Under consideration for publication in Formal Aspects of Computing

Generic Composition

Yifeng Chen
Department of Mathematics and Computer Science,
University of Leicester, Leicester, UK

Keywords: Predicative semantics; Healthiness condition; Modal logic; Parallelism

Abstract. This paper presents a technique called *generic composition* to provide a uniform basis for modal operators, sequential composition, different kinds of parallel compositions and various healthiness conditions appearing in a variety of semantic theories. The weak inverse of generic composition is defined. A completeness theorem shows that any predicate can be written in terms of generic composition and its weak inverse. A number of algebraic laws that support reasoning are derived.

1. Introduction

The modelling of programs as predicates coupled with the view of observation-oriented semantics simplifies the semantic interface, benefits from the expressive power of set theory and logic, and allows refinement to be expressed as implication. Such predicative semantics corresponds directly to relational semantics [Tar41, Heh84, Hoa85b, HH98] and has been shown to span sequential (imperative, logic and functional) and parallel (synchronous and asynchronous) programming paradigms. This paper helps unify those various applications of the theory by defining an operator that unifies many forms of composition which are found in those programming paradigms.

We take a set-theoretic view of predicates. A predicate is a set of assignments, each of which is a mapping from variables to their values. If a variable does not appear free in a predicate, its value is arbitrary. Conjunction and disjunction become intersection and union respectively. Infinitely-many predicates are allowed in a conjunction (or disjunction). Other connectives, quantifiers and substitution can be easily defined as functions on predicates.

Predicates form a complete lattice ordered by set containment, which corresponds to the refinement ordering in formal program derivation [BS89]. The bottom and top of the complete lattice are the constant predicates *true* (i.e. the full set of assignments) and *false* (i.e. the empty set) respectively. The least upper bound and the greatest lower bound are conjunction and disjunction respectively. *Two predicates are equal if and only if they correspond to the same set of assignments.*

A program can be represented as a predicate on undashed and dashed variables, which denote the

---

Correspondence and offprint requests to: Yifeng Chen, Department of Mathematics and Computer Science, University of Leicester, University Road, Leicester LE1 7RH, UK. Email: Y.Chen@mcs.le.ac.uk

observation on the initial and final states of a computation respectively. For example, the predicate $(x' = x+1)$ represents a sequential program $x := x+1$ that increases the value of program variable x by 1.

An operator (or program composition) is a function on predicates. For example, the nondeterministic choice between two predicates (which arises from abstraction) is their disjunction, and their sequential composition is defined as a relational composition:

$$A \,;\, B \;\widehat{=}\; \exists x_0 \cdot (A[x_0/x'] \wedge B[x_0/x]) \tag{1}$$

where x_0 is a 'fresh' variable, and substitution $A[x_0/x']$ means to replace all occurrences of x' by x_0.

In order to distinguish terminating and nonterminating programs, Hoare and He introduced two special variables ok and ok'. The former denotes the proper start of a computation, while the latter denotes its successful termination. The assignment statement $x := x+1$ then becomes a predicate $ok \Rightarrow (x' = x+1 \wedge ok')$ on four variables. Not all predicates on the four variables are valid computations. For example, the predicate $(\neg ok \wedge x' = 1)$ sets the program variable x to be 1 even when the computation has not started. Such a predicate does not correspond to any realistic computation and thus should be excluded from the semantic space. In general, Hoare and He pointed out that a valid sequential computation (also called a *design*) has a normal form:

$$b \vdash P \;\widehat{=}\; (b \wedge ok) \Rightarrow (P \wedge ok') \tag{2}$$

where b is the weakest precondition on an initial state leading to termination, and P is the postcondition relating the initial and final states after termination.

The basic form of parallelism tackled by Hoare and He is called *disjoint parallelism.* If ok and ok' are the *only* variables shared by two designs $A \;\widehat{=}\; b \vdash P$ and $B \;\widehat{=}\; c \vdash Q$, their parallel composition is defined by $A \parallel B \;\widehat{=}\; (b \wedge c \vdash P \wedge Q)$, which informally means that if the computation has started from an initial state satisfying both preconditions, then the parallel composition must terminate into a state satisfying both postconditions.

Following further the work of Hoare and He [HH98] a more general notion of parallelism called *parallel by merge* can be defined on disjoint parallelism. In order to model various different parallelism, a further parameter, the predicate M was added. M, called *merge*, is composed sequentially after the 'disjoint' composition of designs $A \parallel_M B \;\widehat{=}\; A \parallel B \,;\, M$. Each parameter M determines a specific parallel composition.

If two designs A and B share not only ok and ok' but also a pair of variables m and m', then their postconditions can no longer be placed directly in conjunction without interfering with each other. The shared dashed variable m' must be re-labeled as $\mathtt{0}.m'$ for A and $\mathtt{1}.m'$ for B in order to avoid naming conflict. The digital prefixes of variables $\mathtt{0}.m$ and $\mathtt{1}.m$ are merely syntactical decoration (see Section 2). Merge M then combines the re-labeled values $\mathtt{0}.m'$ and $\mathtt{1}.m'$ from the designs and produces the value of m' for their composition:

$$A \parallel_M B \;\widehat{=}\; (A[\mathtt{0}.m'/m'] \parallel B[\mathtt{1}.m'/m'])_{+m} \,;\, M\,.$$

where $M = M(ok, m, \mathtt{0}.m, \mathtt{1}.m, m', ok')$ is a predicate on six variables and P_{+m} denotes $(P \wedge m = m')$.

Properties of parallel-by-merge compositions are determined by the properties of the merge. If the merge M is *commutative* in the sense that it remains the same if the two variables $\mathtt{0}.m$ and $\mathtt{1}.m$ are swapped:

$$M \;=\; M[\mathtt{1}.m/\mathtt{0}.m,\; \mathtt{0}.m/\mathtt{1}.m]\,,$$

then their parallel composition is also commutative. If the merge M is *associative* in this sense

$$M_3 \;=\; M_3[\mathtt{1}.m/\mathtt{0}.m,\; \mathtt{2}.m/\mathtt{1}.m,\; \mathtt{0}.m/\mathtt{2}.m]$$

where M_3 is a three-way merge

$$M_3 \;\widehat{=}\; \exists x \exists t \cdot M(ok, m, \mathtt{0}.m, \mathtt{1}.m, x, t) \wedge M(t, m, x, \mathtt{2}.m, m', ok')\,,$$

then their parallel composition is also associative. Here, M_3 is essentially a relational composition of two M merges. The variables m' and ok' of the first merge are connected to the variables m and ok of the second merge respectively, and the interface variables x and t become hidden by existential quantifiers. To better understand these definitions, please refer to Chapter 7 of [HH98].

