Math. Struct. in Comp. Science (2018), *vol.* 28, *pp.* 775–799. © Cambridge University Press 2017
doi:10.1017/S096012951700007X First published online 4 April 2017

Hoare Semigroups

GEORG STRUTH

Department of Computer Science, University of Sheffield, Sheffield S1 4DP, U.K.
Email: g.struth@sheffield.ac.uk

Received 15 June 2016; revised 22 December 2016

A semigroup-based setting for developing Hoare logics and refinement calculi is introduced together with procedures for translating between verification and refinement proofs. A new Hoare logic for multirelations and two minimalist generic verification and refinement components, implemented in an interactive theorem prover, are presented as applications that benefit from this generalisation.

1. Introduction

Kleene algebra with tests (Kozen 1997) can be seen as the algebra of while programs. It provides a two-sorted signature with one carrier set for programs equipped with operations for their non-deterministic choice, sequential composition and finite iteration. A second carrier set models tests or assertions with operations for join, meet and complementation. It is well known that the rules of propositional Hoare logic (PHL) for the partial correctness of while programs – Hoare logic without assignment rules – can be derived from its axioms (Kozen 2000). A simple expansion allows axiomatising Morgan's specification statement and deriving a basic calculus for the stepwise refinement of while programs (Armstrong et al. 2016).

From these foundations, program construction and verification components for interactive theorem provers can be developed (Armstrong et al. 2016, 2013a; Pous 2013). The algebraic axioms can be linked with denotational semantics of the program store based on binary relations or program traces by formal soundness proofs. Assignment laws can then be derived in this concrete semantics and programs constructed and verified directly within it. A main feature of this method is that the control flow level of programs, as modelled by the algebra, is cleanly and modularly separated from the data level, which is modelled within the concrete semantics (Armstrong et al. 2016).

Yet, for simplifying such components and making them available for larger classes of models and applications, it seems natural to review and try to generalise the algebraic foundations on which they depend.

This motivates the definition of Hoare semigroups (or H-semigroups), which are sets equipped with an associative multiplication, an addition of which nothing is required, a transitive relation, with respect to which these operations are compatible, and an operation of weak iteration that satisfies a simulation law (Section 2). A generic Hoare logic is derivable in this minimalist setting when simple additional conditions are imposed on the conditional and the loop rule. A basic refinement calculus is derivable in R-semigroups, an expansion by one additional operation and two further axioms (Section 3). Within this approach, Hoare triples are encoded à la Kleene algebra with tests, but programs are not distinguished from tests or assertions.

Test H-monoids are introduced next to capture that distinction and provide a basic abstract semantics for structured programs (Section 4). Hoare triples and specification statements can then be restricted to assertions that capture pre- and post-conditions and yield a basis for Kleene algebra with tests and its relatives.

The derivation of refinement laws in R-semigroups uses the rules of Hoare logic in H-semigroups directly. Beyond this, a simple algebraic setting is presented in which the two sets of laws are interderivable. In addition, it is shown that effective translations between verification and refinement proofs are possible (Section 5).

Two main benefits of the H-semigroup approach to Hoare logic are as follows: First of all, it allows developing such logics over arbitrary semirings, and it supports the instantiation of the generic H-semigroup operations in various ways. Addition, for instance, can be interpreted as non-deterministic choice or as parallel composition; weak iteration as finite iteration in a Kleene algebra or possibly infinite iteration in a demonic refinement algebra (Section 6). The development of a new Hoare logic for multirelations (Furusawa and Struth 2015) with rules for sequential and concurrent composition provides an extended example (Section 7) with potential for probabilistic and quantum extensions.

Second, the approach supports the design of simple modular program correctness components. A minimalist verification component based on H-semigroups and a refinement component based on R-semigroups are presented as examples. Both have been implemented in the interactive theorem prover Isabelle/HOL (Nipkow et al. 2002) from scratch using only its main libraries; and both are correct by construction. The compactness of the axioms makes the derivation of generic verification and refinement rules by automated theorem proving and the soundness proof with respect to a relational store semantics very simple (Sections 8 and 9), but some conceptual insights were of course necessary to achieve this level of simplicity and modularity.

The final sections of this article provide a series of counterexamples that justify the H-semigroup and R-semigroup axioms (Section 10), and a derivation of the rules of PHL from the H-semigroup axioms by diagram chase (Section 11).

2. H-semigroups and verification

A set S equipped with an unconstrained operation of type $S \times S \to S$ is sometimes called *magma* or *groupoid*.

Definition 2.1. An *H-semigroup* is a structure $(S, \cdot, +, {}^\circ, \preceq)$ such that $(S, \cdot)$ is a semigroup and $(S, +)$ a magma. The binary relation $\preceq$ on S is transitive; multiplication and addition are left and right isotone with respect to it, that is, $x \preceq y$ implies $zx \preceq zy$, $xz \preceq yz$, $x + z \preceq y + z$ and $z + x \preceq z + y$. The operation ${}^\circ : S \to S$ satisfies the simulation axiom

$$yx \preceq xy \Rightarrow yx^\circ \preceq x^\circ y.$$

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

Definition 2.2. An *H-monoid* is an H-semigroup expanded by a multiplicative unit 1 in which $\preceq$ is a pre-order, that is, reflexive and transitive.

The elements of S can be interpreted as actions, events or tasks of a system, in particular as programs. The product xy could mean that x happens and then, after it finishes, y. The relation $x \preceq y$ could mean that whenever x can happen, y can happen as well, for instance, because action y allows at least the behaviour of action x, or because task y is less prescriptive than task x. In these situations, on the one hand, y has at least as many reasons to be true as x, which gives $x \preceq y$ the flavour of material implication. On the other hand, it is always safe to perform x in place of y, as it would not allow any behaviour prohibited by y. The sum $x + y$ could be a non-deterministic choice between x and y or their parallel composition. Yet, no algebraic assumptions are made and $\cdot$ will be interpreted as a parallel composition in Section 7 as well. Finally, x° could model a weak kind of repetition or iteration of x that could be empty, finite or infinite. In this case, the simulation axiom states that y can happen after a sequence of x's whenever it can happen before such a sequence, provided that y can happen after a single x whenever it can happen before it. Yet, x° could also be an abstraction or projection of x.

Definition 2.3. An *H-triple* is a ternary relation $H \subseteq S \times S \times S$ over an H-semigroup S such that, for all $p, x, q \in S$,

$$H\,p\,x\,q \Leftrightarrow px \preceq xq.$$

This generalised Hoare triple captures the fact that whenever p can happen before x, then q can happen after it. In the context of program verification, p and q are the pre-condition and post-condition of program x. Then, $H\,p\,x\,q$ expresses in the style of Kleene algebra with tests that whenever program x is executed from states that satisfy the pre-condition p and whenever x terminates, then it does so in states satisfying post-condition q.

For deriving the rules of a generalised Hoare logic, two more concepts are needed.

Definition 2.4. Let S be an H-semigroup. An element $p \in S$ is *left superdistributive* if $p(x + y) \preceq px + py$ holds for all $x, y \in S$. It is *right subdistributive* if $xp + yp \preceq (x + y)p$ holds for all $x, y \in S$.

Definition 2.5. An element x of an H-semigroup *reflects* y whenever $xy \preceq yxy$.

This means that whenever y can happen after x, then it can happen before x, too. In other words, if y happens after x, then executing y before x does not restrict x's behaviour.

Reflection is the opposite of preservation $yx \preceq yxy$, which means that if y can happen before x, then it can also happen afterwards. Reflection is preservation when x and y happen backwards in time.

Lemma 2.6. In every H-semigroup,

i. x reflects y if and only if $H\,x\,y\,(xy)$,
ii. x reflects y if x and y commute and y is multiplicatively idempotent.

We are now prepared for the main result in this section.

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

Proposition 2.7.

i. In every H-semigroup with left-superdistributive element t and right-subdistributive element u,

$$H\,p\,x\,q' \wedge q' \leq q \Rightarrow H\,p\,x\,q, \tag{HCons1}$$

$$p \leq p' \wedge H\,p'\,x\,q \Rightarrow H\,p\,x\,q, \tag{HCons2}$$

$$H\,p\,x\,r \wedge H\,r\,y\,q \Rightarrow H\,p\,(xy)\,q, \tag{HSeq}$$

$$H\,t\,v\,(tv) \wedge H\,t\,w\,(tw) \wedge H\,(tv)\,x\,u \wedge H\,(tw)\,y\,u \Rightarrow H\,t\,(vx + wy)\,u, \tag{HCond}$$

$$H\,p\,q\,(pq) \wedge H\,p\,r\,(pr) \wedge H\,(pq)\,x\,p \Rightarrow H\,p\,((qx)^{\circ}r)\,(pr). \tag{HLoop}$$

ii. In every H-monoid, in addition,

$$H\,p\,1\,p. \tag{HSkip}$$

The relation H does not distinguish programs from tests or assertions. As a compensation, reflection conditions have been imposed on some rules. Obviously, (HCons1) and (HCons2) are generalised consequence rules; (HSeq) is a sequential composition rule and (HSkip) a skip rule. (HCond) is a conditional rule with two reflection conditions and (HLoop) an iteration rule with one reflection condition. Assignment rules cannot be specified in this setting (cf. Section 8). In the tradition of Kleene algebras with tests, I call the rules in Proposition 2.7 PHL. In H-monoids, one can merge (HCons1) and (HCons2) into the single consequence rule

$$p \leq p' \wedge H\,p'\,x\,q' \wedge q' \leq q \Rightarrow H\,p\,x\,q. \tag{HCons}$$

Lemma 10.1 below shows that the distributivity assumptions on (HCond) are necessary. Section 8 presents a formal proof of Proposition 2.7 with Isabelle, Section 11 an alternative one by diagram chase. By Lemma 10.3 below, the reflection conditions are necessary for (HCond) and (HLoop), but without them, simplified versions can still be obtained.

Corollary 2.8. In every H-semigroup with left-superdistributive element t and right-subdistributive element u,

$$H\,t\,x\,u \wedge H\,t\,y\,u \Rightarrow H\,t\,(x+y)\,u,$$

$$H\,p\,x\,p \Rightarrow H\,p\,x^{\circ}\,p.$$

The second law simply translates the simulation axiom. A frame rule is derivable, too.

