INFINITE BEHAVIOUR AND FAIRNESS IN CONCURRENT CONSTRAINT PROGRAMMING

Marta Kwiatkowska*

Department of Mathematics and Computer Science†
University of Leicester
Leicester LE1 7RH
and
Department of Computing‡
Imperial College
180 Queens Gate
London SW7 2BZ

Abstract

In concurrent constraint programming, *divergence* (*i.e.* an infinite computation) and *failure* are often identified. This is undesirable when modelling systems in which infinite behaviour arises naturally. This paper sets out a framework for an axiomatic and denotational view of concurrent constraint programming, and considers the relationship of both views as an instance of Stone duality. We propose a construction of a constraint system which allows both finite and infinite constraints. Subsequently, we provide semantic, topological definitions of safety, liveness and fairness properties in a given constraint system. The process language considered is parametrized by the constraint system. It allows the actions *ask* and *tell*, the prefix operator $\rightarrow$, the (angelic) non-deterministic choice operator $\oplus$, the procedure call $p(X)$, and the concurrency operator $\|$.

Keywords: concurrent constraint programming, liveness, fairness, semantic properties.

*This paper was partly written when the author was Visiting Professor at CWI, Amsterdam, sponsored by the Netherlands Organisation for Scientific Research (NWO).

†Permanent address.

‡Institution visited for the duration of the Nuffield Science Research Fellowship.

Contents

1 INTRODUCTION

Concurrent constraint programming (18, 19, 4, 8) is an approach rooted in constraint logic programming and concurrent logic programming. The basic paradigm is that a number of agents, which are executed concurrently, communicate via a common store. The store is not viewed as a mapping of variables to their contents, but as a "constraint", that is, partial information about what the variables may contain. Agents communicate through the (indivisible) *ask* and *tell* actions. The action *ask(c)*, where c is a constraint, succeeds if the store contains enough information to entail c; otherwise, it will be blocked until such time. The action *tell(c)* results in a new store, which is a conjunction of the old store and the constraint c. Depending on the choice of remaining operators (*e.g.* guards, concurrency, non-determinism), we can obtain concurrent asynchronous languages of varying power.

This paper is concerned with the foundations of a semantic framework for such languages, which would allow to formalise the notion:

$$P \models \phi \tag{1}$$

i.e. "P satisfies ϕ", where P is a syntactic description of a program (*i.e.* a (well-formed) term in a concurrent constraint programming language), and ϕ is a syntactic description of a formula (*i.e.* a specification written in some logic). Note that the two syntaxes are not necessarily the same, and there is no reason why they should be; while P will be defined in a transition-system based language and given meaning as a point in some denotational domain, ϕ will be defined in some axiomatic, deductive system.

We shall employ the concept of *Stone duality*, the significance of which to computer science was first recognised by Abramsky (2). Stone duality refers to a categorical correspondence (in fact, it is a duality of categories) between logical, axiomatic systems and their topological models. The first instance of this duality was pointed out by Stone, and it concerned Boolean algebras. Stone sought a representation of every Boolean algebra as a field of sets, in which *the operations of meet, join and complement are represented by set-theoretic intersection, union and complement*. Apart from Boolean algebras, a number of logical systems have been considered to date; for more information see (9). The duality ensures that the program logic and the denotational semantics are in perfect synchrony (not only soundness and completeness in the usual sense, but also that each determines the other up to isomorphism). We are thus provided with two different perspectives: the "localic" (axiomatic) and "spatial" (model-theoretic). In the spatial side of the duality, points (programs) are viewed as primary objects, and properties are constructed from open sets of points; this is the usual denotational view, where $P \in \phi$ (P is a point, ϕ an open set) is used to model $P \models \phi$. The "localic" side views properties as primary, and constructs points (programs) as theories (prime filters or, dually, ideals) of properties; this is a logical view, where $\phi \in P$ (ϕ is a property, P a filter) is used to model $P \models \phi$.

The ultimate aim of the research initiated in this paper is to provide a framework for expressing and reasoning about a large class of program properties including *infinitary* properties, that is, properties that cannot be detected in finite time. This class includes *e.g.* *liveness* and *fairness* properties (15, 13, 5, 11), such as "guaranteed response", "weak fairness" and "strong fairness". One characteristic feature of these properties is that one has to consider infinite computations, as well as the finite ones. Unfortunately, in concurrent constraint programming infinite computations are often treated as *divergence*, and are identified with *failure*. This is unsatisfactory if the so called *reactive systems*, *e.g.* operating systems, dataflow networks, are to be modelled correctly. In such systems, the possibility of infinite behaviour arises naturally, and is often desirable. Thus, the identification of infinite computations with failure is no longer acceptable.