Defining parallelism with sequential composition may become inconvenient in some parallel languages. The interface between designs and merge always includes all shared dashed variables. If the value of a shared dashed variable is supposed to be identical on both sides, there is no need to include it in the interface.

On the other hand, the undashed variables of a program may be related to the dashed variables of another program in parallel, and that means parallel programs do not always share the same initial state. These difficulties can be solved if we allow the interface to be set freely and explicitly.

Using a normal form like (2) is one way of defining a semantic space. Another technique proceeds top-down by imposing, without explicit construction, the laws that valid predicates satisfy. Such laws are also called *healthiness conditions* and were first introduced by Dijsktra for predicate-transformer semantics [Dij75]. Hoare and He [HH98] explored the use of healthiness conditions in the context of predicative semantics. The role of healthiness conditions in semantics is very similar to the role of axioms in logic. For example, a predicate on four variables ok, ok', x and x' is a design iff it satisfies three healthiness conditions, one of which $A = (ok \Rightarrow A)$ states that if a valid computation has not started (i.e. $\neg ok$), its behaviour is arbitrary. It corresponds to an axiom schema $A \leftrightarrow (ok \rightarrow A)$ in logic where A can be any well-formed formula. In logic, a formal system becomes more concrete if new axioms are added; similarly in predicative semantics, a semantic model becomes more specialised if new healthiness conditions are imposed.

Healthiness conditions are semantic notions. They are free of most syntactical restrictions and hence more flexible than axioms. One of Dijkstra's healthiness conditions for *weakest liberal precondition* states that an operator (or a predicate transformer) f is *universally disjunctive* if it satisfies the condition:

$$f(\bigvee \mathcal{A}) \;=\; \bigvee \{ f(A) \mid A \in \mathcal{A} \} \quad \text{for any set } \mathcal{A} \text{ of predicates.}$$

Such an operator can be represented as a binary relation R between *assignments*: $x \; R \; y \;\widehat{=}\; y \in f(\{x\})$ where $\{x\}$ is a singleton predicate containing just one assignment. On the other hand, any relation between assignments also defines a universally disjunctive operator $f(A) = \{ y \mid x \in A,\, x \; R \; y \}$. Unfortunately universal disjunctivity corresponds to no axiom in that the set $\mathcal{A}$ may be infinite while the structure of any well-formed formula must be finite. For a counterexample, the operator

$$f(A) \;\widehat{=}\; \begin{cases} A & \text{if } A \text{ is a finite set of assignments} \\ \mathit{true} & \text{otherwise} \end{cases} \tag{3}$$

is finitely disjunctive $f(A \vee B) = f(A) \vee f(B)$ and strict $f(\mathit{false}) = \mathit{false}$, but not universally disjunctive. Finite disjunctivity and strictness correspond to the two axioms of basic modal logic and are incomplete with respect to Kripke's relational semantics [JT51, Kri63, Che80, Fin74, Kar98].

In predicative semantics, a healthiness condition is normally expressed as a fixpoint identity $A = f(A)$ where f is an operator called *healthiness function*. A healthiness condition $A = f(A)$ is *idempotent* if $f(A) = f \circ f(A)$ holds for any predicate A and is *monotonic* if, for any A and B, $A \supseteq B$ implies $f(A) \supseteq f(B)$. A monotonic and idempotent healthiness function characterises a link between semantic models by transforming each unhealthy predicate to a healthy one in a more detailed model and maps each complete-lattice semantic model to a subset complete lattice containing exactly the fixpoints of the function. These advantages suggest that we should write healthiness conditions in the above way whenever it is possible.

The simplest forms of healthiness conditions are based on disjunction and conjunction. The condition $A = (\neg ok \vee A)$ is an example of disjunctive healthiness condition. Most healthiness conditions cannot be written in such a simple form. For example, one of the healthiness conditions of designs states that if a computation does not terminate (i.e. $\neg ok'$), then its final state is not observable. Although this condition is neither disjunctive nor conjunctive, it can be defined using a sequential composition $A = A \,;\, \mathrm{I\!I}$ where $\mathrm{I\!I}$ denotes the program of no operation and is the unit of sequential composition. Reasoning about such healthiness conditions is well supported by sequential calculus [Tar41]. For example, because of sequential composition's associativity, it becomes trivial to check the idempotence of the above healthiness condition by showing that $\mathrm{I\!I} \,;\, \mathrm{I\!I} = \mathrm{I\!I}$.

The interface of a sequential composition always connects all dashed variables of the first predicate with the undashed variables of the second predicate. That means many important healthiness conditions cannot be defined with sequential composition. For example, another healthiness condition of designs

$$A[\mathit{false}/ok'] \;\Rightarrow\; A[\mathit{true}/ok']$$

states that the final value of ok' is arbitrary if the computation does not terminate. This condition is not even written in the form $A = f(A)$ (although we will be able to do so in Section 5). Another example is the healthiness condition of history independence in predicative semantics of CSP [HH98]:

$$A \;=\; \bigvee_s A\,[s/tr,\; s^\frown(tr' - tr)/tr']$$

where tr (or tr') is the trace of events before the start (or end) of the processes. We assume that $tr \leqslant tr'$.

This healthiness condition is rather difficult to understand. In fact, it states that the behaviour of a process does not depend on the history before its start. These healthiness conditions cannot be defined using sequential composition. More systematic higher-level techniques are needed.

From the above discussions, we can see that predicative semantics is heavily involved with manipulation of operators on predicates. Direct low-level reasoning with predicate calculus is too complicated in many circumstances. Sequential composition helps simplify reasoning in some examples, but its applicability is limited due to the fixation of its interface.

In this paper, we propose a new composition called *generic composition* with which a variety of program compositions and healthiness conditions can be defined. Its main difference from sequential composition lies in the fact that the interface of the composition can be explicitly set. The weak inverse of generic composition is also defined using a Galois connection. A completeness theorem shows that any predicate can be written in terms of generic composition and its weak inverse, thus showing the technique is general.

Modal logic was first proposed to describe *possibility* and *necessity* in logic. Two *modal operators* $\Diamond P$ and $\Box P$ are normally used. If P is a predicate, $\Diamond P$ informally means that "*the predicate P may be true*", while $\Box P$ informally means that "*the predicate P must be true*". The most popular semantics is Kripke's relational semantics [Kri63], based on which we can define the modal operator of possibility as a generic composition of a predicate and the converse of accessibility relation and the operator of necessity as a weak inverse. The correspondence between modal axioms and frame conditions of Kripke's semantics can then be established using algebraic laws of generic composition and its weak inverse.