Lemma 2.9. In every H-semigroup,

$$H\,p\,x\,p \wedge H\,q\,x\,r \Rightarrow H\,(pq)\,x\,(pr).$$

The condition $H\,p\,x\,p$ in Corollary 2.8 and Lemma 2.9, which is equivalent to $px \leq xp$, is a strong preservation property. It expresses that if p holds before x, then it holds after x as well. Loop invariants, of course, have this property. Another typical situation is that the actions in p and x do not affect each other and are, in that sense, independent.

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

3. R-semigroups and refinement

It is straightforward to express Morgan's specification statement (Morgan 1994) in H-semigroups by adding one operation and two axioms. A generalised refinement calculus can then be derived. It allows the stepwise modular construction of programs from specifications by restricting their behaviour, usually by elimination of non-determinism. In this context, programs are often seen as executable specifications, or as implementations of specifications in the sense that programs constructed must satisfy the correctness criteria prescribed by their specifications. It is therefore necessary that each individual refinement step preserves correctness.

Definition 3.1. An *R-semigroup* is an H-semigroup expanded by the operation $R : S \times S \to S$ in which $^\circ$ is isotone and that satisfies

$$H\, p\, (R\, p\, q)\, q,$$
$$H\, p\, x\, q \Rightarrow x \leq R\, p\, q.$$

Definition 3.2. An *R-monoid* is an *R-semigroup* that is also an H-monoid.

In every R-semigroup, by definition, the *refinement statement* $R\, p\, q$ is the greatest solution in x of $H\, p\, x\, q$, that is, the greatest element x that satisfies $H\, p\, x\, q$. In the context of program refinement, when $H\, p\, x\, q$ states that program or specification x meets the partial correctness specification in terms of pre-condition p and post-condition q, $R\, p\, q$ thus models the most general program or specification that satisfies the specification statement expressed by the Hoare triple.

The relation $\leq$ serves as the converse of the usual refinement relation, which is sometimes modelled as an implication between specifications. In line with the interpretation given in Section 2, a program or specification x can be used safely in place of y whenever $x \leq y$, as it would not violate the correctness criteria prescribed by y. Hence, whenever y is correct and $x \leq y$, then x must be correct as well. The minimal requirements on $\leq$ imposed in Definition 2.1 are of course essential for refinement: Transitivity makes refinement incremental and the isotonicity properties guarantee its modularity.

Proposition 3.3.

i. In every H-semigroup with left-superdistributive element t and right-subdistributive element u,

$$p \leq p' \wedge q' \leq q \Rightarrow R\, p'\, q' \leq R\, p\, q, \tag{RCons}$$
$$(R\, p\, r)(R\, r\, q) \leq R\, p\, q, \tag{RSeq}$$
$$H\, t\, v\, (tv) \wedge H\, t\, w\, (tw) \Rightarrow vR\, (tv)\, u + wR\, (tw)\, u \leq R\, t\, u, \tag{RCond}$$
$$H\, p\, q\, (pq) \wedge H\, p\, r\, (pr) \Rightarrow (q(R\, (pq)\, p))^\circ r \leq R\, p\, (pr). \tag{RLoop}$$

ii. In every R-monoid,

$$p \leq q \Rightarrow 1 \leq R\, p\, q. \tag{RSkip}$$

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

These formulas generalise Morgan's refinement laws (Morgan 1994). In analogy to the rules of PHL, from which they can be derived (see Lemma 5.1 below), I call them *propositional refinement calculus* (PRC). A formal proof of Proposition 3.3 with Isabelle can be found in Section 9. By Lemma 10.2 and Lemma 10.4 (1) and (2) below, the distributivity and reflection conditions cannot be removed.

The laws $x \preceq R\,0\,1$ and $R\,1\,0 \preceq x$ are often added to PRC for platonic reasons. Obviously, $x \preceq R\,0\,1 \Leftrightarrow H\,0\,x\,1 \Leftrightarrow 0x \preceq x$, which holds in an R-monoid whenever the element 0 satisfies $0 \preceq x$ and $0x = 0$. Moreover, $H\,1\,(R\,1\,0)\,0 \Leftrightarrow 1R\,1\,0 \preceq (R\,1\,0)0$, whence $R\,1\,0 = 0 \preceq x$ holds in an R-monoid whenever $x0 = 0$. Another interesting law is $1 \preceq R\,p\,p$, wich follows in R-monoids from (RSkip) and transitivity of $\preceq$.

Finally, the following frame law holds.

Lemma 3.4. In every R-semiring,

$$H\,r\,(R\,p\,q)\,r \Rightarrow R\,p\,q \preceq R\,(rp)\,(rq).$$

R-semigroups could have been axiomatised by using $H\,p\,x\,q \Leftrightarrow x \preceq R\,p\,q$. The two R-semigroup axioms above, and hence the laws of PRC, can be derived from this law. However, by Lemma 10.4(3) below, this law is not implied by the two R-semigroup axioms, which are therefore strictly weaker.

4. Test H-monoids and assertions

Hoare logics usually distinguish programs from assertions or tests. This cannot be captured by H-semigroups or R-semigroups alone. Structured programs such as conditionals or loops, which depend on binary tests, cannot be specified either. To address this in a semigroup setting, the following definition has been proposed in unpublished joint work with Peter Jipsen.

Definition 4.1. A *test monoid* is a monoid $(S, \cdot, 1)$ expanded by an *anti-test* operation $- : S \to S$ that satisfies

$$-(-(-0)) = -0,$$
$$-x \cdot -(-x) = 0,$$
$$-x \cdot -(-(-z) \cdot -(-y)) = -(-(-x \cdot -y) \cdot -(-x \cdot -z)).$$

Defining a *test* operation $t\,x = -(-x)$ as well as $0 = -1$, and using the *test disjunction* operation defined as $t\,x \oplus t\,y = -(-x \cdot -y)$, these axioms can be written more succinctly as $t\,1 = 1$, $-x \cdot t\,x = 0$ and $-x \cdot -(t\,z \cdot t\,y) = -((t\,x \oplus t\,y) \cdot (t\,x \oplus t\,z))$.

Lemma 4.2. In every test monoid S, the operation t is a retraction, $t \circ t = t$, hence $x \in t(S) \Leftrightarrow t\,x = x$.

The elements of $t(S)$, the image of S under t, are called *tests*, and according to Lemma 4.2, tests are fixpoints of t. I henceforth write $p, q, r, \ldots$ for tests and x, y, z for general

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

elements. It is straightforward to show that $t(S) = -(S)$. Lemma 4.2 is useful for proving the following fact.

Lemma 4.3. In every test monoid, $(t(S), \oplus, \cdot, -, 0, 1)$ forms a boolean subalgebra of S.

This boolean structure is of course desirable for tests and assertions. The following definition links test monoids with H-monoids.

Definition 4.4. A *test H-monoid* is a structure $(S, +, \cdot, -, ^\circ, 1, \preceq)$ where $(S, +, \cdot, ^\circ, 1, \preceq)$ is an H-monoid, $(S, \cdot, -, 1)$ a test monoid, and all tests $p \in t(S)$ are left subdistributive and right superdistributive.

Note that $\preceq$ need not coincide with the lattice order on tests and that $\oplus$, which is associative, commutative and idempotent, does not in general coincide with $+$ on tests.

In test H-monoids, Hoare triples can be specified à la Kleene algebra with tests. The rules of PHL, as instances of those in Proposition 2.7, are now derivable without reflection conditions.

Lemma 4.5. If S is a test H-monoid, then $H\,p\,q\,(pq)$ holds for all $p, q \in t(S)$.

Conditional and loop commands can be defined as

$$\mathbf{if}\ p\ \mathbf{then}\ x\ \mathbf{else}\ y = px + -py,$$
$$\mathbf{loop}_\circ\ p\ x = (px)^\circ \cdot -p.$$

Hoare triples can be restricted to tests in the first and third argument: $H \subseteq t(S) \times S \times t(S)$. The conditional and loop rules in Proposition 2.7 thus specialise as follows.

Corollary 4.6. In every test H-monoid,

$$H\,(tv)\,x\,u \wedge H\,(t \cdot -v)\,y\,u \Rightarrow H\,t\,(\mathbf{if}\ v\ \mathbf{then}\ x\ \mathbf{else}\ y)\,u,$$
$$H\,(pq)\,x\,p \Rightarrow H\,p\,(\mathbf{loop}\ q\ x)\,(p \cdot -q).$$

A *test R-monoid* is an R-monoid that is also a test H-monoid in which the specification statement is restricted to type $t(S) \times t(S) \to S$. The refinement laws for conditionals and loops in Proposition 3.3 then specialise as follows.

Corollary 4.7. In every test R-monoid,

$$\mathbf{if}\ v\ \mathbf{then}\ R\,(vt)\,u\ \mathbf{else}\ R\,(-vt)\,u \preceq R\,t\,u,$$
$$\mathbf{loop}_\circ(q, R\,(qp)\,p) \preceq R\,p\,(p \cdot -q).$$

5. Verification vs. refinement

In practise, it is often straightforward to transform verification proofs into refinement proofs and vice versa. This section considers this observation from a formal point of view. Related to this, I first collect some conditions under which verification rules and refinement laws become interderivable.

Lemma 5.1. Let $(S, \preceq)$ be a pre-order endowed with operations $\cdot, +, R : S \times S \to S$, $^\circ : S \to S$ and $1 \in S$. Assume that $\cdot$, $+$ and $^\circ$ are isotone and that, for $H\,p\,x\,q \Leftrightarrow px \preceq xq$,

$$H\,p\,x\,q \Leftrightarrow x \preceq R\,p\,q. \tag{1}$$

Then $(HX) \Leftrightarrow (RX)$ for $X \in \{\text{Cons, Seq, Skip, Cond, Loop}\}$ under the usual distributivity constraints on the conditional rules.

Proof. The R-monoid axioms $H\,p\,(R\,p\,q)\,q$ and $H\,p\,x\,q \Leftrightarrow x \preceq R\,p\,q$ are derivable from Equation (1) and can thus be used in the proof.

— (HCons) $\Leftrightarrow$ (RCons). Suppose that $p \preceq p'$ and $q' \preceq q$. Then

$$H\,p'\,(R\,p'\,q')\,q' \Rightarrow H\,p\,(R\,p'\,q')\,q \Leftrightarrow R\,p'\,q' \preceq R\,p\,q$$

by (HCons). Conversely, $H\,p'\,x\,q' \Leftrightarrow x \preceq R\,p'\,q' \preceq R\,p\,q \Leftrightarrow H\,p\,x\,q$ by (RCond) and the two assumptions.