The outline of the paper is as follows. We begin with a deductive, finitely axiomatizable system E of constraints, the purpose of which is to express constraints syntactically and reason about them. We then construct a constraint system $\mathcal{C}(E)$ as a model of E, and consider the issue of Stone duality of E and $\mathcal{C}(E)$. Through the localic side of the Stone duality, E can be shown isomorphic to a sub-algebra of a topology on $\mathcal{C}(E)$. This allows us to interpret each syntactic constraint ϕ as *a subset* $[\![\phi]\!]$ of $\mathcal{C}(E)$.

At the other end of spectrum, we consider a process language parametrized by the constraint system $\mathcal{C}$. The language allows: actions $tell(c)$ for $c \in \mathcal{K}(C)$, guards $ask(c) \ \rightarrow \ A$, concurrency operator $\|$, angelic non-determinism operator $\oplus$, hiding operator $\exists X.A$ and procedure call $p(X)$. Following the approach of (19, 8), which defines denotations of processes as certain closure operators on $\mathcal{C}$, we can identify a denotation $[\![P]\!]$ of a term P in this language as *a subset* of $\mathcal{C}$. Finally, we take advantage of the representations of formulae and processes as subsets of $\mathcal{C}$ and the notion $P \models \phi$ by $[\![P]\!] \subseteq [\![\phi]\!]$.

Following the work of Smyth (21) and Abramsky (2), we identify *finite observations* in $\mathcal{C}$ with (a subfamily of) the *open sets*, and construct more general *semantic* properties, *e.g.* safety, liveness and fairness, from these. We also exemplify these notions using some well-known properties. Although we are not yet capable of expressing the above classes of properties *syntactically* as formulae of E as finite axiomatizability does not suffice for this purpose, we briefly suggest possible extensions to the theory.

2 SAFETY, LIVENESS AND FAIRNESS

The classification of program properties into safety and liveness is due to Lamport (14). He justified the distinction on the grounds of employing two different methods when verifying that a given program P satisfies a property ϕ. In order to verify properties such as mutual exclusion (no two processes should be in their critical section at the same time) it is sufficient to use induction. On the other hand, properties such as termination require the well-founded set method. Informally, a *safety* property is one which states that

> *Nothing bad happens.*

whereas a liveness property states that

> *Something good happens.*

Thus, in mutual exclusion the "bad" thing is (at least) two processes being in their critical sections at the same time, while in termination the "good" thing is that the program reaches its final state. While safety properties must *avoid* the bad things, liveness properties must *guarantee* the good things. A more detailed classification based on the syntactic representation in temporal logic appeared in (15).

The class of liveness properties includes two quite different types of properties: those which can be detected at finite time (termination being such), and those which cannot. An example of the latter class is "guaranteed response" (every request will eventually be granted), since no finite examination of a computation can determine that every occurrence of a request action is followed by a granting action.

Fairness properties constitute a large class of infinitary properties, typically viewed as a a subclass of liveness properties. There are many definitions of fairness properties, see *e.g.* (5, 10, 12). What they all have in common is that they restrict the set of potential computations of a system by disallowing those infinite computations which indefinitely delay some system component. Depending on the choice of a component (*e.g.* a process, non-deterministic guard or state), one can obtain fairness properties of different *granularity*, which can be further varied in terms of *strength* (*e.g.* weak, strong fairness). In state-transition system models, fairness properties usually take the form of an implication, see *e.g.* a typical formulation of "strong process fairness":

> $\forall P$, *P process. P enabled infinitely often $\Rightarrow$ P taken infinitely often.*

In denotational semantics, where the notion of "being enabled in a state" is not readily definable, other formulations of fairness have to be used. A typical notion applicable in such contexts is *fairmerge* (16), which for strings can be defined as an interleaving merge of string languages in such a way that the whole of each string is taken. For example,

$$fm(0^\omega, 1^\omega) = (0^+1^+)^\omega + (1^+0^+)^\omega \tag{2}$$

which excludes strings such as 0^ω, 1^n0^ω, and is clearly an infinitary property in the sense explained above.

Fairness is known to affect liveness properties (even the finitary ones), in the sense that the correctness of the program with respect to some property cannot be proved *unless* the infinite computations of the program are restricted to just the fair ones. Thus, the issue of fairness cannot be easily ignored.

On the other hand, fairness is regarded as a confused and difficult subject: it requires the addition of infinite computations, causes discontinuity of semantic functions (16), and is sensitive to the particular features of the semantic model such as confusion (10).