A kind of *parallel-via-medium* composition is also defined using generic composition. Both undashed and dash variables are allowed in the composition's interface, which is linked to a parameter predicate called the *medium*. The composition is more general than parallel-by-merge composition in that parallel programs no longer need to share the same initial state, and their the interface can include only those shared variables with different proposed meanings on the two sides. For example, conjunction becomes a special case of the composition whose interface is empty.

If two predicates are identical, their parallel-via-medium composition becomes a *second-order generic composition*. Such a composition 'simulates' a predicate twice and then presents the combined results via a medium predicate. If up to infinitely-many copies of a predicate are allowed, we can then define a kind of *higher-order generic composition*. Such a composition simulates a predicate infinitely-many times and presents the combined results via a medium predicate. Neither kind is finitely disjunctive, but they are very useful in expressing cardinality, limit points and other advanced properties.

Using generic compositions, we can systematically define a series of healthiness conditions from a variety of models, such as sequential designs [HH98], CSP [Hoa85a], Game Semantics [Abr99], probabilistic programming [HSM97], safety and liveness properties [AS85] and BSP [Val90].

Section 2 introduces generic composition and its weak inverse. In Section 4 we use the technique to define general modal operators, sequential and parallel compositions and healthiness conditions. Higher-order generic compositions are also introduced. In Section 5, we apply the techniques to yield parallel compositions and healthiness conditions from a variety of computational models.

2. Generic composition and its weak inverse

As motivation we observe that various program composition tend to have a similar pattern. For example,

- in relational semantics [Tar41, Mad96], the sequential composition of two binary relations R and S is defined by

  $$x\ (R \mathbin{;} S)\ z \ \widehat{=}\ \exists y \cdot (x\ R\ y\ \wedge\ y\ S\ z)\ ;$$

- in CSP [Hoa85a], the parallel composition between two communicating processes is defined by

  $$((c!v \to P) \parallel (c?x \to Q(x))) \setminus C\ =\ (P \parallel Q(v)) \setminus C$$

  where communications along channel c are concealed;

- in CCS [Mil89], two components A and B with ports c and $\overline{c}$ respectively can be combined, and they communicate with each other through the ports according to a rule

$$\frac{A \xrightarrow{c} A',\ B \xrightarrow{\overline{c}} B'}{A|B \xrightarrow{\tau} A'|B'}$$

where each communication becomes a silent action not directly observable from the outside.

In general, a system combining two components A and B demonstrates the behaviours of both A *and* B, and communications between the two components are normally delivered through an interface hidden from the outside.

A *generic composition* written as $A :_x B$ is a general form of such compositionality based on first-order logic. Here x is a *non-overlined* variable called an *interface*, and A and B can be *any* kind of predicates. The generic composition connects the non-overlined variable x of A with the overlined variable $\overline{x}$ of B and hides them with an existential quantifier. Note that the overlined variable $\overline{x}$ of A and the non-overlined variable x of B are still observable from the outside. The two predicates must agree on the value of any other variable. A generic composition can be viewed as a kind of sequential composition on a designated interface.

Definition 1. $A :_x B \ \widehat{=}\ \exists x_0 \cdot A[x_0/x] \wedge B[x_0/\overline{x}]$ where x_0 is a 'fresh' variable.

For example, the generic composition $(x = 5) :_x (\overline{x} \leqslant x)$ equals $(5 \leqslant x)$, and the generic composition $(\overline{x} \leqslant x) :_x (\overline{x} < x)$ equals $(\overline{x} < x)$.

We assume that a variable may split into a *list* of variables. For example, we may have a generic composition $A :_{(y,y')} B$ whose interface is a pair of undashed variable y and dashed variable y'. In particular, if the interface is an empty list $()$, the above definition collapses to conjunction.

In this paper, we shall use $A = \exists x \cdot A$ to denote the condition that a predicate A *does not depend on* the variable x. For example, neither of the predicates $(y > 2)$ or $(x = x \wedge y > 2)$ depends on x. In the latter example, x is actually a free variable, although it does not state anything interesting about x.

Overlining is just a kind of variable decoration. We assume that if $x = (x_1, x_2, \cdots, x_n)$ is a list of undecorated variables, then $\overline{x} = (\overline{x}_1, \overline{x}_2, \cdots, \overline{x}_n)$ is a list of overlined variables, $x' = (x'_1, x'_2, \cdots, x'_n)$ is a list of dashed variables, $0.x = (0.x_1, 0.x_2, \cdots, 0.x_n)$ is a list of indexed variables, and $0.\overline{x} = (0.\overline{x}_1, 0.\overline{x}_2, \cdots, 0.\overline{x}_n)$ is a list of overlined indexed variables. A predicate is called *normal*, if it does not depend on overlined variables. Variable decoration is merely a syntactical convention to relate variables.

The following propositions indicate some simple cases of generic composition and are proved by routine predicate manipulations.

Proposition 1. $A \wedge B \ =\ A :_{()} B$.

Proposition 2. $\exists x \cdot A \ =\ A :_x \mathit{true}$.

Proposition 3. $A[e/x] \ =\ A :_x (\overline{x} = e)$ where e is an expression independent of $\overline{x}$.

Some arithmetic operators have inverse operations. For example $-$ is the inverse of $+$ in the sense that if $x + p = q$, we can restore x by calculating $q - p$.

Operators such as sequential composition do not have exact inverses. In general the first predicate X cannot be restored from a sequential composition $X \fatsemi A$ by calculation. Nevertheless it is possible to define sequential composition's *weak* inverse $B \,/\, A$ (called the *weakest prespecification* [HH86]), which is the weakest specification refining B if followed by A:

$$B \,/\, A \ \widehat{=}\ \bigvee \{ X \mid (X \fatsemi A) \subseteq B \}.$$

Similar weak inverse can be defined for generic composition. Let A and B be two predicates. The *weakest pre-predicate* composition $B \,/_x\, A$ is the weakest predicate X such that $X :_x A \subseteq B$. The following Galois connection provides a neat definition as a weak inverse in the sense of adjunctions.

Definition 2. $X :_x A \subseteq B$ iff $X \subseteq B \,/_x\, A$.

Theorem 4. $B \,/_x\, A \ =\ \neg(\neg B :_x \lceil A \rfloor_x)$.

Here $\lceil A \rfloor_x \ \widehat{=}\ A[x, \overline{x}/\overline{x}, x]$ is the converse of A (on x). If the variable list x is empty, then $\lceil A \rfloor_x = A$. Now we can express other primitive operators. Again the proofs are routine predicate manipulations.

Proposition 5. $A \Rightarrow B \;=\; B \,/_{()}\, A\,.$

Proposition 6. $\neg A \;=\; \mathit{false} \,/_{()}\, A\,.$

Proposition 7. $A \vee B \;=\; B \,/_{()}\, (\mathit{false} \,/_{()}\, A)\,.$

Proposition 8. $\forall x \cdot A \;=\; A \,/_x\, \mathit{true}\,.$

Theorem 9 (Completeness). Any normal predicate can be written in terms of generic composition and its weak inverse using only constant predicates.