— (HSeq) $\Leftrightarrow$ (RSeq).

$$H\,p\,(R\,p\,r)\,r \wedge H\,r\,(R\,r\,q)\,q \Rightarrow H\,p\,((R\,p\,r)(R\,r\,q))\,q \Leftrightarrow (R\,p\,r)(R\,r\,q) \preceq R\,p\,q$$

by (HSeq). Conversely, by (RSeq) and isotonicity of $\cdot$,

$$H\,p\,x\,r \wedge H\,r\,y\,q \Leftrightarrow x \preceq R\,p\,r \wedge y \preceq R\,r\,q \Rightarrow xy \preceq (R\,p\,r)(R\,r\,q) \preceq R\,p\,q \Leftrightarrow H\,p\,(xy)\,q.$$

— (HSkip) $\Leftrightarrow$ (RSkip). $1 \preceq R\,p\,p$ by (HSkip), and the claim follows from (RCons). Conversely, (RSkip) implies $H\,p\,1\,p$.

— (HCond) $\Leftrightarrow$ (RCond). First,

$$\begin{aligned} H\,(tv)\,(R\,(tv)\,u)\,u \wedge H\,(tw)\,(R\,(tw)\,u)\,u &\Rightarrow H\,t\,(vR\,(tv)\,u + wR\,(tw)\,u)\,u \\ &\Leftrightarrow vR\,(tv)\,u + wR\,(tw)\,u \preceq R\,t\,u \end{aligned}$$

by (HCond). Conversely,

$$\begin{aligned} H\,(tv)\,x\,u \wedge H\,(tw)\,y\,u &\Leftrightarrow x \preceq R\,(tv)\,u \wedge y \preceq R\,(tw)\,u \\ &\Rightarrow vx + wy \preceq vR\,(tv)\,u + wR\,(tw)\,u \preceq R\,t\,u \\ &\Leftrightarrow H\,t\,(vx + wy)\,u \end{aligned}$$

by (RCond) and isotonicity of $\cdot$ and $+$. The distributivity and reflection conditions have not been mentioned explicitly. They are the same for (RCond) and (HCond).

— (HLoop) $\Leftrightarrow$ (RLoop). First,

$$H\,(pq)(R\,(pq)\,p)\,p \Rightarrow H\,p\,((qR\,(pq)\,p)^\circ r)\,(pr) \Leftrightarrow (qR\,(pq)\,p)^\circ r \preceq R\,p\,(pr)$$

by (HLoop). Conversely,

$$H\,(pq)\,x\,p \Leftrightarrow x \preceq R\,(pq)\,p \Rightarrow (qx)^\circ r \preceq (qR\,(pq)\,p)^\circ r \preceq R\,p\,(pr) \Leftrightarrow H\,p\,((qx)^\circ r)\,pr$$

by (RLoop) and isotonicity of $\cdot$ and $^\circ$. The reflection conditions have again not been mentioned.

$\square$

The derivation of PRC from PHL has already been reported in the context of Kleene algebras with tests expanded by the refinement statement and Equation (1) as an axiom (Armstrong et al. 2016); the converse direction is new. Lemma 10.5 below yields a counterexample if the R-semigroup axioms $H\,p\,(R\,p\,q)\,q$ and $H\,p\,x\,q \Rightarrow x \preceq R\,p\,q$ are assumed instead of Equation (1). In that case, PRC follows from PHL, but not conversely.

Finally, Equation (1) covers the interderivability of assignment rules. Hoare's assignment axiom is of the form $H\,p[e/v]\,(v := e)\,p$, where $p[e/v]$ denotes that the value of variable v in the program store is updated to e. The standard refinement rule for assignments is $(v := e) \preceq R\,p[e/v]\,p$. Thus, obviously,

$$H\,p[e/v]\,(v := e)\,p \Leftrightarrow (v := e) \preceq R\,p[e/v]\,p \tag{2}$$

is an instance of Equation (1). Other typical refinement rules for assignments are derivable by using the consequence and sequential composition rules of PHL (cf. Section 9).

The next lemmas show that PRC proofs can be constructed from step-wise proofs in PHL and vice versa. In this context, I assume the pre-order axioms for $\preceq$. If the rules PHL from Proposition 2.7 have been added to them, I write H for the resulting set. Otherwise, if the refinement laws PRC from Proposition 3.3 and the isotonicity laws for all operations have been added, I write R. While programs are defined in the standard way as a recursive data type or grammar over a given set of atoms.

Lemma 5.2. Every proof in H translates effectively into a proof in R if, for every atomic program c, $\mathsf{H} \vdash H\,p\,c\,q \Rightarrow \mathsf{R} \vdash c \preceq R\,p\,q$.

Proof. By induction on H-proofs:

- There are two base cases. For 1, $\mathsf{H} \vdash H\,p\,1\,q$ and $\mathsf{R} \vdash 1 \preceq R\,p\,p$ by Proposition 2.7 and Proposition 3.3. Thus, $\mathsf{H} \vdash H\,p\,1\,q \Rightarrow \mathsf{R} \vdash 1 \preceq R\,p\,p$. For atomic programs, the assumption applies.
- Suppose $\mathsf{H} \vdash H\,p\,x\,q$ and the last proof step was $\mathsf{H} \vdash H\,p'\,x\,q'$ by (HCons) with $p \preceq p'$ and $q' \preceq q$. Then $\mathsf{R} \vdash x \preceq R\,p'\,q'$ by the induction hypothesis and $\mathsf{R} \vdash x \preceq R\,p\,q$ with the assumptions and (RCons).
- Suppose $\mathsf{H} \vdash H\,p(xy)\,q$ and the last proof step was (HSeq). Then there were proofs $\mathsf{H} \vdash H\,px\,r$ and $\mathsf{H} \vdash H\,ry\,q$ for some $r \in S$. Therefore, $\mathsf{R} \vdash x \preceq R\,p\,r$ and $\mathsf{R} \vdash y \preceq R\,r\,q$ by the induction hypothesis and $\mathsf{R} \vdash xy \preceq (R\,p\,r)(R\,r\,q) \preceq R\,p\,q$ with (RSeq) and isotonicity of $\cdot$.
- Suppose $\mathsf{H} \vdash H\,p\,(vx + wy)\,q$ and the last proof step was (HCond). Then there were proofs $\mathsf{H} \vdash H\,(pv)\,x\,q$ and $\mathsf{H} \vdash H\,(pw)\,y\,q$. Therefore, $\mathsf{R} \vdash x \preceq R\,(pv)\,q$ and $\mathsf{R} \vdash y \preceq R\,(pw)\,q$ by the induction hypothesis and $\mathsf{R} \vdash vx + wy \preceq vR\,(pv)\,q + wR\,(pw)\,q \preceq R\,p\,q$ by (RCond) with isotonicity of $+$ and $\cdot$.
- Suppose $\mathsf{H} \vdash H\,p\,((qx)^\circ r)\,(pr)$ and the last proof step was (HLoop). Then there was a proof $\mathsf{H} \vdash H\,(pq)\,x\,p$. Therefore, $\mathsf{R} \vdash x \preceq R\,(pq)\,p$ by the induction hypothesis and $\mathsf{R} \vdash (qx)^\circ r \preceq (qR\,(pq)\,p)^\circ r \preceq R\,p\,(pr)$ by (RLoop) with isotonicity of $\cdot$ and $^\circ$.

□

For translating refinement proofs into verification proofs, the following fact is useful.

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

Lemma 5.3. Let α, β be specification statements. The expressions generated by R satifsy

i. if $\mathsf{R} \vdash x \preceq \alpha\beta$, then $x = x_1x_2$ and $\mathsf{R} \vdash x_1 \preceq \alpha$ and $\mathsf{R} \vdash x_2 \preceq \beta$ for some $x_1, x_2 \in S$;
ii. if $\mathsf{R} \vdash x \preceq y\alpha + z\beta$, then $x = yx_1 + zx_2$ and $\mathsf{R} \vdash x_1 \preceq \alpha$ and $\mathsf{R} \vdash x_2 \preceq \beta$ for some $x_!, x_2 \in S$;
iii. if $\mathsf{R} \vdash x \preceq (y\alpha)^\circ z$, then $x = (yw)^\circ z$ and $\mathsf{R} \vdash z \preceq \alpha$ for some $w \in S$.

Proof. The laws in R form a context-free grammar when the constraints are ignored. The conditional rule can be seen as an infinite set of rules between non-terminals. It is therefore clear that these rules generate trees. For instance, $x \preceq \alpha\beta$ says that from a certain tree $\alpha \cdot \beta$ with terminal $\cdot$ one can expand to a tree x. Hence, the tree x will still have root $\cdot$ and two subtrees x_1 and x_2 stemming from the expansion of α and β, respectively. In other words, $x = x_1 \cdot x_2$. The arguments for the other cases are similar. □

Lemma 5.4. Every proof in R translates effectively into a proof in H if, for every atomic program c (e.g. an assignment statement), $\mathsf{R} \vdash c \preceq R\,p\,q \Rightarrow \mathsf{H} \vdash H\,p\,c\,q$.

Proof. By induction on R proofs:

— The base cases are trivial as in Lemma 5.2 and I do not repeat them.
— Suppose the last step is (RCons). Then there is a proof $\mathsf{R} \vdash x \preceq R\,p'\,q' \preceq R\,p\,q$ and the last step uses $p \preceq p'$ and $q' \preceq q$. Then $\mathsf{H} \vdash H\,p'\,x\,q'$ by the induction hypothesis and $\mathsf{H} \vdash H\,p\,x\,q$ by (HCons).
— Suppose the last step is (RSeq). Then there is a proof $\mathsf{R} \vdash x \preceq (R\,p\,r)(R\,r\,q) \preceq R\,p\,q$ for some $r \in S$. By Lemma 5.3, there are proofs $\mathsf{R} \vdash x_1 \preceq R\,p\,r$ and $\mathsf{R} \vdash x_2 \preceq R\,r\,q$ for some $x_1, x_2 \in S$ with $x = x_1x_2$. Then $\mathsf{H} \vdash H\,p\;x_1\,r$ and $\mathsf{H} \vdash H\,r\,x_2\,q$ by the induction hypothesis, and $\mathsf{H} \vdash H\,p\,x\,q$ by (HSeq).
— Suppose the last step is (RCond). Then there is a proof $x \preceq vR\,(pv)\,q + wR\,(pw)\,q \preceq R\,p\,q$. By Lemma 5.3, there are $x_1, x_2 \in S$ such that $x = v + x_1 + w + x_2$ and $\mathsf{R} \vdash x_1 \preceq R\,(pv)\,q$ and $\mathsf{R} \vdash x_2 \preceq R\,(pw)\,q$. Then $\mathsf{H} \vdash H\,(pv)\,x_1\,q$ and $\mathsf{H} \vdash H\,(pw)\,x_2\,q$ by the induction hypothesis, and $\mathsf{H} \vdash H\,p\,x\,q$ by (HCond).
— Suppose that the last step is (RLoop). Then there is a proof $\mathsf{R} \vdash x \preceq (pR\,(pq)\,p)^\circ r \preceq R\,p\,(pr)$. By Lemma 5.3, there exists an $y \in S$ such that $x = (qy)^* r$ and $\mathsf{R} \vdash y \preceq R\,(pq)\,p$. Then $\mathsf{H} \vdash H\,(pq)\,y\,p$ by the induction hypothesis and $\mathsf{H} \vdash H\,p\,x\,(pr)$ by (HLoop).