3 MATHEMATICAL PRELIMINARIES

We shall assume some knowledge of lattice theory and topology. For the benefit of those readers who are not familiar with these subjects, we give a brief summary of basic definitions and facts. For more information consult (6, 7, 3, 9).

3.1 Lattices

A *join-semilattice* is a poset $(A, \leq)$ in which every pair of elements $a, b \in A$ has a join $a \vee b$, and which has the least element 0. Dually, we can define a *meet-semilattice* as a poset $(A, \leq)$ in which every pair of elements $a, b \in A$ has a meet $a \wedge b$, and which has the greatest element 1.

A *semilattice homomorphism* is a map f which preserves the distinguished element 0 (*i.e.* $f(0) = 0$) and the operation $\vee$ (*i.e.* $f(a \vee b) = f(a) \vee f(b)$). Every semilattice homomorphism is necessarily monotone (*i.e.* it satisfies $a \leq b \Rightarrow f(a) \leq f(b)$), but a monotone map between semilattices need not be a homomorphism.

A *lattice* is a poset $(A, \leq)$ in which every finite subset has a join and a meet. Equivalently, a lattice is a structure $(A, \vee, \wedge, 0, 1)$ such that: A is a set, 0 and 1 are the distinguished elements, $\vee$ (resp. $\wedge$) is associative, commutative and idempotent and has 0 (resp. 1) as unit element, and the absorptive laws ($a \vee (a \wedge b) = a$, $a \wedge (a \vee b) = a$) are satisfied. A lattice A is *distributive* iff the distributive law $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ is satisfied. The dual of this law can be deduced by applying the absorptive laws. A is *complete* iff every subset has a join and a meet.

In any lattice, an element x satisfying $x \wedge a = 0$ and $x \vee a = 1$ is called a *complement* of a. It can be shown that in a distributive lattice complements are unique if they exist. A *Boolean algebra* is a distributive lattice A equipped with an additional operation $\neg : A \longrightarrow A$ such that $\neg a$ is a complement of a.

A lattice is said to be a *Heyting algebra* if, for each pair of elements (a, b), there exists an element $(a \to b)$ such that $c \leq (a \to b)$ iff $c \wedge a \leq b$[1]. It can be shown that a Heyting algebra is distributive. In fact, it satisfies the infinite distributive law:

$$a \wedge \bigvee S \quad = \quad \bigvee \{a \wedge s \mid s \in S\}$$

Furthermore, a complete lattice is a Heyting algebra iff it satisfies the infinite distributive law (9). Also, a Heyting algebra A is a Boolean algebra iff $\neg\neg a = a$ for all $a \in A$.

An element p of a complete Heyting algebra is *prime* if $S \subseteq A$, S finite, and $\wedge S \leq p$, then $s \leq p$ some $s \in S$. An element p is *coprime* iff it is prime with respect to the opposite ordering.

Example 3.1 *Let X be a set, then the set $F(X)$ of cofinite subsest of X is a meet-semilattice, and the powerset $\wp(X)$ is a complete lattice, with $\leq$ interpreted as inclusion, and the operations $\vee$ and $\wedge$ interpreted as union and intersection.*

3.2 Ideals and Filters

A subset S of a join-semilattice is *directed* iff it is non-empty, and every pair of elements of S has an upper bound in S. A subset I of a join-semilattice A is said to be an *ideal* if it is a lower set (*i.e.* $a \in I$ and $b \leq a$ imply $b \in I$) such that $0 \in I$, and $a \in I, b \in I$ imply $a \vee b \in I$. The set of all ideals of the join-semilattice A is denoted $\mathcal{I}(A)$.

For any $a \in A$, the subset $\downarrow a = \{b \in A \mid b \leq a\}$ is the smallest ideal of A containing a, and is often called the *principal ideal* generated by a.

The notion dual to that of an ideal is a filter. A subset F of a meet-semilattice A is said to be a *filter* if it is an upper set such that $1 \in F$, and $a \in F, b \in F$ imply $a \wedge b \in F$.

Let I be an ideal of a lattice A. I is said to be a *prime ideal* iff the complement of I in A is a filter, or, equivalently, $1 \notin I$ and $a \wedge b \in I$ implies either $a \in I$ or $b \in I$. This definition is also equivalent to I being the *kernel*, that is $f^{-1}(0)$, of a lattice homomorphism $f : A \to \mathbf{2}$, where $\mathbf{2}$ denotes the two-element lattice. The dual (and a complement) of a prime ideal is a *prime filter*. Likewise, a prime filter can be characterized as the dual kernel $f^{-1}(1)$ of a lattice homomorphism $f : A \to \mathbf{2}$.