This is justified by the fact that by Propositions 1 to 7 all logical operators of predicate calculus can be eliminated from the formula of a predicate. The completeness theorem shows that generic composition and its weak inverse are as expressive as first-order predicate calculus.

Predicates dependent on overlined variables are convertible using the converse operator and substitution. For example, an existential quantifier $\exists \overline{x} \cdot A$ can be expressed as $\lceil \lceil A \rfloor_x :_x \mathit{true} \rfloor_x\,.$

Corollary 10. Any predicate can be written in terms of generic composition, its weak inverse and the converse operator using only constant predicates.

3. Laws

Generic composition and its weak inverse enjoy some powerful algebraic laws. Let x, y and z be three *lists* of variables, and A, B and C be three predicates. We use $x \diamond y$ to denote the *disjointness* of two variable lists in the sense that they do not share any variable. Proofs are routine manipulations and thus omitted.

Any Galois connection satisfies two standard laws.

Law 1 (Galois propoerties).

(1) $A \;\subseteq\; (A :_x B) \,/_x\, B$
(2) $(B \,/_x\, A) :_x A \;\subseteq\; B\,.$

Generic composition and its weak inverse are strict in the sense that they preserve *false* and *true* respectively.

Law 2 (Strictness).

(1) $\mathit{false} :_x A \;=\; \mathit{false}$
(2) $\mathit{true} \,/_x\, A \;=\; \mathit{true}\,.$

The compositions enjoy various distributivity laws. Note that these laws can be generalised to universal distributivity.

Law 3 (Distributivity).

(1) $A :_x (B \vee C) \;=\; (A :_x B) \vee (A :_x C)$
(2) $(B \vee C) :_x A \;=\; (B :_x A) \vee (C :_x A)$
(3) $A \,/_x\, (B \vee C) \;=\; (A \,/_x\, B) \wedge (A \,/_x\, C)$
(4) $(B \wedge C) \,/_x\, A \;=\; (B \,/_x\, A) \wedge (C \,/_x\, A)\,.$

The compositions distribute through existential or universal quantifier.

Law 4 (Distributivity of quantifiers). Assume that $x \diamond y$ and $A = \exists y \cdot A$. Then

(1) $A :_x (\exists y \cdot B) \;=\; \exists y \cdot (A :_x B)$
(2) $(\exists y \cdot B) :_x A \;=\; \exists y \cdot (B :_x A)$
(3) $A \,/_x\, (\exists y \cdot B) \;=\; \forall y \cdot (A \,/_x\, B)$
(4) $(\forall y \cdot B) \,/_x\, A \;=\; \forall y \cdot (B \,/_x\, A)\,.$

If the interfaces are identical, generic composition and its weak inverse become associative. Note that the predicates A, B and C may depend on both x and $\overline{x}$.

Law 5 (Associativity).

(1) $(A :_x B) :_x C \;=\; A :_x (B :_x C)$
(2) $(A \,/_x\, B) \,/_x\, C \;=\; A \,/_x\, (C :_x B)$.

The following law reveals an interesting precondition for general associativity when two interfaces are not identical. Law 5(1) is a special case of Law 6 when y and z are empty lists.

Law 6 (Associativity′). Assume that $x \diamond y$, $y \diamond z$, $z \diamond x$, $A = \exists z \cdot A$ and $C = \exists \overline{y} \cdot C$. Then

$$(A :_{(y,x)} B) :_{(x,z)} C \;=\; A :_{(y,x)} (B :_{(x,z)} C) .$$

The identity predicate $(\overline{x} = x)$ is the unit of generic composition and its weak inverse.

Law 7 (New interface).

(1) $A \;=\; A :_x (\overline{x} = x)$
(2) $A \;=\; (\overline{x} = x) :_x A$
(3) $A \;=\; A \,/_x\, (\overline{x} = x)$.

Law 8 (Disjunction elimination).

(1) $A \vee (A :_x B) \;=\; A :_x (\overline{x} = x \vee B)$
(2) $(A :_x B) \vee B \;=\; (\overline{x} = x \vee A) :_x B$.

Law 9 (Conjunction elimination).

$$A \wedge (A \,/_x\, B) \;=\; A \,/_x\, (\overline{x} = x \vee B) .$$

Law 10 (Conjunction merge). Assume $C = \exists x \cdot C$ and $D = \exists \overline{x} \cdot D$. Then

(1) $(A :_x B) \wedge C \;=\; (A \wedge C) :_x B$
(2) $(A :_x B) \wedge D \;=\; A :_x (B \wedge D)$.

Law 11 (Disjunction merge). Assume $C = \exists x \cdot C$. Then

$$(A \,/_x\, B) \vee C \;=\; (A \vee C) \,/_x\, B .$$

The interface of a generic composition can be replaced by another variable list including the same number of variables.

Law 12 (Interface re-labeling). Assume $A = \exists y \cdot A$ and $B = \exists \overline{y} \cdot B$. Then

(1) $A :_x B \;=\; A[y/x] :_y B[\overline{y}/\overline{x}]$
(2) $A \,/_x\, B \;=\; A[y/x] \,/_y\, B[\overline{y}/x]$.

If the predicates do not depend on a variable, then the variable can be either included in or excluded from the interface.

Law 13 (Interface expansion). Assume $A = \exists y \cdot A$, $B = \exists \overline{y} \cdot B$ and $C = \exists y \cdot C$. Then

(1) $A :_x B \;=\; A :_{(x,y)} B$
(2) $A \,/_x\, C \;=\; A :_{(x,y)} C$.

An interface can be extended or reduced.

Law 14 (Interface extension). Assume $x \diamond y$, $B = \exists \overline{y} \cdot B$ and $C = \exists y \cdot C$. Then

(1) $A :_x B \;=\; A :_{(x,y)} (\overline{y} = y \wedge B)$
(2) $C :_x A \;=\; (\overline{y} = y \wedge C) :_{(x,y)} A$
(3) $A \,/_x\, B \;=\; A \,/_{(x,y)}\, (\overline{y} = y \wedge B)$.

The following law describes a sufficient condition under which two interfaces x and y do not interfere with each other and hence are commutative. The definition of parallel-via-medium composition in Section 4 relies on this law.

Law 15 (Interface commutativity). Assume $x \diamond y$, $B = \exists y \cdot B$, $C = \exists x \cdot C$, $D = \exists \overline{y} \cdot D$ and $E = \exists \overline{x} \cdot E$. Then

$$\begin{array}{lrcl} (1) & (A :_x B) :_y C & = & (A :_y C) :_x B \\ (2) & E :_y (D :_x A) & = & D :_x (E :_y A) \\ (3) & (A /_x B) /_y C & = & (A /_y C) /_x B\ . \end{array}$$

Law 16 is a special case of Law 15 when y is empty.