□

By Equation (2), assignment rules translate as well. An example verification proof can be found in Section 8; a refinement proof and a brief discussion of their translation in Section 9.

6. Instances

In H-semigroups and R-semigroups with or without tests, no algebraic axioms have been imposed on addition and little interaction between addition and multiplication has been assumed apart from the weak distributivity conditions for (HCond). This makes Propositions 2.7 and 3.3 available in a wide range of algebras. Here, I briefly review

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

two classes of instances: algebras without tests in which distributivity assumptions are subsumed, and test algebras in which reflection assumptions are subsumed.

An *H-semiring* is a pre-ordered semiring equipped with an operation $^\circ$ that satisfies the simulation axiom. In this case, the distributivity laws $x(y+z) = xy + xz$ and $(x+y)z = xz + yz$ hold, and distributivity conditions are subsumed. Pre-ordered means that the carrier set of the semiring is a pre-order and the operations of addition and multiplication are isotone. If there is an additive unit 0, then it is reasonable to assume that S is *positive*, that is, $0 \preceq x$ for all $x \in S$. Obviously, every H-semiring is an H-semigroup.

Of particular interest are *H-dioids*, which are H-semirings S in which addition is idempotent; $x+x = x$ for all $x \in S$. This means that the additive semigroup is a semilattice, whence every H-dioid is naturally ordered by the semilattice order $x \leqslant y \Leftrightarrow x + y = y$.

In both settings, PHL with the usual reflection conditions can be derived. The extensions to R-semirings are obvious, and the rules in PRC can then be derived as well.

The operation $^\circ$ of weak iteration can be instantiated in various ways, two of which have been studied widely in the context of program correctness.

First, a *Kleene algebra* is a dioid S with additive unit 0 and multiplicative unit 1 that is expanded by an operation $^* : S \to S$ that satisfies $1 + xx^* \leqslant x^*$, $1 + x^*x \leqslant x^*$, $z + xy \leqslant y \Rightarrow x^*z \leqslant y$ and $z + yx \leqslant y \Rightarrow zx^* \leqslant y$. In this setting, the simulation law $yx \leqslant xy \Rightarrow yx^* \leqslant x^*y$ is derivable (cf. Armstrong 2013b), whence very Kleene algebra is an H-monoid with $^\circ$ instantiated by *.

Second, a *demonic refinement algebra* (von Wright 2004) is a Kleene algebra S expanded by an operation $^\infty : S \to S$ that satisfies $1 + xx^\infty = x^\infty$, $y \leqslant xy + z \Rightarrow y \leqslant x^\infty z$ and $x^\infty = x^* + x^\infty 0$. Now, the simulation law $yx \leqslant xy \Rightarrow yx^\infty \leqslant x^\infty y$ is derivable as well (cf. Armstrong 2013b). It follows that every demonic refinement algebra is an H-monoid with respect to both * and $^\infty$.

The rules of PHL are thus derivable in Kleene algebras and demonic refinement algebras, in particular loop rules for both * and $^\infty$ can be obtained in the latter. In these algebras, * models finite iteration whereas $^\infty$ models potentially infinite iteration that may or may not terminate. A strictly infinite iteration can also be modelled in the context of dioids. While this operation is interesting for the verification of reactive systems, it does not satisfy the simulation axiom needed for H-semigroups (Armstrong 2013b), and therefore does not yield a loop rule in the style of Proposition 2.7 or Corollary 2.8.

A wide range of test semirings has been introduced and formalised in Isabelle (Armstrong et al. 2014). Their axioms are similar to those of test monoids, and it follows that each of these variants forms a dioid and a test monoid. The most important examples are as follows. A *Kleene algebra with tests* is a Kleene algebra that is also a test semiring. Every Kleene algebra with tests is a test H-monoid, and, for $\mathbf{loop}_*(p, x) = \mathbf{while}\ p\ \mathbf{do}\ x$, the classical while-rule

$$H\,(pq)\,x\,p \Rightarrow H\,p\,(\mathbf{while}\ q\ \mathbf{do}\ x)\,(p \cdot -q)$$

can be derived. Demonic refinement algebras with tests can be defined along the lines of Kleene algebras with tests (Armstrong et al. 2014) and the conditional and loop laws can be restricted accordingly.

Finally, the results discussed in this section are compatible with modal Kleene algebras (Desharnais and Struth 2011), for which Hoare logics have been derived as well. Intuitively, a modal Kleene algebra is a Kleene algebra S expanded with an anti-domain operation $a : S \to S$ that models those states from which a program is not enabled. Every modal Kleene algebra is a Kleene algebra with tests (Desharnais and Struth 2011) because the anti-domain operations satisfies the axioms for anti-tests above. The results for Kleene algebras with tests and PHL thus carry over seamlessly.

7. Hoare logic for multirelations

This section presents binary multirelations as an extended example. In this setting, the rules of PHL can be instantiated in various ways, giving addition and multiplication of an H-semigroup different interpretations as non-deterministic choice, sequential composition and parallel composition. A slight caveat is that multirelations with sequential composition as multiplication only form H-monoids if some restrictions on factors are imposed. Classes of multirelations that form H-monoids directly, for instance, union-closed or up-closed multirelations (cf. Furusawa and Struth 2015) and similar restrictions relevant for modelling probabilistic or quantum programs (e.g. Chadha et al. 2006; den Hartog and de Vink 2002) could have been considered instead, but the slight mathematical inconvenience of the general case is certainly compensated by the fact that it leads to Hoare logics and refinement calculi for all specialisations mentioned.

Peleg (1987) has proposed a concurrent dynamic logic that aims to study concurrency in its purest form as the dual notion to non-determinism. The semantics of this logic can be presented in terms of an algebra of binary multirelations (Furusawa and Struth 2015). This section shows how multirelations can be endowed with a Hoare logic.

A *multirelation* over a set X is a binary relation of type $X \times \mathscr{P} X$. Hence, an element of a multirelation relates an element $a \in X$ with a subset A of X. I write $\mathscr{M}(X) = \mathscr{P}(X \times \mathscr{P} X)$ for the set of multirelations over X.

Peleg's sequential composition of multirelations R and S is rather complicated. It is the multirelation

$$R \cdot S = \left\{ (a, A) \mid \exists B.\ (a, B) \in R \wedge \exists f.\ (\forall b \in B.\ (b, f\, b) \in S) \wedge A = \bigcup_{b \in B} f\, b \right\}.$$

By this definition, a pair (a, A) is in the multirelation $R \cdot S$ whenever R relates a to some intermediate set B and S relates each $b \in B$ to a set $f\, b$ in such a way that $A = \bigcup_{b \in B} f\, b$. The unit of sequential composition is $1_\sigma = \{(a, \{a\}) \mid a \in X\}$. The parallel composition of R and S is defined as the multirelation

$$R \| S = \{(a, A \cup B) \mid (a, A) \in R \wedge (a, B) \in S\}.$$

The unit of parallel composition is $1_\pi = \{(a, \emptyset) \mid a \in X\}$.

Multirelations $P \subseteq 1_\sigma$, the (sequential) subidentities, form a boolean subalgebra of $\mathscr{M}(X)$ in which $\emptyset$ is the least and 1_σ the greatest element. Union is join and sequential composition, which coincides with parallel composition, is meet. The complement in the subalgebra is $-P = 1_\sigma \cap \overline{P \cdot U}$, where $\overline{P}$ is the complement of P on $\mathscr{M}(X)$ and U the universal multirelation, which relates any element of X to any of it subsets.

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

It is easy to show that $P \subseteq 1_\sigma \Leftrightarrow --P = P$, and that $-$ satisfies the test axioms for arbitrary multirelations. However, multirelations do not form test monoids with respect to sequential composition. The main reason is that sequential composition of multirelations is not associative: $(R \cdot S) \cdot T \subseteq R \cdot (S \cdot T)$, but not in general $R \cdot (S \cdot T) \subseteq (R \cdot S) \cdot T$. However, $(R \cdot S) \cdot T = R \cdot (S \cdot T)$ if one of R, S, T is a subidentity. Similarly, $R \subseteq S \Rightarrow T \cdot R \subseteq T \cdot S$ and $(R \cup S) \cdot T = R \cdot T \cup S \cdot T$, but $R \cdot (S \cup T) = R \cdot S \cup R \cdot T$ holds only if R is a subidentity. These properties suffice to prove the following fact.

Lemma 7.1. For all $P, P', Q, Q', R, S \in \mathscr{M}(X)$ such that $P, P', Q, Q' \subseteq 1_\sigma$,

$$\begin{aligned} & H\,P\,1_\sigma\,P, \\ P \subseteq P' \wedge H\,P'\,R\,Q' \wedge Q' \subseteq Q \Rightarrow {} & H\,P\,R\,Q, \\ H\,P\,R\,P' \wedge H\,P'\,S\,Q \Rightarrow {} & H\,P\,(R \cdot S)\,Q, \\ H\,(P \cdot Q)\,R\,P' \wedge H\,(P \cdot -Q)\,S\,P' \Rightarrow {} & H\,P\,(\mathsf{if}\,Q\,\mathsf{then}\,R\,\mathsf{else}\,S)\,P'. \end{aligned}$$

This lemma – more precisely its sequential composition and conditional rule – is not an immediate instance of Proposition 2.7, as subidentities are required in the right places to apply the associativity properties needed. By contrast, the distributivity assumptions of (HCond) hold in the multirelational setting according to the discussion above. With these restrictions in place, each proof of the respective rule of PHL goes through as before. In particular, all subidentities P and Q satisfy the reflection conditions $H\,P\,Q\,(P \cdot Q)$ as their multiplication is commutative and idempotent.