Let A be a complete Heyting algebra. A subset F of A is said to be a *completely prime filter* iff $1 \in F$; $a \in F, b \in F$ imply $a \wedge b \in F$; F is an upper set; and $\bigvee S \in F$ iff $\exists a \in S. a \in F$. Every completely prime filter F can be identified with $f^{-1}(1)$ for $f : A \longrightarrow \mathbf{2}$ a frame homomorphism.

[1] In a Boolean algebra, the operation $\neg$ can be recovered from the binary operation $\to$ since $\neg a = (a \to 0)$. In a general Heyting algebra, this is the *definition* of $\neg$.

3.3 Algebraic Lattices and Closure Operators

Let A be a complete lattice. An element $a \in A$ is called *compact* (or *finite*) iff, for every directed set $S \subseteq A$ with $\bigvee S \geq a$, there exists $s \in S$ with $s \geq a$. The set of all compact elements of A is denoted $\mathcal{K}(A)$. A is *algebraic* iff every element in A is expressible as the join of compact elements of A.

The following results are now considered standard. The isomorphism is given by $b \mapsto \downarrow b$ (*i.e.* an element maps onto the principal ideal determined by it, and $I \mapsto \bigvee I$ (*i.e.* an ideal maps onto its join).

Lemma 3.2 (Johnstone (9)) *Let $(A, \leq)$ be a join-semilattice.*

1. *The ideal completion $(\mathcal{I}(A), \subseteq)$ is a complete algebraic lattice.*

2. *$(\mathcal{K}(\mathcal{I}(A)), \subseteq)$ is isomorphic to $(A, \leq)$.*

Lemma 3.3 (Vickers (22)) *The ideal completion $(\mathcal{I}(A), \subseteq)$ of a distributive lattice A is a complete Heyting algebra.*

Let $(A, \leq)$ be a partial order. A *closure operator* on A is a map $f : A \longrightarrow A$ which is: extensive (*i.e.* $\forall c. c \leq f(c)$), idempotent (*i.e.* $\forall c. f(f(c)) = f(c)$), and monotone (*i.e.* $\forall c, d. c \leq d \Rightarrow f(c) \leq f(d)$). A *kernel operator* on A is a map $f : A \longrightarrow A$ which is idempotent, monotone and satisfies $\forall c. f(c) \leq c$.

Example 3.4 *Let X be a topological space, then the function c mapping $Y \subseteq X$ to its closure is a closure operator, and the function k mapping $Y \subseteq X$ to its interior is a kernel operator. The image of c are the closed sets $\Gamma(X)$ and of k are the open sets $\Omega(X)$.*

3.4 Topological Spaces and Locales

A *topology* on the set X is a family of subsets $\Omega(X)$ (called *open sets*) which contains X and $\emptyset$, and is closed under finite intersections and arbitrary union. A *topological space* is a pair $(X, \Omega(X))$, where X is a set and $\Omega(X)$ is a topology on X.

A topological space is *Hausdorff* (or $\mathcal{T}_2$) iff for each pair of points $x \neq y$ there exist two open sets G, H such that

$$x \in G, y \in H, G \cap H = \emptyset. \tag{3}$$

A topological space is called $\mathcal{T}_1$ iff for each pair of points $x \neq y$ there exists an open set G such that $x \in G$ and $y \notin G$. A topological space is called $\mathcal{T}_0$ iff for each pair of distinct points there exists an open set which contains one of the points and does not contain the other one. A topology is *compact* iff every open cover contains a finite subcover.

A set $Y \subseteq X$ is *closed* iff its complement $\overline{Y}$ is open. The family of all closed sets of the space X will be denoted $\Gamma(X)$. A set $Y \subseteq X$ is *clopen* iff it is both closed and open. A G_δ*-set* is a countable (*i.e.* finite or infinite) intersection of open sets. An F_σ*-set* is a countable (*i.e.* finite or infinite) union of closed sets. A set $Y \subseteq X$ is *dense* iff its complement contains no non-empty open set.

A family $\mathcal{B}$ of open sets is called a *base* iff each open set can be represented as the union of elements of a subfamily of $\mathcal{B}$. A family $\mathcal{S}$ of open sets is called a *subbase* iff the family of all finite intersections of elements of $\mathcal{S}$ is a base of the space.