Law 16 (Commutativity with connectives). Assume $C = \exists x \cdot C$ and $E = \exists \overline{x} \cdot E$. Then

$$\begin{array}{lrcl} (1) & (A :_x B) \wedge C & = & (A \wedge C) :_x B \\ (2) & E \wedge (D :_x A) & = & D :_x (E \wedge A) \\ (3) & C \Rightarrow (A /_x B) & = & (C \Rightarrow A) /_x B\ . \end{array}$$

Two independent compositions can be conjoined if they do not interfere with each other.

Law 17 (Interface conjoining). Assume $x \diamond y$, $B = \exists y \overline{y} \cdot B$ and $C = \exists x \overline{x} \cdot C$. Then

$$\begin{array}{lrcl} (1) & (A :_x B) :_y C & = & A :_{(x,y)} (B \wedge C) \\ (2) & C :_y (B :_x A) & = & (C \wedge B) :_{(x,y)} A \\ (3) & (A /_x B) /_y C & = & A /_{(x,y)} (B \wedge C)\ . \end{array}$$

If the interfaces of two generic composition in a conjunction have no interference with each other, they can be merged into a joint interface.

Law 18 (Interface merge). Assume $A = \exists y \cdot A$, $B = \exists x \cdot B$, $C = \exists \overline{y} \cdot C$, $D = \exists \overline{x} \cdot D$, $E = \exists x \cdot E$ and $F = \exists x \cdot F$. Then

$$\begin{array}{lrcl} (1) & (A :_x C) \wedge (B :_y D) & = & (A \wedge B) :_{(x,y)} (C \wedge D) \\ (2) & (A /_x E) \vee (B /_y F) & = & (A \vee B) /_{(x,y)} (E \wedge F)\ . \end{array}$$

4. Defining semantic operators

General operators

Any universally disjunctive operator f on normal predicates can be represented as a generic composition:

$$f(A) \;=\; A :_x B$$

where x is the list of all variables and $B \mathrel{\widehat{=}} \bigvee_{x_0} (\overline{x} = x_0 \;\wedge\; f(x = x_0))$. If the operator is only disjunctive for non-empty set of normal predicates, we can still represent it using a disjunction:

$$f(A) \;=\; (A :_x B) \vee B_0$$

where $B_0 \mathrel{\widehat{=}} f(\mathit{false})$.

If an operator $f(X)$ is universally conjunctive, then $\neg f(\neg X)$ is universally disjunctive. The operator can be represented as the weak inverse of a generic composition:

$$f(A) \;=\; A /_x C$$

where $C \mathrel{\widehat{=}} \lceil \bigvee_{x_0} (\overline{x} = x_0 \wedge \neg f(\neg (x = x_0)) \rfloor_x$. If the operator is only conjunctive for non-empty set of normal predicates, we can still represent it using a conjunction:

$$f(A) \;=\; (A /_x C) \wedge C_0$$

where $C_0 \mathrel{\widehat{=}} f(\mathit{true})$.

Note that if the actual predicate B (or C) keeps some non-overlined variables unchanged from its corresponding overlined variables, the interface can be reduced using Law 14.

Modal operators

Modal operators of possibility and necessity can be defined according to Kripke's relational semantics [Kri63] as follows:

Definition 3. $\Diamond P \;\widehat{=}\; P :_x \lceil R \rfloor_x$
$\Box P \;\widehat{=}\; P \,/_x\, R$.

In the above definitions, the special variable x is called a *modal variable*. The predicate R corresponds to the *accessibility relation* in Kripke's semantics. Each R defines a pair of modal operators. Reasoning about the correspondence between axioms and *frame conditions* is supported by the algebraic laws of the last section. More details can be found in [CL02].

Sequential composition

A sequential program is a normal predicate on a list x of undashed variables and a corresponding list x' of dashed variables. The sequential composition $A \fatsemi B$ of two programs A and B connects the dashed variables of A with undashed variables of B.

Definition 4. $A \fatsemi B \;\widehat{=}\; A :_{x'} (B :_x \overline{x} = \overline{x}')$.

The two generic compositions are commutative according to Law 15(2).

Proposition 11. $A \fatsemi B \;=\; B :_x (A :_{x'} \overline{x} = \overline{x}')$.

Parallel composition via medium

In a parallel-via-medium composition, two predicates A and B are connected by a parameter predicate M called the *medium* through a list x of variables called the *interface*.

Definition 5. $A \overset{M}{\|}_x B \;\widehat{=}\; (A[\mathtt{0}.x/x] \wedge B[\mathtt{1}.x/x]) :_{(\mathtt{0}.x,\mathtt{1}.x)} M$.

The variable list x is first re-labeled so that the two predicates can be placed in conjunction without interfering with each other. We assume that the two predicates do not depend on the variable lists $\mathtt{0}.x$ and $\mathtt{1}.x$; otherwise, Law 12 can be used to re-label the interface with some 'fresh' variable list to avoid interference. The digital prefixes are merely variable decorations.

To make a parallel-via-medium composition commutative and associative, the medium predicate must satisfy some conditions. The convention $\Pi_i \;\widehat{=}\; (i.\overline{x} = x)$ will be used in the following theorems. For example, we have $M = \Pi_0 \overset{M}{\|}_x \Pi_1$.

Theorem 12. $A \overset{M}{\|}_x B \;=\; B \overset{M}{\|}_x A$ for any A and B, iff M is commutative in this sense:

$\Pi_0 \overset{M}{\|}_x \Pi_1 \;=\; \Pi_1 \overset{M}{\|}_x \Pi_0$.

Theorem 13. $(A \overset{M}{\|}_x B) \overset{M}{\|}_x C \;=\; A \overset{M}{\|}_x (B \overset{M}{\|}_x C)$ for any A, B and C, iff M is associative in this sense: $(\Pi_0 \overset{M}{\|}_x \Pi_1) \overset{M}{\|}_x \Pi_2 \;=\; \Pi_0 \overset{M}{\|}_x (\Pi_1 \overset{M}{\|}_x \Pi_2)$.