Parallel composition is better behaved. $(\mathscr{M}(X), \|, 1_\pi)$ forms a commutative monoid, and the distributivity laws $(R\|S) \cdot T = (R \cdot T)\|(S \cdot T)$ and $T \cdot (R\|S) = (T \cdot R)\|(T \cdot S)$ hold if T is a subidentity. Thus, with subidentities occurring in the right places in equations, multirelations under sequential and parallel composition almost form semirings (though not dioids because parallel composition is not idempotent). Because subidentities satisfy the subdistributivity and superdistributivity conditions of Corollary 2.8, the following simple parallel composition rule is an immediate consequence of that corollary.

Lemma 7.2. For all $P, Q, R, S \in \mathscr{M}(X)$ such that $P, Q \subseteq 1_\sigma$,

$$H\,P\,R\,Q \wedge H\,P\,S\,Q \Rightarrow H\,P\,(R\|S)\,Q.$$

A star operation as the least fixpoint of $\lambda X.\ S \cup R \cdot X$ has already been investigated (Furusawa and Struth 2015). Here, however, the least fixpoints of the dual functions

$$\begin{aligned} F_{SR} &= \lambda X.\ S \cup X \cdot R, \\ F_R &= 1_\sigma \cup X \cdot R \end{aligned}$$

are needed for Hoare logic. Both are isotone on the complete lattice $(\mathscr{M}(X), \cup, \cap, \emptyset, U)$ and thus have indeed least (pre-)fixpoints. I write $(S^* R)$ for the (binary) fixpoint μF_{SR} of F_{SR} and R^* for the (unary) fixpoint μF_R of F_R. Proving the simulation law of H-semigoups requires showing that $(P^* R) = P \cdot R^*$, that is, $\mu F_{PR} = P \cdot \mu F_R$, at least for $P \subseteq 1_\sigma$. This can be established by fixpoint fusion whenever P is a subidentity.

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

Theorem 7.3 (Meijer et al. 1991). If f and g are isotone functions and h is a continuous function over a complete lattice, then $f \circ h = h \circ g$ implies $\mu f = h\,\mu g$.

Lemma 7.4. Let $R, P \in \mathscr{M}(X)$ and $P \subseteq 1_\sigma$. Then $(P^* R) = P \cdot R^*$.

Proof. Instantiate $f = F_{SR}$, $g = F_R$ and $h = H = \lambda X.X \cdot P$ in the fixpoint fusion theorem. It is routine to show that H is continuous if P is a subidentity, $P \cdot \bigcup_{i \in I} R_i = \bigcup_{i \in I} P \cdot R_i$. Moreover,

$$(F_{SR} \circ H)\,x = P \cup (P \cdot x) \cdot R = P \cdot (1_\sigma \cup x \cdot R) = (H \circ F_R)\,x$$

by left distributivity and associativity of multiplication with subidentities. $\square$

The following fact is then immediate from the (least) fixpoint properties of F_R and F_{SR}.

Lemma 7.5. Let $P, R, S \in \mathscr{M}(X)$ and $P \subseteq 1_\sigma$. Then the following star unfold and induction laws hold:

$$1_\sigma \cup R^* \cdot R \subseteq R^*,$$
$$P \cup S \cdot R \subseteq S \Rightarrow P \cdot R^* \subseteq S.$$

Lemma 7.6. Let $P, R \in \mathscr{M}(X)$ and $P \subseteq 1_\sigma$. Then the star simulation law holds

$$P \cdot R \subseteq R \cdot P \Rightarrow P \cdot R^* \subseteq R^* \cdot P.$$

Proof. Let $P \cdot R \subseteq R \cdot P$. For $P \cdot R^* \subseteq R^* \cdot P$, it suffices to show that $P \cup (R^* \cdot P) \cdot R \subseteq R^* \cdot P$ by star induction. Indeed, $P \cup R^* \cdot P \cdot R \subseteq P \cup R^* \cdot R \cdot P = (1_\sigma \cup R^* \cdot R) \cdot P = R^* \cdot P$, by associativity of multiplication in the presence of P, the isotonicities and distributivities of multirelations and the star unfold law. $\square$

The following loop rule is thus derivable:

Lemma 7.7. For all $P, Q, R \in \mathscr{M}(X)$ such that $P, Q \subseteq 1_\sigma$,

$$H\,(P \cdot Q)\,R\,P \Rightarrow H\,P\,(\textbf{while } Q \textbf{ do } R)\,(P \cdot -Q).$$

A refinement calculus for multirelations as in Proposition 3.3 can be obtained as well, since all assumptions for Lemma 5.1 are satisfied.

It is interesting to note that $(\mathscr{M}(X), \cup, \|, \emptyset, 1_\pi)$ forms a positive semiring. In this setting, Hoare triples can be defined with respect to parallel composition as well, $\tilde{H}\,P\,R\,Q \Leftrightarrow P \| R \subseteq Q \| R$, and for arbitrary P, Q and R. The rules of PHL can then be derived directly as an instance of Proposition 2.7, including a loop rule with respect to a star for parallel composition. The sequential composition rule then becomes

$$P \| R \subseteq P' \| R \wedge P' \| S \subseteq Q \| S \Rightarrow P \| R \| S \subseteq Q \| R \| S.$$

This law covers situation where $P \nsubseteq P'$ and $P' \nsubseteq Q$. Similarly, even when $P \nsubseteq Q$,

$$P \| R \subseteq Q \| R \wedge P \| S \subseteq Q \| S \Rightarrow P \| (R \cup S) \subseteq Q \| (R \cup S).$$

Assignment rules can be added to a multirelational program semantics in various ways. The simplest case are deterministic assignments which can be modelled essentially as outlined in the next section, using the fact that every binary relation R can be embedded into a multirelation $R^\dagger$ by $(a,b) \in R \Leftrightarrow (a,\{b\}) \in R^\dagger$. The consideration of demonically non-deterministic, probabilistic or random assignments $x := E$, where E is a set of values such as a probability (sub)distribution, can be expressed by multirelations, but is beyond the scope of this article.

8. A minimalist verification component

A main benefit of the generalisation of Hoare logics via H-semigroups and H-monoids is that it makes the implementation of verification components in interactive theorem provers easy: only very few simple algebraic properties need to be checked.

This section presents a minimalist verification component for while programs. It has been implemented in Isabelle/HOL from scratch, using only Isabelle's main libraries[†] (a GCD component is needed for reasoning about Euclid's algorithm). The general method is as follows. In the first part of the Isabelle code below, H-monoids are defined as an axiomatic type class. Hoare triples are then introduced and the rules of a generic and polymorphic PHL are derived by automated theorem proving from the H-monoid axioms. By contrast to Section 2, rule (HLoop) is refined to be used for while loops with invariants.

Next, polymorphic predicates are defined as boolean-valued functions from type $'a$. A polymorphic store is modelled, as usual, as a function from variables, which are represented by strings, to elements of type $'a$. It can handle data of arbitrary type.

It is then shown that binary relations satisfy the simulation axioms of H-semigroups and, by an interpretation statement, that binary relations over arbitrary unspecific universes X under the identity relation $Id = \{(a,a) \mid a \in X\}$, relative composition $R;S = \{(a,b) \mid \exists c.\ (a,c) \in R \wedge (c,b) \in S\}$, set union and the reflexive–transititive closure operation $R^* = \bigcup_{i\in\mathbb{N}} R^i$ form H-monoids.

The assignment command $v ::= e$ is then defined in this denotational relational store semantics of H-monoids, using Isabelle's built-in function update operation :=, and a variant of Hoare's assignment axiom is derived, using a function $\lceil\_\rceil$ that embeds predicates into relations. The last few lines set up syntactic sugar for while programs. Note that the distributivity assumptions and reflection conditions have been discharged in the conditional and while rules in the relational semantics.

```
theory H-Semigroup
  imports Main GCD

begin

notation times (infixl · 70)
and relcomp (infixl ; 70)

class H-monoid = monoid-mult + plus +
  fixes preo :: 'a ⇒ 'a ⇒ bool (infixl ≤ 50)
  and star :: 'a ⇒ 'a (-⋆ [101] 100)
```

[†] https://github.com/gstruth/h-semigroups

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

assumes *preo-refl*: $x \preceq x$
and *preo-trans*: $x \preceq y \Longrightarrow y \preceq z \Longrightarrow x \preceq z$
and *add-isol*: $x \preceq y \Longrightarrow z + x \preceq z + y$
and *add-isor*: $x \preceq y \Longrightarrow x + z \preceq y + z$
and *mult-isol*: $x \preceq y \Longrightarrow z \cdot x \preceq z \cdot y$
and *mult-isor*: $x \preceq y \Longrightarrow x \cdot z \preceq y \cdot z$
and *star-sim*: $y \cdot x \preceq x \cdot y \Longrightarrow y \cdot x^\star \preceq x^\star \cdot y$

begin

definition H :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ **where** $H\ p\ x\ q \longleftrightarrow p \cdot x \preceq x \cdot q$

lemma *H-skip*: $H\ p\ 1\ p$
by (*simp add*: *H-def preo-refl*)

lemma *H-cons*: $p \preceq p' \Longrightarrow H\ p'\ x\ q' \Longrightarrow q' \preceq q \Longrightarrow H\ p\ x\ q$
by (*meson H-def local.mult-isol local.mult-isor local.preo-trans*)

lemma *H-seq*: $H\ r\ y\ q \Longrightarrow H\ p\ x\ r \Longrightarrow H\ p\ (x \cdot y)\ q$
by (*simp add*: *H-def*, *rule preo-trans*, *drule mult-isor*, *auto simp*: *mult-assoc mult-isol*)

lemma *H-cond*:
assumes $\bigwedge x\ y.\ p \cdot (x + y) \preceq p \cdot x + p \cdot y$ **and** $\bigwedge x\ y.\ x \cdot q + y \cdot q \preceq (x + y) \cdot q$
shows $H\ p\ v\ (p \cdot v) \Longrightarrow H\ p\ w\ (p \cdot w) \Longrightarrow H\ (p \cdot v)\ x\ q \Longrightarrow H\ (p \cdot w)\ y\ q$
$\Longrightarrow H\ p\ (v \cdot x + w \cdot y)\ q$
by (*meson H-def assms add-isol add-isor preo-trans H-seq assms*)