Let $(A, \leq)$ be a complete partial order. The *Scott topology* on A consists of all sets U such that U is upward-closed (*i.e.* $x \in U \wedge x \leq y \Rightarrow y \in U$) and, for every directed set $M \subseteq A$, if $\bigsqcup M \in U$ then some element of M is in U. The Scott-open subsets of a cpo A will be denoted $\sigma\Omega(A)$, and the Scott-closed by $\sigma\Gamma(A)$. It is easy to see that a set is Scott-closed iff it is a lower set closed under directed joins. The closure of a singleton set $\{x\}$ in the Scott topology is $\downarrow x$.

In an algebraic lattice, the basis of the Scott topology is the family of sets $\uparrow x$ for $x \in \mathcal{K}(A)$, together with the empty set. A subset U of A is *Scott-compact* iff any cover of U with compact elements, *i.e.* a set $V \subseteq \mathcal{K}(A)$ such that $U \subseteq \uparrow V$, contains a finite subset $F \subseteq U$ which is also a cover. The Scott-compact subsets are closed under finite union and intersection. A subset U of A is *Scott compact-open*[2] (notation $\mathcal{K}\Omega(A)$) iff there exists a finite subset $F \subseteq \mathcal{K}(U)$ such that $U = \uparrow F$.

A topological space X is *coherent* if the compact-open subsets $\mathcal{K}\Omega(X)$ form a basis closed under finite intersections, *i.e.* for which $\mathcal{K}\Omega(X)$ is a distributive sublattice of $\Omega(X)$.

Given a topological space X, the lattice $\Omega(X)$ of open subsets is complete and satisfies the infinite distributive law:

$$a \wedge (\bigvee S) \quad = \quad \bigvee\{a \wedge s \mid s \in S\}$$

and hence it is a complete Heyting algebra. However, the natural morphisms on Heyting algebras are maps preserving finite meets and arbitrary joins, and not necessarily the continuous maps used as morphisms for topological spaces.

Given $f : X \longrightarrow Y$ continuous, then $f^{-1}(U)$ is open, and thus f^{-1} can be viewed as a map $\Omega(Y) \longrightarrow \Omega(X)$, which clearly preserves finite meets and infinite joins. We shall sometimes distinguish between the category of *frames*, which are complete Heyting algebras with morphisms preserving finite meets and infinite joins, and the category of *locales*, which is the opposite of the category of frames (the morphisms on locales are

[2] This coincides with the usual definition of compactness in terms of existence of a finite subcover for every open cover.

referred to as continuous maps). The functor from topological spaces to locales sends the space X to $\Omega(X)$ and a continuous map $f : X \to Y$ to the function $f^{-1} : \Omega(Y) \longrightarrow \Omega(X)$. We adopt the convention that if $f : A \to B$ is a continuous map of locales, we write $f^* : B \to A$ for the corresponding frame homomorphism.

Lemma 3.5 (Johnstone (9)) *A locale is coherent iff it is isomorphic to the locale of ideals of a distributive lattice.*

3.5 Stone Duality

The importance of *Stone duality* in computer science (9) was first recognised by Abramsky (1, 2); it provides, in the words of the author, “a mechanical way of finding the right logic for the denotational semantics”. The original Stone theorem established a precise correspondence between Boolean algebras and topological spaces (so called *Stone spaces*[3]), where

> *operations of conjunction, disjunction and complement correspond to the set-theoretic operations of intersection, union and complement.*

Stone observed that, for any topological space S, the lattice ClopS of the clopen subsets is a Boolean algebra. He also constructed for an arbitrary Boolean algebra B a topological space SpecB. The construction is as follows: points are Boolean algebra homomorphisms $f : B \longrightarrow \mathbf{2}$ where $\mathbf{2} = \{0, 1\}$ is a two-element lattice. Each such f can be identified with the inverse image $f^{-1}(1) \subseteq B$, which is a prime filter. This allows to define the topology by

$$U_a \quad = \quad \{x \in \mathit{SpecB} \mid a \in x\}$$

for $a \in B$. Furthermore, there are isomorphisms:

$$\begin{array}{ll} B \cong \mathit{ClopSpecB} & b \mapsto \{x \in \mathit{SpecB} \mid b \in x\} \\ S \cong \mathit{SpecClopS} & x \mapsto \{U \in \mathit{ClopS} \mid s \in U\} \end{array}$$

which can be expressed as a duality of categories:

$$\mathrm{Bool}^{\mathrm{op}} \quad \cong \quad \mathrm{Stone}.$$

A variety of Stone-duality-type theorems have been proposed to date (9), typically identifying a category of axiomatic systems with a dual of a category of certain topological spaces. The duality that is of interest to us in this paper is between distributive lattices and coherent spaces. Let A be a distributive lattice. The space SpecA (*prime spectrum*) is the set of prime filters over A with the topology generated by:

$$U_a \quad = \quad \{x \in \mathit{SpecA} \mid a \in x\}$$

[3]Hausdorff and coherent spaces; see (9) for alternative characterizations.