Proof. Let $A_0 \;\widehat{=}\; A[\mathtt{0}.x/x]$, $B_1 \;\widehat{=}\; B[\mathtt{1}.x/x]$ and $C_2 \;\widehat{=}\; C[\mathtt{2}.x/x]$. If M is associative then

$(A \overset{M}{\|}_x B) \overset{M}{\|}_x C$

$=$ definition and Law 6

$(A_0 \wedge B_1) :_{(\mathtt{0}.x,\mathtt{1}.x)} (\Pi_0 \overset{M}{\|}_x \Pi_1) \overset{M}{\|}_x (C_2 :_{\mathtt{2}.x} \Pi_2)$

$=$ Law 6 and 10

$(A_0 \wedge B_1) :_{(\mathtt{0}.x,\mathtt{1}.x)} (C_2 :_{\mathtt{2}.x} (\Pi_0 \overset{M}{\|}_x \Pi_1) \overset{M}{\|}_x \Pi_2)$

$=$ Law 9

$(A_0 \wedge B_1 \wedge C_2) :_{(\mathtt{0}.x,\mathtt{1}.x,\mathtt{2}.x)} (\Pi_0 \overset{M}{\|}_x \Pi_1) \overset{M}{\|}_x \Pi_2$

$=$ associativity of M

$(A_0 \wedge B_1 \wedge C_2) :_{(\mathtt{0}.x,\mathtt{1}.x,\mathtt{2}.x)} \Pi_0 \overset{M}{\|}_x (\Pi_1 \overset{M}{\|}_x \Pi_2)$

$=$ Law 9

$$(B_1 \wedge C_2) :_{(0.x, 2.x)} (A_0 :_{0.x} \mathrm{II}_0 \overset{M}{\|}_x (\mathrm{II}_1 \overset{M}{\|}_x \mathrm{II}_2))$$
$=$ Law 6 and 10
$$(B_1 \wedge C_2) :_{(0.x, 2.x)} (A_0 :_{0.x} \mathrm{II}_0) \overset{M}{\|}_x (\mathrm{II}_1 \overset{M}{\|}_x \mathrm{II}_2)$$
$=$ definition and Law 6
$$A \overset{M}{\|}_x (B \overset{M}{\|}_x C) .$$

The other half of the theorem is proved by assigning A, B and C to be II_0, II_1 and II_2 respectively. $\square$

Parallel-by-merge composition is a special case of parallel-via-medium composition when the interface is the list of all dashed variables.

Healthiness conditions

A healthiness condition $A = f(A)$ is the fixpoint of a healthiness function f. If f is universally disjunctive, then it can be defined using a generic composition $A = A :_x B$. Writing healthiness conditions in this form is motivated by one of Hoare and He's healthiness conditions for designs: $A = A \mathbin{;} \mathrm{II}$ where II is the skip command that keeps the state unchanged. Besides the simplicity of this condition, the proof of its idempotence is pleasingly straightforward: $\mathrm{II} \mathbin{;} \mathrm{II} = \mathrm{II}$. Most healthiness conditions cannot be defined with a sequential composition. Fortunately, using generic composition, we can generalise the technique and tackle a variety of more complicated healthiness conditions (see Section 5).

The following proposition is a consequence of associativity Law 5. It identifies a simple condition of idempotence.

Proposition 14. $A = (A :_x B)$ is idempotent iff $B :_x B = B$.

Second-order generic composition

A parallel-via-medium composition becomes a *second-order generic composition* if the predicates on both sides are identical.

Definition 6. $A ::_x B \;\widehat{=}\; A \overset{B}{\|}_x A$.

A second-order generic composition $A ::_x B$ simulates the predicate A twice by allowing only variables in the interface to take different values. Behaviours of A are pairwise compared by the second predicate B. In general the composition is not even finitely distributive, i.e.

$$(A \vee C) ::_x B \;\neq\; (A ::_x B) \vee (C ::_x B)$$

and thus cannot be reduced to the basic form of generic composition.

A healthiness condition based on a second-order generic composition is normally written $A = (A ::_x B)$. The predicate A actually appears twice in the definition; thus it becomes possible to compare each pair of values of x. This composition is very useful when we need to impose a healthiness condition on the *range* of some variable. A range is the set of all possible values of a variable when other variables are fixed. For example, the range of y in the predicate $(y \geqslant x + 1)$ is $\{n \in \mathbb{N} \mid n \geqslant 5\}$ if x is fixed to 4.

Higher-order generic composition

We take a further step to allow up to infinitely-many copies of a predicate in a *higher-order generic composition.* Let I denote an *indexing set.* Each index $i \in I$ corresponds to a copy of A whose interface x is re-labeled as $i.x$. Let $I \cdot x$ denote a variable list including every $i.x$ $(i \in I)$ in some agreed order.

Definition 7. $A :_{x^I} B \;\widehat{=}\; \bigwedge_{i \in I} A[i.x/x] :_{I \cdot x} B$.

In general, the composition is not reducible to the basic form of generic composition.

If $I = \{0, 1\}$, this definition collapses to a second-order generic composition; otherwise, if I is an infinite set, values taken from the interfaces of the copies of A form a set whose cardinality can be as big as the cardinality of I; if I is the set of all possible values of x, then the values taken from the copies of A can be any subset of possible values. Using this composition, we can then reason about cardinality and the limit points of a variable's range.

A healthiness condition based on a higher-order generic composition is normally written $A = (A :_{x'} B)$.

5. Applications

In this section we shall demonstrate the expressiveness of generic composition by applying it to yield operators and healthiness conditions for a variety of computational models.

5.1. Parallel compositions

Conjunction

A (binary) parallel composition demonstrates two predicates at the same time and hence becomes a conjunction if there is no interference between the predicates.

Proposition 15. $A \wedge B \;=\; A \overset{true}{\|}_x B$.

Interference in the initial state

In a parallel-by-merge composition, the merge is composed sequentially after two predicates. That means the initial states of the two components are simply the direct copies of the initial state of the parallel composition.

This may not be general enough in some cases. For example, let the variable x denote the number of resources available. A program may consume and/or produce resources. Thus it can be modelled by a predicate on two variables x and x' which denote the numbers of resources available in the initial and final states respectively. Two programs compete with each other for resources in the initial state. In the initial state each resource is received by only one of the programs, and in the final state resources from both programs must be collected.

Such parallelism of resource sharing is not compatible with parallel-by-merge composition in that the numbers of resources received by parallel programs are related to each other and are normally not equal to the total number of resources in the initial state.

Fortunately the parallel composition can be defined as a parallel-via-medium composition with an interface containing both x and x'.

Definition 8. $A \| B \;\widehat{=}\; A \overset{M}{\|}_{(x,x')} B$ where $M \;\widehat{=}\; (x = 0.\overline{x} + 1.\overline{x}) \wedge (x' = 0.\overline{x}' + 1.\overline{x}')$.

5.2. Healthiness conditions

We now study some healthiness conditions appearing in various semantic models. Each healthiness condition is written in the form $A = (A :_x R)$, $A = (A ::_x R)$ or $A = (A :_{x'} R)$. The corresponding healthiness function transforms any unhealthy predicate to a healthy one.