lemma *H-loopi*: $H\ i\ v\ (i \cdot v) \Longrightarrow H\ i\ w\ (i \cdot w) \Longrightarrow H\ (i \cdot v)\ x\ i \Longrightarrow p \preceq i \Longrightarrow i \cdot w \preceq q$
$\Longrightarrow H\ p\ ((v \cdot x)^\star \cdot w)\ q$
by (*meson H-cons H-def H-seq local.star-sim*)

end

type-synonym $'a\ pred = 'a \Rightarrow bool$
type-synonym $'a\ store = string \Rightarrow 'a$

lemma *rel-star-sim-aux*: $Y\ ;\ X \subseteq X\ ;\ Y \Longrightarrow Y\ ;\ X\ \hat{}\hat{}\ i \subseteq X\ \hat{}\hat{}\ i\ ;\ Y$
by (*induct i*, *simp-all*, *blast*)

interpretation *rel-hm*: *H-monoid Id op ; op* $\cup$ *op* $\subseteq$ *rtrancl*
by (*standard*, *auto simp*: *SUP-subset-mono rtrancl-is-UN-relpow relcomp-UNION-distrib relcomp-UNION-distrib2 rel-star-sim-aux*)

definition *p2r* :: $'a\ pred \Rightarrow 'a\ rel$ ($\lceil$-$\rceil$) **where** $\lceil P \rceil = \{(s,s)\ |s.\ P\ s\}$

lemma *p2r-mult-hom* [*simp*]: $\lceil P \rceil\ ;\ \lceil Q \rceil = \lceil \lambda s.\ P\ s \wedge Q\ s \rceil$
by (*auto simp*: *p2r-def*)

definition *gets* :: $string \Rightarrow ('a\ store \Rightarrow 'a) \Rightarrow 'a\ store\ rel$ (- ::= - [70, 65] 61) **where**
v ::= $e = \{(s,s\ (v := e\ s))\ |s.\ True\}$

lemma *H-assign*: $(\forall s.\ P\ s \longrightarrow Q\ (s\ (v := e\ s))) \Longrightarrow$ *rel-hm.H* $\lceil P \rceil\ (v$ ::= $e)\ \lceil Q \rceil$
by (*auto simp*: *rel-hm.H-def p2r-def gets-def*)

definition *if-then-else* :: $'a\ pred \Rightarrow 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel$ (*if - then - else - fi* [64,64,64] 63) **where**
if P then X else Y fi $= \lceil P \rceil\ ;\ X \cup \lceil \lambda s.\ \neg\ P\ s \rceil\ ;\ Y$

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

definition *while* :: *'a pred* ⇒ *'a rel* ⇒ *'a rel* (*while - do - od* [64,64] 63) **where**
while P do X od = ($\lceil P \rceil$; X)* ; $\lceil \lambda s.\ \neg\ P\ s \rceil$

definition *while-inv* :: *'a pred* ⇒ *'a pred* ⇒ *'a rel* ⇒ *'a rel* (*while - inv - do - od* [64,64,64] 63) **where**
while P inv I do X od = ($\lceil P \rceil$; X)* ; $\lceil \lambda s.\ \neg\ P\ s \rceil$

lemma *rel-ref* : *rel-hm.H* $\lceil P \rceil$ $\lceil Q \rceil$ ($\lceil P \rceil$; $\lceil Q \rceil$)
by (*auto simp*: *rel-hm.H-def p2r-def*)

lemma *sH-cons*: ($\forall s.\ P\ s \longrightarrow P'\ s$) $\Longrightarrow$ *rel-hm.H* $\lceil P' \rceil$ X $\lceil Q \rceil$ $\Longrightarrow$ ($\forall s.\ Q'\ s \longrightarrow Q\ s$)
$\Longrightarrow$ *rel-hm.H* $\lceil P \rceil$ X $\lceil Q \rceil$
by (*rule rel-hm.H-cons*, *auto simp*: *p2r-def*)

lemma *sH-cond*: *rel-hm.H* ($\lceil P \rceil$; $\lceil T \rceil$) X $\lceil Q \rceil$ $\Longrightarrow$ *rel-hm.H* ($\lceil P \rceil$; $\lceil \lambda s.\ \neg\ T\ s \rceil$) Y $\lceil Q \rceil$
$\Longrightarrow$ *rel-hm.H* $\lceil P \rceil$ (*if T then X else Y fi*) $\lceil Q \rceil$
by (*simp only*: *if-then-else-def*, *intro rel-hm.H-cond*, *auto*, (*metis p2r-mult-hom rel-ref*)+)

lemma *sH-whilei*: $\forall s.\ P\ s \longrightarrow I\ s \Longrightarrow \forall s.\ I\ s \wedge \neg\ R\ s \longrightarrow Q\ s \Longrightarrow$ *rel-hm.H* ($\lceil I \rceil$; $\lceil R \rceil$) X $\lceil I \rceil$
$\Longrightarrow$ *rel-hm.H* $\lceil P \rceil$ (*while R inv I do X od*) $\lceil Q \rceil$
by (*simp only*: *while-inv-def*, *intro rel-hm.H-loopi*, *auto simp*: *p2r-def*, (*metis p2r-def rel-ref*)+)

lemma *euclid*:
rel-hm.H $\lceil \lambda s$::*nat store*. $s\ ''x'' = x \wedge s\ ''y'' = y \rceil$
(*while* ($\lambda s.\ s\ ''y'' \neq 0$) *inv* ($\lambda s.$ *gcd* ($s\ ''x''$) ($s\ ''y''$) = *gcd* $x\ y$)
do
($''z''$ ::= ($\lambda s.\ s\ ''y''$)) ;
($''y''$ ::= ($\lambda s.\ s\ ''x''$ *mod* $s\ ''y''$)) ;
($''x''$ ::= ($\lambda s.\ s\ ''z''$))
od)
$\lceil \lambda s.\ s\ ''x'' =$ *gcd* $x\ y \rceil$
apply (*rule sH-whilei*, *simp-all*, *clarsimp simp*: *p2r-def*, *intro rel-hm.H-seq*)
apply (*rule H-assign*, *auto*)+
using *gcd-red-nat* **by** *auto*

end

The verification of Euclid's algorithm has been added as a simple example. Isabelle's syntax conventions require that program variables, which have been implemented as strings, are decorated with double quotes.

This verification component is correct by construction relative to Isabelle's small trustworthy core because the axiomatic extension introduced by the type class for H-monoids is made consistent with Isabelle's core by the interpretation proof with respect to the relational program semantics. The verification could have been automated further by programming tactics for verification condition generation in Isabelle, but this is not the purpose of this section. Finally, the PHL rules for the control level are cleanly separated from the data level. In fact, the relational semantics can be replaced in a modular fashion by, for instance, a denotational trace semantics of programs.

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

9. A minimalist refinement component

This section shows how the Isabelle verification component based on H-monoids can be expanded to a minimalist refinement component based on R-monoids[‡].

First, in the Isabelle code below, the type class of H-monoids is expanded to that of R-monoids. The rules of PRC are derived next within this algebra. The relational specification statement is then defined as the supremum of all the elements that satisfy the associated H-triple, and it is shown by an interpretation proof that binary relations under the operations listed form R-monoids. After that, three assignment rules are derived in the relational model; the second and third one allowing the introduction of assignments after and before a block of code (Morgan 1994). Again, the refinement laws are generic and can be replaced in a modular fashion by other denotational semantics.

theory *R-Semigroup*
  imports *H-Semigroup*

begin

class *R-monoid* $=$ *H-monoid* $+$
  fixes $R :: 'a \Rightarrow 'a \Rightarrow 'a$
  assumes *star-iso*: $x \leq y \Longrightarrow x^{\star} \leq y^{\star}$
  and *R1*: $H\ p\ (R\ p\ q)\ q$
  and *R2*: $H\ p\ x\ q \Longrightarrow x \leq R\ p\ q$

begin

lemma *R-skip*: $1 \leq R\ p\ p$
  by (*simp add*: *H-skip R2*)

lemma *R-cons*: $p \leq p' \Longrightarrow q' \leq q \Longrightarrow R\ p'\ q' \leq R\ p\ q$
  using *H-cons R1 R2* **by** *blast*

lemma *R-seq*: $(R\ p\ r) \cdot (R\ r\ q) \leq R\ p\ q$
  using *H-seq R1 R2* **by** *blast*

lemma *R-loop*: $H\ p\ q\ (p \cdot q) \Longrightarrow H\ p\ r\ (p \cdot r) \Longrightarrow (q \cdot R\ (p \cdot q)\ p)^{\star} \cdot r \leq R\ p\ (p \cdot r)$
  by (*simp add*: *H-loopi R1 R2 preo-refl*)

lemma *R-cond*:
assumes $\bigwedge x\ y.\ p \cdot (x + y) \leq p \cdot x + p \cdot y$ **and** $\bigwedge x\ y.\ x \cdot q + y \cdot q \leq (x + y) \cdot q$
shows $H\ p\ v\ (p \cdot v) \Longrightarrow H\ p\ w\ (p \cdot w) \Longrightarrow v \cdot R\ (p \cdot v)\ q + w \cdot R\ (p \cdot w)\ q \leq R\ p\ q$
  by (*simp add*: *assms H-cond R1 R2*)

end

definition *rel-R* $:: 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel$ **where** *rel-R* $P\ Q = \bigcup\{X.\ \textit{rel-hm.H}\ P\ X\ Q\}$

interpretation *rel-rm*: *R-monoid Id op* ; *op* $\cup$ *op* $\subseteq$ *rtrancl rel-R*
  by (*standard*, *auto simp add*: *rel-R-def rel-hm.H-def*, *blast*)

lemma *R-assign*: $(\forall s.\ P\ s \longrightarrow Q\ (s\ (v := e\ s))) \Longrightarrow (v ::= e) \subseteq \textit{rel-R}\ \lceil P\rceil\ \lceil Q\rceil$