for $a \in A$. On the other hand, the *points* $Pt\mathcal{I}(A)$ of $\mathcal{I}(A)$ are the completely prime filters over $\mathcal{I}(A)$ ($\mathcal{I}(A)$ is a complete Heyting algebra) with the topology given by the basis:

$$U_a = \{x \in Pt\,\mathcal{I}(A) \mid a \in x\}.$$

Theorem 3.6 (Johnstone (9))

1. *If A is a distributive lattice then $Spec A \cong Pt\mathcal{I}(A)$*

2. *The duality theorem for distributive lattices is:*

$$\mathrm{DLat}^{\mathrm{op}} \cong \mathrm{CohLoc} \cong \mathrm{CohSp}$$

   where CohSp *is the category of coherent $\mathcal{T}_0$-spaces with continuous maps which preserve compact-open subsets under inverse image, and* CohLoc *is the image of* DLat *through the functor $\mathcal{I}$ (ideal completion).*

As observed in (2), the significance of this theorem is that finitary syntax (specifically finite disjunctions) suffices. This is because the duality allows to recover the original lattice from its prime spectrum by taking the set $\mathcal{K}\Omega(X)$ of the compact-open subsets; the latter can be identified with finite disjunctions of compact elements. More precisely, there are isomorphisms (A a distributive lattice, X coherent topological space):

$$\begin{array}{ll} A \cong \mathcal{K}\Omega(Spec A) & a \mapsto \{x \in Spec A \mid a \in x\} \\ X \cong Spec\mathcal{K}\Omega(X) & x \mapsto \{U \in \mathcal{K}\Omega(X) \mid x \in U\} \end{array}$$

Theorem 3.7 (Abramsky (2)) *Let A be a countable distributive lattice. Say that $a \in A$ is* coprime *if $a \sqsubseteq \sqcup_{i=1}^{n} b_i$ implies $a \sqsubseteq b_i$ some $1 \leq i \leq n$. Then $Spec A$ is a coherent algebraic domain in its Scott topology iff 1 is coprime, and every element $a \in A$ is expressible as a finite join $\sqcup_{i=1}^{n} b_i$ for b_i coprime.*

The isomorphism should be understood as isomorphism of ordered spaces, where the order is the *specialization* order defined for the space X by:

$$x \leq y \iff \forall O \in \Omega(X). x \in O \Rightarrow y \in O$$

(this is equivalent to stating that $x \leq y$ iff x is contained in the closure of y). The significance of the theorem is due to the fact that the specialization order on the Scott topology (of an algebraic cpo) coincides with the original order, and hence Scott domains can be viewed either as topological spaces or partial orders.

Stone-duality-type theorems allow two different perspectives: the *localic* side (a formal system with axioms and rules), and the *spatial* side (a model), each being determined by the other up to isomorphism. In computer science, where models correspond to denotational semantics and axiomatic systems define program logics, this "provides the right framework for understanding the relationship between denotational semantics and program logics" (2).

4 CONSTRAINT SYSTEMS

4.1 Elementary Constraint Systems

We introduce a notion of a pre-constraint system, which is an axiomatic system involving a set A of propositions and an entailment preorder $\vdash$. We assume A is a countable set of terms built from *atomic* propositions ϕ and constants tt and ff by means of binary operations such as conjunction $\wedge$ and disjunction $\vee$.

Definition 4.1 *An* elementary pre-constraint system *is a tuple* $E = \langle A, \vdash, tt, ff, \wedge, \vee \rangle$ *such that:*

1. *A is a countable set of tokens (propositions) over syntax:*

$$A \quad ::== \quad tt \mid ff \mid \phi \mid \phi \wedge \phi \mid \phi \vee \phi$$

2. $\vdash \subseteq A \times A$ *is an entailment relation (preorder);*

3. $\wedge : A \times A \to A$ *and* $\vee : A \times A \to A$ *are binary conjunction and disjunction operations;*

4. ff *is coprime;*

5. *if* ϕ *is coprime and* $\bigwedge_{i \in I} \psi_i \vdash \phi$ *(I finite index set) then* $\psi_i \vdash \phi$ *for some* $i \in I$*;*

6. *every* ϕ *is expressible as* $\wedge_{i=1}^{n} \psi_i$ *with* ψ_i *coprime;*

subject to the set of axioms shown in Table 1 being satisfied.

