Street, R., and Verity, D. 1996. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society 119*, 447–468.

Kozen, D. 2000. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic (TOCL) 1,* 1, 60–76.

Mac Lane, S. 1998. *Categories for the Working Mathematician.* Graduate texts in mathematics, vol. 5. Springer, 2nd ed.

Mac Lane, S. and Birkhoff, G. 1999. *Algebra*, Third ed. AMS Chelsea Publishing.

Manes, E. G. and Arbib, M. A. 1986. *Algebraic Approaches to Program Semantics.* AKM series in theoretical computer science. Springer-Verlag, New York, NY.

Martin, U., Mathiesen, E. A., and Oliva, P. 2006. Abstract Hoare logic. In *CSL'06, LNCS.*

O'Hearn, P., Reynolds, J., and Yang, H. 2001. Local reasoning about programs that alter data structures. In *CSL'01, LNCS*, L. Fribourg, Ed. Vol. 2142. Springer-Verlag, 1–19.

Pavlović, D. 1995. Maps I: relative to a factorisation system *Journal of Pure and Applied Algebra* 99(1995). 9–34.

Pratt, V. R. 1976. Semantical considerations on Floyd-Hoare logic. In *Proc. 17th Annual IEEE Symposium on Foundations of Computer Science.* 109–121.

Reynolds, J. C. 2002. Separation logic: A logic for shared mutable data structures. In *Proc. Logic in Computer Science (LICS'2002).* 55–74.

Rutten, J. J. M. M. 2004. An application of stream calculus to signal flow graphs. *Lecture Notes in Computer Science 3188*, 276–291.

Rutten, J. J. M. M. 2005. A coinductive calculus of streams *Mathematical Structures in Computer Science* 15(2005). 93–147.

Simpson, A. K. and Plotkin, G. D. 2000. Complete axioms for categorical fixed-point operators. In *Proc. Logic in Computer Science (LICS'2000).* 30–41.

Winskel, G. 2005. Relations in concurrency. In *LICS.* IEEE Computer Society, 2–11.

Received July 2008; revised May 2009; accepted May 2009