‡ https://github.com/gstruth/h-semigroups

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

by (*simp add*: *H-assign rel-rm.R2*)

lemma *R-assignr*: $(\forall s.\ Q'\ s \longrightarrow Q\ (s\ (v\ := e\ s))) \Longrightarrow (\textit{rel-R}\ \lceil P \rceil\ \lceil Q' \rceil)\ ;\ (v\ ::= e) \subseteq \textit{rel-R}\ \lceil P \rceil\ \lceil Q \rceil$
by (*metis H-assign rel-hm.H-seq rel-rm.R1 rel-rm.R2*)

lemma *R-assignl*: $(\forall s.\ P\ s \longrightarrow P'\ (s\ (v\ := e\ s))) \Longrightarrow (v\ ::= e)\ ;\ (\textit{rel-R}\ \lceil P' \rceil\ \lceil Q \rceil) \subseteq \textit{rel-R}\ \lceil P \rceil\ \lceil Q \rceil$
by (*metis H-assign rel-hm.H-seq rel-rm.R1 rel-rm.R2*)

lemma *if-then-else-ref*: $X \subseteq X' \Longrightarrow Y \subseteq Y' \Longrightarrow \textit{if}\ P\ \textit{then}\ X\ \textit{else}\ Y\ \textit{fi} \subseteq \textit{if}\ P\ \textit{then}\ X'\ \textit{else}\ Y'\ \textit{fi}$
by (*auto simp*: *if-then-else-def*)

lemma *while-ref*: $X \subseteq X' \Longrightarrow \textit{while}\ P\ \textit{do}\ X\ \textit{od} \subseteq \textit{while}\ P\ \textit{do}\ X'\ \textit{od}$
by (*simp add*: *while-def rel-hm.mult-isol rel-hm.mult-isor rel-rm.star-iso*)

Once more Euclid's algorithm is used as an example. The initial specification consists of the pre-condition and post-condition used in the previous verification proof. The first step brings the specification statement in shape for introducing a while loop in the second step. The next three steps introduce the assignments in the body of the loop. The final step ties these facts together by isotonicity; it shows that Euclid's algorithm refines its specification statement.

lemma *euclid1* :
$\textit{rel-R}\ \lceil \lambda s{::}\textit{nat store}.\ s\ ''x'' = x \wedge s\ ''y'' = y \rceil\ \lceil \lambda s.\ s\ ''x'' = \textit{gcd}\ x\ y \rceil$
$\supseteq$
$\textit{rel-R}\ \lceil \lambda s.\ \textit{gcd}\ (s\ ''x'')\ (s\ ''y'') = \textit{gcd}\ x\ y \rceil\ \lceil \lambda s.\ \textit{gcd}\ (s\ ''x'')\ (s\ ''y'') = \textit{gcd}\ x\ y \wedge \neg\ s\ ''y'' \neq 0 \rceil$
by (*intro rel-rm.R-cons*, *auto simp*: *p2r-def*)

abbreviation $P\ x\ y \equiv \lceil \lambda s{::}\textit{nat store}.\ \textit{gcd}\ (s\ ''x'')\ (s\ ''y'') = \textit{gcd}\ x\ y \wedge s\ ''y'' \neq 0 \rceil$

lemma *euclid2* :
$\textit{rel-R}\ \lceil \lambda s.\ \textit{gcd}\ (s\ ''x'')\ (s\ ''y'') = \textit{gcd}\ x\ y \rceil\ \lceil \lambda s.\ \textit{gcd}\ (s\ ''x'')\ (s\ ''y'') = \textit{gcd}\ x\ y \wedge \neg\ s\ ''y'' \neq 0 \rceil$
$\supseteq$
$\textit{while}\ (\lambda s.\ s\ ''y'' \neq 0)\ \textit{do rel-R}\ (P\ x\ y)\ \lceil \lambda s.\ \textit{gcd}\ (s\ ''x'')\ (s\ ''y'') = \textit{gcd}\ x\ y \rceil\ \textit{od}$
apply (*simp only*: *while-def p2r-mult-hom*[*symmetric*])
by (*intro rel-rm.R-loop*, *auto simp*: *p2r-def rel-hm.H-def*)

lemma *euclid3* :
$\textit{rel-R}\ (P\ x\ y)\ \lceil \lambda s.\ \textit{gcd}\ (s\ ''x'')\ (s\ ''y'') = \textit{gcd}\ x\ y \rceil$
$\supseteq$
$\textit{rel-R}\ (P\ x\ y)\ \lceil \lambda s.\ \textit{gcd}\ (s\ ''z'')\ (s\ ''y'') = \textit{gcd}\ x\ y \rceil\ ;\ (''x''\ ::= (\lambda s.\ s\ ''z''))$
by (*intro R-assignr*, *simp*)

lemma *euclid4* :
$\textit{rel-R}\ (P\ x\ y)\ \lceil \lambda s.\ \textit{gcd}\ (s\ ''z'')\ (s\ ''y'') = \textit{gcd}\ x\ y \rceil$
$\supseteq$
$\textit{rel-R}\ (P\ x\ y)\ \lceil \lambda s.\ \textit{gcd}\ (s\ ''z'')\ (s\ ''x''\ \textit{mod}\ s\ ''y'') = \textit{gcd}\ x\ y \rceil\ ;\ (''y''\ ::= (\lambda s.\ s\ ''x''\ \textit{mod}\ s\ ''y''))$
by (*intro R-assignr*, *simp*)

lemma *euclid5* :
$\textit{rel-R}\ (P\ x\ y)\ \lceil \lambda s.\ \textit{gcd}\ (s\ ''z'')\ (s\ ''x''\ \textit{mod}\ s\ ''y'') = \textit{gcd}\ x\ y \rceil$
$\supseteq$
$(''z''\ ::= (\lambda s.\ s\ ''y''))$
by (*intro R-assign*, *auto simp*: *gcd-non-0-nat*)

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

```
lemma euclid-ref:
  rel-R ⌈λs::nat store. s ''x'' = x ∧ s ''y'' = y⌉ ⌈λs. s ''x'' = gcd x y⌉
    ⊇
  while (λs. s ''y'' ≠ 0)
    do
      (''z'' ::= (λs. s ''y'')) ;
      (''y'' ::= (λs. s ''x'' mod s ''y'')) ;
      (''x'' ::= (λs. s ''z''))
    od
  apply (rule dual-order.trans, subst euclid1, simp, rule dual-order.trans, subst euclid2, simp)
  apply (intro while-ref) using euclid3 euclid4 euclid5 by fast

end
```

The proof steps could have been automated further once more by programming tactics that apply the refinement laws and simplify the results of their applications.

The step-wise refinement proof of Euclid's algorithm could be translated into a verification proof as follows. The fifth, fourth and third step of the refinement proof translate directly into a Hoare-style proof for the body of the while loop by using Equation (1) and combining the results by using (HSeq). Alternatively, (RSeq) can be used first for obtaining a refinement proof for the body of the loop, and then Equation (1) can be used for translating the result into a verification proof. It can be completed by applying the rule for while loops and then the consequence rules.

Conversely, a fine-grained verification proof of Euclid's algorithm could be translated directly into a step-wise refinement proof by using the proof obligations generated by Isabelle when applying the PHL rules as pre-conditions and post-conditions in specification statements. Isotonicity laws can then be used for breaking refinement proofs into pieces, as in step three to five in the example above.

10. Counterexamples

When trying to find general algebras in which properties such as PHL or PRC are derivable, the question of counterexamples arises. This section presents counterexamples related to the most important axiomatisations in this article. Isabelle's Nitpick tool assisted in their generation. Where possible and relevant, counterexamples are given in the presence of interesting additional structure.

The first two counterexamples relate to Proposition 2.7.

Lemma 10.1.

i. In some dioid, without the left distributivity axiom, (HCond) does not hold.
ii. In some dioid, without the right distributivity axiom, (HCond) does not hold.

Proof.

i. Consider the structure with carrier set $\{a, b, 0, 1\}$, addition defined by $0 < 1 < b$, $0 < a < b$, whereas 1 and a are incomparable, and multiplication by $aa = ba = a$ and $ab = bb = b$. It is routine to check that it forms a dioid, but left distributivity fails

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

because $a(1+a) = ab = b \neq a = a1 + aa$. Then $H\,a\,1\,(a1)$, because $a \leqslant a$, $H\,a\,b\,(ab)$, because $ab = b = bab$, $H\,(a1)\,1\,a$, because $a \leqslant a$ and $H\,(ab)\,a\,a$, because $aba = a = aa$, but not $H\,a\,(11+ba)\,a$, as $a(1+ba) = b \nleq a = (1+ba)a$.

ii. Consider the structure with carrier set $\{a, b, 0, 1\}$, with addition defined by $0 < a < 1$ and $0 < b < 1$ whereas a and b are incomparable, and multiplication by $aa = ba = 0$, $ab = a$ and $bb = b$. It is routine to check that this forms a dioid, but right distributivity fails because $(a+b)b = b \neq 1 = ab + bb$. Then $H\,1\,1\,(11)$, $H\,1\,b\,b$, because $b = bb$ and $H\,1\,a\,b$, because $a = a$, but not $H\,1\,(b+a)\,b$, as $b + a = 1 \nleq b$. □

The following counterexamples relate to Proposition 3.3.

Lemma 10.2.

i. In some R-monoid, without the left distributivity axiom, (RCond) does not hold.
ii. In some R-monoid, without the right distributivity axiom, (RCond) does not hold.

Proof.

i. Consider the R-monoid with carrier set $\{a, 1\}$, partial order $a \prec 1$, multiplication by $aa = a$, iteration $a^\circ = 1^\circ = 1$ and tables for the other operations given by

R	1	a
1	1	a
a	1	1

$+$	1	a
1	1	1
a	1	1

Then $H\,1\,1\,(11)$, but $1R\,(11)\,a + 1R\,(11)\,a = a + a = 1 \nleq a \preceq R\,1\,a$. Note that addition is commutative in this counterexample.

ii. Consider the R-monoid with carrier set $\{a, b, 1\}$, partial order $a \prec 1 \prec b$, iteration $a^\circ = b^\circ = 1^\circ = 1$, multiplication defined by $ab = bb = b$ and $aa = ba = a$, and the remaining operations by

R	1	a	b
1	b	a	b
a	b	1	b
b	b	a	b

$+$	1	a	b
1	b	b	b
a	b	a	b
b	b	b	b

Then $H\,a\,b\,(ab)$ because $ab = b = bab$ and $H\,a\,1\,(a1)$, but $bR\,(ab)\,a + 1R\,(a1)\,a = b \npreceq 1 = R\,a\,a$. Note that addition is again commutative. □

In this case, a dioid-based counterexample for Lemma 10.2(i) was rather large whereas I did not succeed to find one for Lemma 10.2(ii). The following counterexamples relate again to Proposition 2.7.