By $\phi \vdash \psi$, where $\phi, \psi \in A$, we mean ϕ implies ψ (ϕ "is larger than" ψ). The entailment relation is a preorder, rather than a partial order, so that account is taken of different syntactic forms of logical formulae. The conjunction $\phi \wedge \psi$ of two tokens ϕ and ψ is "larger than" both ϕ and ψ with respect to the entailment order $\vdash$, and thus it behaves like the join operation. The equivalence relation $\approx$ is a congruence.

The *Lindenbaum algebra* of an elementary pre-constraint system $E = \langle A, \vdash, tt, ff, \wedge, \vee \rangle$ is defined to be the quotient structure:

$$\mathcal{LA}(E) \quad = \quad \langle A/_{\approx}, \vdash/_{\approx}, [tt], [ff], \wedge/_{\approx}, \vee/_{\approx} \rangle$$

where $\vdash/_{\approx}$, $\wedge/_{\approx}$ and $\vee/_{\approx}$ are the induced partial order, meet and join on the $\approx$-quotient of A defined by:

1. $[\phi] \vdash/_{\approx} [\psi]$ iff $\phi \vdash \psi$;

Table 1: Axioms for elementary pre-constraint systems.

(RF)	$\phi \vdash \phi$	(TR)	$\dfrac{\phi \vdash \psi, \psi \vdash \theta}{\phi \vdash \theta}$
($\approx$-I)	$\dfrac{\phi \vdash \psi, \psi \vdash \phi}{\phi \approx \psi}$	($\approx$-E)	$\dfrac{\phi \approx \psi}{\phi \vdash \psi, \psi \vdash \phi}$
(tt-I)	$\phi \vdash tt$	(ff-E)	$ff \vdash \phi$
($\wedge$-I)	$\dfrac{\phi \vdash \psi, \phi \vdash \theta}{\phi \vdash \psi \wedge \theta}$		
($\wedge$-E-L)	$\phi \wedge \psi \vdash \phi$	($\wedge$-E-R)	$\phi \wedge \psi \vdash \psi$
($\vee$-I)	$\dfrac{\phi \vdash \theta, \psi \vdash \theta}{\phi \vee \psi \vdash \theta}$		
($\vee$-E-L)	$\phi \vdash \phi \vee \psi$	($\vee$-E-R)	$\psi \vdash \phi \vee \psi$
($\wedge$-DST)	$\phi \wedge (\psi \vee \theta) \vdash (\phi \wedge \psi) \vee (\phi \wedge \theta)$		

2. $[\phi] \wedge/_{\approx} [\psi] = [\phi \wedge \psi]$.
3. $[\phi] \vee/_{\approx} [\psi] = [\phi \vee \psi]$.

That $\wedge/_{\approx}$ and $\vee/_{\approx}$ are well-defined follows from the fact that $\approx$ is a congruence wrt $\wedge$ and $\vee$.

Definition 4.2 *The Lindenbaum algebra $\mathcal{LA}(E) = \langle A/_{\approx}, \sqsubseteq, [tt], [ff], \sqcup, \sqcap \rangle$ of some elementary pre-constraint system E will be called an* elementary constraint system.

Note that we have introduced new symbols: $\sqsubseteq$ will denote $\vdash/_{\approx}$, while $\sqcup$ and $\sqcap$ will denote $\wedge/_{\approx}$ and $\vee/_{\approx}$ respectively. Also, if it is clear from the context, we shall shall often omit square brackets when referring to elements of an elementary constraint system, *i.e.* we shall use ϕ instead of the $\approx$-equivalence class $[\phi]$.

Observe that the Lindenbaum algebra $\mathcal{LA}(E)$ has the structure of a (coherent algebraic) distributive lattice[4] with two distinguished elements (0 identified with $[tt]$ and 1 with $[ff]$), $[\phi] \sqcup [\psi]$ interpreted as $[\phi \wedge \psi]$, and $[\phi] \sqcap [\psi]$ interpreted as $[\phi \vee \psi]$. Note that coprimes

[4]In concurrent constraint programming, a semilattice structure with tt, ff and $\wedge$ often suffices. It is possible to give Stone duality for semilattices, see *e.g.* (20), but we shall not consider this case. We should point out, however, that distributivity plays an important part in duality.

are essential as demonstrated by Theorem 3.7. It is also possible to axiomatize the notion of coprimeness in the style of (2).