Recall that the range of a variable in a predicate is the set of all possible values of the variable when the values of other variables are fixed.

Downwards-closure

In semantic studies we often need to stipulate a variable x to be downwards-closed or upwards-closed. Let the type of variable x be real numbers. The following healthiness condition requires the range of x to be $(-\infty, a)$ or $(-\infty, a]$ where a can be any real number. Note that downwards closure does not require the inclusion of the 'limit point' a. We will be able to reason about limit points using higher-order generic composition.

HC 1. $A \;=\; A :_x x \leqslant \overline{x}$.

The ranges of downwards-closed predicates can be easily compared under the refinement order. For example $(y+1>x) \Rightarrow (y+1\geqslant x) \Rightarrow (y+2>x) \Rightarrow (y+2\geqslant x)$. According to Proposition 14, the above healthiness condition is idempotent if $(x\leqslant\overline{x}) :_x (x\leqslant\overline{x}) = (x\leqslant\overline{x})$, which is obvious. HC 1 defines a more general downwards-closure if the type of x is a partial order. Upwards-closure can be expressed by replacing $\leqslant$ with $\geqslant$.

Comparison of nondeterminism

A similar healthiness condition states that the value of the variable y is always a possible nondeterministic outcome of x. Thus x is more nondeterministic than y.

HC 2. $\quad A \;=\; A :_x x\in\{\overline{x},\, y\}$.

Prefix independence

The healthiness condition $A \;=\; \bigvee_s A[s, (s \,\hat{}\, tr' - tr)/tr, tr']$ for traces mentioned in Section 1 now simplifies:

HC 3. $\quad A \;=\; A :_{(tr,tr')} (\overline{tr}' - \overline{tr} = tr' - tr)$.

It is an easy exercise to check that the healthiness condition is indeed idempotent.

Healthy Designs

Another healthiness condition $A[\mathit{false}\,/\,ok'] \Rightarrow A[\,\mathit{true}\,/\,ok']$ for designs mentioned in Section 1 is redefined:

HC 4. $\quad A \;=\; A :_{ok'} (\,\overline{ok}' \Rightarrow ok'\,)$.

If the range of ok' in a predicate A is $\{\,\mathit{false}\,\}$, the generic composition will transform the predicate by adding the value *true* to the range. Thus a predicate is healthy in this sense iff it is either deterministic on 'true' or nondeterministic between 'true' and 'false'. This healthiness condition represents total correctness of sequential programs.

Prefix closure of strategy

In game semantics [Abr99] a *strategy* S is a non-empty set of even-length sequences satisfying a few healthiness conditions. One of them is the prefix-closure condition:

$$sab\in S \;\Rightarrow\; s\in S\ .$$

This can be represented as a generic composition:

HC 5. $\quad A \;=\; A :_s (2\,|\,len(s) \,\wedge\, s\leqslant\overline{s})$.

The above healthiness function is monotonic and idempotent and it transforms any unhealthy predicate to a healthy one by adding all even-length prefixes of each sequence.

Decoupled variables

In distributed computing asynchronous communication exhibits latency. If there is no causal relation established by any communication between two observable variables, we call them *decoupled* variables. The following healthiness condition states that, if x and y are *decoupled* variables, their nondeterministic choices are not directly correlated. That means the range of joint nondeterministic choices of the two variables is exactly the Cartesian product of their separate ranges.

HC 6. $\quad A \;=\; A ::_{(x,y)} (x\in\{\,0.\overline{x},\, 1.\overline{x}\,\} \,\wedge\, y\in\{\,0.\overline{y},\, 1.\overline{y}\,\})$.

If a predicate A depends on only three variables x, y and z, then this healthiness condition holds iff A can be decomposed into two predicates $A_1(x,z)$ and $A_2(z,y)$ in conjunction $A = A_1(x,z) \wedge A_2(z,y)$ in which x is not directly related to y.

Determinism

In relational semantics, a variable is deterministic if its range is always a singleton set. The semantic space of deterministic relations is not a complete lattice because the least upper bound (i.e. relational union) is not closed in the space. However, adding the bottom *true* and the top *false* to such a space will create a complete lattice, which can be expressed as:

HC 7. $A \;=\; A \vee A ::_x (0.\overline{x} \neq 1.\overline{x})$.

The corresponding healthiness function transforms any deterministic predicate to itself and any nondeterministic predicate to the predicate *true* . Although this definition has an equivalent form $A \;=\; (A ::_x (x = 0.\overline{x} = 1.\overline{x} \;\vee\; 0.\overline{x} \neq 1.\overline{x}))$, HC 7 appears simpler. Here the range of x may be an empty set, a singleton set or the set of all possible values. Although this is not exactly a healthiness condition for pure determinism, their difference lies only at some 'extreme' points. This is a low price to pay, because what we obtain is a uniform way of writing the healthiness condition and the semantic space now becomes a complete lattice. This has been our standard technique for dealing with non-complete-lattice semantic spaces.

Deterministic strategies

In game semantics [Abr99], a player and an opponent act alternatively in a game. The player is supposed to play deterministically according to the pre-history: if both *sab* and *sac* are two even-length sequences of a strategy, they must be identical, i.e. $b = c$. This healthiness condition is defined as follows:

HC 8. $A \;=\; A \vee A ::_s \exists\, t\, a\, b\, c \cdot (0.\overline{s} = tab \wedge 1.\overline{s} = tac \wedge b \neq c)$.

The corresponding healthiness function transforms any deterministic strategy to itself and any nondeterministic one to the bottom predicate 'true'. The *Innocence* condition in game semantics [Abr99] is very similar to HC 8. Other healthiness conditions in [Abr99] including the bracketing condition can be trivially defined as conjunctive healthiness conditions.

Convexity of probabilistic distributions

In probabilistic semantics [HSM97], a probabilistic distribution is a function $f : W \to [0, 1]$ from a set W of states to the real interval $[0, 1]$. The following healthiness condition states that the distributions of variable f form a convex area in the space containing all possible distribution functions.

HC 9. $A \;=\; A ::_f (\exists \lambda \in [0, 1] \cdot (f = \lambda \times 0.\overline{f} \;+\; (1 - \lambda) \times 1.\overline{f}))$.

The corresponding healthiness function transforms each predicate by adding all intermediate linear combinations of any pair of distributions into the range.

Closed sets and safety properties

If infinitely-many copies of a predicate are allowed to appear in a healthiness condition, we will be able to reason about the cardinality and the limit points of a range.

Let the type of variable x be X where $(X, \mathcal{T})$ is a topological space. The following healthiness condition requires that the range of the variable be a closed set.

HC 10. $A \;=\; A :_x X \;\; x \in \mathrm{cl}\, \Sigma(x, X) \quad$ where $\;\Sigma(x, X) \;\widehat{=}\; \{ i.\overline{x} \mid i \in X \}$.