Lemma 10.3.

i. In some Kleene algebra, (HCond) without reflection conditions does not hold.
ii. In some Kleene algebra, (HLoop) without reflection conditions does not hold.

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

Proof.

i. Consider the Kleene algebra with carrier set $\{a, 0, 1\}$, addition defined by $0 < a < 1$, multiplication by $aa = 0$ and $x^* = 1$ for $x \in \{a, 0, 1\}$. Then $H\,(1a)\,0\,a$ and $H\,(1a)\,1\,a$, but not $H\,1\,(a0 + a1)\,a$ because $a \not< 0 = aa$.
ii. Consider again the Kleene algebra from (1). Then $H\,(11)\,1\,1$, but not $H\,1\,((11)^*a)\,(1a)$ because $1^*a = a \not< 0 = aa$. $\square$

The next counterexamples relate to Proposition 3.3 and the discussion at the end of Section 3.

Lemma 10.4.

i. There is an R-Kleene algebra in which (RCond) without reflection conditions does not hold.
ii. There is an R-Kleene algebra in which (RLoop) without reflection conditions does not hold.
iii. There is an R-Kleene algebra in which $x \leqslant R\,p\,q \Rightarrow H\,p\,x\,q$ does not hold.

Proof.

i. Consider the R-Kleene algebra with carrier set $\{a, 0, 1\}$, addition defined by $0 < a < 1$, multiplication by $aa = 0$, $0^* = 1^* = a^* = 1$ and

R	0	a	1
0	1	1	1
a	a	1	1
1	0	0	1

Then $aR\,(1a)\,a + aR\,(1a)\,a = a \not\leqslant 0 = R\,1, a$.
ii. Consider again the R-Kleene algebra from (1). Then $1R\,(11)\,1)^*a = a \not\leqslant 0 = R\,1\,(1a)$.
iii. Consider the Kleene algebra with carrier set $\{a, 0, 1\}$, addition defined by $0 < 1 < a$, multiplication by $aa = a$, $0^* = 1^* = 1$ and $a^* = a$ and

R	0	1	a
0	a	a	a
1	0	a	a
a	0	a	a

Then $1 \leqslant a = R\,a\,1$, but $H\,a\,1\,1$ does not hold because $a1 = a \not\leqslant 1 = 11$. $\square$

The final counterexample relates to the discussion following Lemma 5.1.

Lemma 10.5. There is a pre-order $(S, \preceq)$ equipped with $+, \cdot, R : S \times S \to S$, $^\circ : S \to S$ and $1 \in S$, such that $\cdot$, $+$ and $^\circ$ are isotone, $H\,p\,(R\,p\,q)\,q$ and $H\,p\,x\,q \Rightarrow x \preceq R\,p\,q$ hold for $H\,p\,x\,q \Leftrightarrow px \preceq xq$, (RSeq) holds, but not (HSeq).

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

Proof. Consider the structure with carrier set $\{a, b, c\}$, pre-order $a \prec b \prec c$ and the other operations defined by the tables

R	a	b	c
a	c	c	c
b	c	c	c
c	c	c	c

$+$	a	b	c
a	a	a	a
b	a	a	a
c	a	a	a

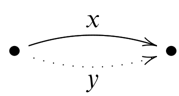
$\cdot$	a	b	c
a	a	a	b
b	a	a	b
c	b	b	b

	$^\circ$
a	b
b	b
c	b

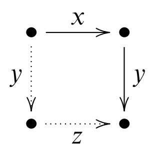
It can be checked that all conditions hold, and in addition $H\,c\,c\,c$, and $H\,c\,c\,b$, because $cc = b = cb$, but $H\,c\,(cc)\,b$ fails because $c(cc) = cb = b \not\prec a = bb = (cc)b$. □

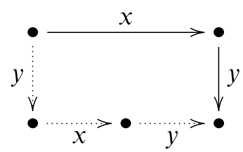
11. Proposition 2.7 by diagram chase

Depict $x \preceq y$ by the following diagram:

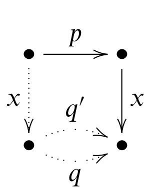


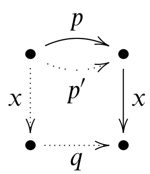
Accordingly, $H\,x\,y\,z$ and $H\,x\,y\,(xy)$ are depicted as follows:

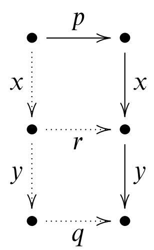


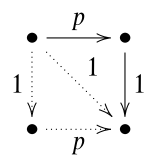


The rules (HCons1), (HCons2), (HSeq) and (HSkip) are then derived as follows:









The next two export rules are helpful for deriving (HLoop) and (HCond).

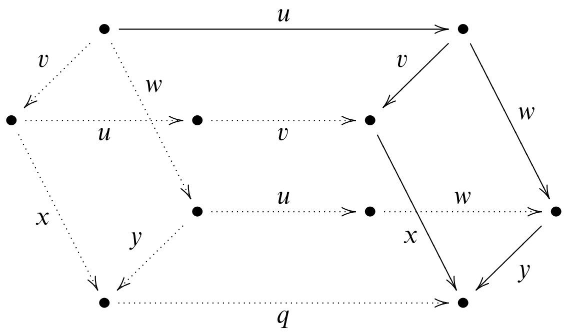
$$H\,p\,r\,(pr) \wedge H\,(pr)\,x\,q \Rightarrow H\,p\,(rx)\,q, \tag{HExp1}$$

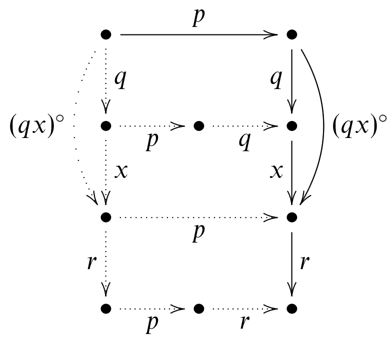
$$H\,q\,r\,(qr) \wedge H\,p\,x\,q) \Rightarrow H\,p\,(xr)\,(qr). \tag{HExp2}$$

https://doi.org/10.1017/S096012951700007X Published online by Cambridge University Press

The first one is obtained by substituting r for x and pr for r and y for x in (HSeq), the second one by substituting q for y and r and qr for q in (HSeq).

Finally, (HCond) is obtained with (HExp1) and (HLoop) with (HExp1) and (HExp2):





12. Conclusion

The range of Hoare logics and refinement calculi covered by H- and R-semigroups requires further exploration. First of all, examples for Hoare logics over (non-idempotent) ordered semirings beyond the parallel rules for multirelations in Section 7 remain to be found. Matrix semirings might come to mind, but the definition of matrix orderings is rather intricate and convincing examples are so far missing.

Hoare logics for probabilistic and quantum programs (e.g. Chadha et al. 2006; den Hartog and de Vink 2002) have already been developed, but further work is needed for relating them with H-semirings, defining refinement calculi for them and implementing components for them in Isabelle.

For the probabilistic case, a concise denotational semantics, e.g. relations between probability (sub)distributions, and the operation of probabilistic choice, which is added to non-deterministic choice, sequential composition and finite iteration, must be included in the algebra. From an abstract point of view, values are mapped to distributions or subdistributions of values, which resembles the multirelations in Section 7, but leads to more pleasant algebraic properties.

Quantum Hoare logics further adapt the probabilistic approach. They seem to be based predominantly on predicate transformer semantics, which encode Hoare triples differently. Hence, a relational semantics should be built before an integration. Such considerations are left as interesting avenues for future work.

Acknowledgements

I am grateful to Alasdair Armstrong and Victor Gomes for their collaboration on building verification components in Isabelle/HOL and shaping the general method that underlies Sections 8 and 9; and in particular to Victor for discussions on the relationship between verification and refinement proofs.

References

Armstrong, A., Gomes, V.B.F. and Struth, G. (2014). Kleene algebra with tests and demonic refinement algebras. *Archive of Formal Proofs*. https://www.isa-afp.org/entries/KAT_and_DRA.shtml.

Armstrong, A., Gomes, V.B.F. and Struth, G. (2016). Building program construction and verification tools from algebraic principles. *Formal Aspects of Computing* **28** (2) 265–293.

Armstrong, A., Struth, G. and Weber, T. (2013a). Program analysis and verification based on Kleene algebra in Isabelle/HOL. In: Blazy, S., Paulin-Mohring, C. and Pichardie, D. (eds.) *ITP 2013*, Lecture Notes in Computer Science, vol. 7998, Springer, 197–212.

Armstrong, A., Struth, G. and Weber, T. (2013b). Kleene algebra. *Archive of Formal Proofs*. https://www.isa-afp.org/entries/Kleene_Algebra.shtml.

Chadha, R., Mateus. P., and Sernadas, A. (2006). Reasoning about imperative quantum programs. *ENTCS* **158** 19–39.

den Hartog, J. and de Vink, E.P. (2002). Verifying probabilistic programs using a Hoare like logic. *International Journal of Foundations of Computer Science* **13** (3) 315–340.

Desharnais, J. and Struth, G. (2011). Internal axioms for domain semirings. *Science of Computer Programming* **76** (3) 181–203.

Furusawa, H. and Struth, G. (2015). Concurrent dynamic algebra. *ACM TOCL* **16** (4) 30:1–30:38.

Kozen, D. (1997). Kleene algebra with tests. *ACM TOPLAS* **19** (3) 427–443.

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

Meijer, E., Fokkinga, M. and Paterson, R. (1991). Functional programming with bananas, lenses, envelopes and barbed wire. In: Hughes, J. (ed.) *Functional Programming Languages and Computer Architecture*, Lecture Notes in Computer Science, vol. 523, Springer, 124–144.

Morgan, C. (1994). *Programming from Specifications*, 2nd edition, Prentice Hall.

Nipkow, T., Paulson, L.C. and Wenzel, M. (2002). *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, Lecture Notes in Computer Science, vol. 2283, Springer.

Peleg, D. (1987). Concurrent dynamic logic. *Journal of the ACM* **34** (2) 450–479.

Pous, D. (2013). Kleene algebra with tests and Coq tools for while programs. In: Blazy, S., Paulin-Mohring, C. and Pichardie, D. (eds.) *ITP 2013*, Lecture Notes in Computer Science, vol. 7998, Springer, 180–196.

von Wright, J. (2004). Towards a refinement algebra. *Science of Computer Programming* **51** (1–2) 23–45.