There are as many as $(\mathtt{card}\; X)$ copies of A appearing in this definition. Thus $\Sigma(x, X)$ can be any non-empty subset of x's range. The corresponding healthiness function of this condition transforms any predicate with non-closed range to a predicate with the smallest closed range (i.e. topological closure) containing $\Sigma(x, X)$.

In linear temporal logic [Pnu77, Lam94], a reactive process is an infinite sequence of states, each of which has a type T . Let $X \;\widehat{=}\; T^{\infty}$ denote the space of all processes. There is a natural topology on T^{∞} in which each basic open set is the set of all infinite sequences sharing a common finite prefix and each open set is the union of some basic open sets. It has been shown that safety properties are exactly the closed sets of the topology, (pure) liveness properties are exactly the dense sets, and any property can be expressed as the intersection of a safety property and a liveness property [AS85].

Thus the set of all predicates satisfying HC 10 on a sequence variable x is a semantic space containing all safety properties. The corresponding healthiness function transforms any property to the strongest safety property implied by it.

Dense sets and liveness properties

A dense set intersects every non-empty open set. The following healthiness condition requires the range of variable x to be a dense set:

HC 11. $A \;=\; A \wedge A :_{x^X} \; \forall S \in \mathcal{T} \cdot (S \cap \Sigma(x, X) = \emptyset \;\Rightarrow\; S = \emptyset)$.

There are as many as $(\texttt{card}\ X)$ copies of A appearing in the definition. The corresponding healthiness function transforms any predicate with a dense range to itself and any unhealthy predicate to the top *false* . Again we need the help of one extra point *false* to let the space be a complete lattice, since the intersection of dense sets may not be dense.

Cauchy-closure of probabilistic distributions

HC 9 requires the range of a variable to be convex. The boundary points (i.e. limit points) may or may not be included in the range. The Cauchy-closure healthiness condition in [HSM97] requires those boundary points to be included. To reason about limit points, we need a higher-order generic composition. Each variable f is a function of probabilistic distribution from a finite set W to the interval $[0,1]$. The set $F \;\widehat{=}\; W \to [0,1]$ of all such distributions is isomorphic to an n-dimensional Euclidean space $[0,1]^n$ where n is the cardinality of W . Cauchy-closure is then defined by:

HC 12. $A \;=\; A :_{f^F} \; \forall \varepsilon > 0 \, \exists i \cdot | \, i.\overline{f} - f \, | < \varepsilon$.

In this definition, $| \cdot |$ is the Euclidean distance associated with the Euclidean space. The corresponding healthiness function transforms any predicate by 'pasting' those boundary points to the range of variable f .

Finitary condition

Another important healthiness condition is called the *finitary condition* [Dij75]. A variable is finitary if its range is either a finite set or the set of all possible values.

HC 13. $A \;=\; A \vee A :_{x^{\mathbb{N}}} \; \texttt{card}\, \Sigma(x, \mathbb{N}) = \omega$.

There are up to ω-many copies of A appearing in the definition (where ω is the cardinality of $\mathbb{N}$). Its corresponding healthiness function transforms any non-finitary predicate to a finitary one by pasting all other possible values to the range of x .

6. Conclusions

Our objective is to support hierarchical construction of semantic models. The introduction of generic composition serves this purpose by unifying various semantic definitions and provides a higher-level reasoning tool. The completeness theorem shows that generic composition and its weak inverse are as expressive as first-order predicate calculus. Any universally disjunctive (or conjunctive) predicate transformer can be represented as a generic composition (or its weak inverse). This has shown the generality in theory. The examples from a variety of models have shown its generality in applications.

More than one copy of a predicate is allowed in higher-order generic compositions. These advanced forms of compositions are neither disjunctive nor conjunctive and hence not reducible to the basic form of generic composition. However such compositions are very useful in expressing advanced properties on cardinality and limit points and deserve further research.

Acknowledgement

This paper arose from a DPhil thesis [Che01]. The author is grateful to his supervisor J.W. Sanders for various discussions, comments, suggestions and review of the draft of this paper. The author would like

to thank referees for their helpful comments and suggestions. An early version of this paper [Che00] was presented at 2nd International Conference on Integrated Formal Methods (IFM 2000).

References

[Abr99] A. Abramsky. *Semantics of interaction: an introduction to Game Semantics*, pages 1–31. Cambridge Press, 1999.

[AS85] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[BS89] R. J. R. Back and K. Sere. Stepwise refinement of action systems. In *Mathematics of Program Construction*, pages 115–138, 1989.

[Che80] B.F. Chellas. *Modal Logic, An Introduction.* Cambridge University Press, 1980.

[Che00] Y. Chen. How to write a healthiness condition. In *2nd International Conference on Integrated Formal Methods*, LNCS 1945, pages 299–317. Springer-Verlag, 2000.

[Che01] Y. Chen. *Formal Methods for Global Synchrony.* PhD thesis, Oxford University Computing Laboratory, 2001.

[CL02] Y. Chen and Z. Liu. Unifying temporal logics. Technical Report 09, Department of Maths & Computer Science, University of Leicester, 2002.

[Dij75] E. W. Dijkstra. Guarded commands, nondeterminacy and the formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[Fin74] K. Fine. An incomplete logic containing s4. *Theoria*, 40:23–29, 1974.

[Heh84] E.C.R. Hehner. Predicative programming I, II. *Communications of ACM*, 27(2):134–151, 1984.

[HH86] C.A.R. Hoare and J. He. Weakest prespecifications, I,II. *Fundamenta Informatica*, 9:51–84, 217–252, 1986.

[HH98] C.A.R. Hoare and J. He. *Unifying Theories of Programming.* Prentice Hall, 1998.

[Hoa85a] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Hoa85b] C.A.R. Hoare. *Programs are predicates*, pages 141–155. Prentice Hall, 1985.

[HSM97] J. He, K. Seidel, and A. K. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2):171–192, 1997.

[JT51] B. J'onsson and A. Tarski. Boolean algebras with operators, Part I. *American Journal of Mathematics*, 73:891–939, 1951.

[Kar98] B.V. Karger. Temporal algebra. *Mathematical Structures in Computer Science*, 8(3):277–320, 1998.

[Kri63] S. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[Lam94] L. Lamport. A temporal logic of actions. *ACM Transctions on Programming Languages and Systems*, 16(3):872–923, 1994.

[Mad96] R. D. Maddux. Fundamental study relation-algebraic semantics. *Theoretical Computer Science*, 160:1–85, 1996.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[Tar41] A. Tarski. On the caculus of relations. *Journal of Symbolic Logic*, 6(3):73–89, 1941.

[Val90] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.