Note also that $\mathcal{LA}(E)$ is not, in general, complete – only *finite* sets of tokens are guaranteed to have joins. Before we explain how to allow joins of infinite sets of tokens, let us consider some examples of elementary pre-constraint systems, which we have simplified by omitting disjunction.

Example 4.3 (First-order constraint system) *Let Var be a non-empty set of variables* $x, y, z, \ldots$. *Let* Σ *be a (many sorted) first-order alphabet (function symbols* $f, g, \ldots$, *predicate symbols and their signature). Let* Φ *be a non-empty subset of* (Var, Σ)*-formulae closed under conjunction, and let* $\vdash$ *to be interpreted as the usual implication. Then*

$$E_M = \langle \Phi \cup \{tt, f\!f\}, \vdash, tt, f\!f, \wedge \rangle$$

is an elementary pre-constraint system.

Example 4.4 (Mutual exclusion) *Let* c_1, c_2 *denote variable names in some concurrent programming language,* t *and* f *the truth and falsity constants in this language. Consider the set* A_M *of terms of the form:*

$$A_M = \{c_1 = t, c_1 = f, c_2 = t, c_2 = f, tt, f\!f\}$$

and the elementary pre-constraint system:

$$E_M = \langle A_M^{\wedge}, \vdash, tt, f\!f, \wedge \rangle$$

where $\vdash$ *and* $\wedge$ *are the usual implication and conjunction, and* $A_M^{\wedge}$ *denotes the set* A_M *closed under finite conjunctions. If* $c_1 = t$ *denotes that the process number* 1 *is in its critical section, and similarly,* $c_2 = t$ *denotes that the process number* 2 *is in its critical section, then:*

1. $c_1 = t \wedge c_2 = f$ *denotes that only the process number* 1 *is in its critical section;*

2. $c_1 = t \wedge c_1 = f$ *entails* $f\!f$ *(failure);*

3. $c_1 = t \wedge c_2 = t$ *denotes the situation in which the* mutual exclusion *protocol is violated as both processes are in their critical sections.*

Example 4.5 (Infinitely Often) *Let* e_1, e_2 *denote variable names in some concurrent programming language, and* $n \in \mathbf{N}$ *denotes a natural number constant. Consider the set* A *of terms of the form:*

$$A_I = \{e_1 \geq n \mid n \in \mathbf{N}\} \cup \{e_2 \geq n \mid n \in \mathbf{N}\} \cup \{tt, f\!f\}$$

and the elementary pre-constraint system:

$$E_I = \langle A_I^{\wedge}, \vdash, tt, f\!f, \wedge \rangle$$

where $\vdash$ *and* $\wedge$ *are the usual implication and conjunction. If* $e_1 \geq n$ *denotes that the process number* 1 *has entered its critical section* at least n *times, and similarly for the process number* 2*, then:*

1. $e_1 \geq 1$ *denotes that the process number* 1 *has entered its critical section at least once;*

2. $c_1 \geq 1 \wedge c_2 \geq 2$ *denotes that both processes have entered their critical section at least once.*

Observe that we cannot correctly express the property that the process number 1 *has entered its critical section* infinitely often*, since*

$$\bigsqcup\{e_1 \geq n \mid n \in \mathbf{N}\} = f\!f$$

and thus this property is identified with failure.

Example 4.6 (Lists) *Let* A_L *be the set of terms* $\{t = t'\} \cup \{tt, f\!f\}$*, where* t, t' *are terms denoting lists built following the syntax rules below:*

$$\begin{array}{rcll} t & ::== & \mathbf{0} & \textit{list terminator} \\ & \mid & x & \textit{list variable} \\ & \mid & ax & \textit{list consisting of } a \textit{ followed by } x \\ & \mid & bx & \textit{list consisting of } b \textit{ followed by } x \end{array}$$

and $=$ *is the usual equality. Consider the elementary pre-constraint system*

$$E_L = \langle A_L^{\wedge}, \vdash, tt, f\!f, \wedge \rangle$$

where $\vdash$ *and* $\wedge$ *are the usual implication and conjunction. Then*

1. $x_1 = ax_2 \wedge x_2 = bx_3 \wedge x_3 = \mathbf{0}$ *denotes the list* $ab\mathbf{0}$*;*

2. $x_1 = ax_2 \wedge x_2 = ax_3 \wedge ... \wedge x_{n+1} = \mathbf{0}$ *denotes the list* $a^n\mathbf{0}$.

Observe that we cannot correctly express infinite *lists such as* a^ω *(a repeated infinitely often), since the join of the infinite set:*

$$\{x_1 = ax_2 \wedge x_2 = ax_3 \wedge ... \wedge x_n = ax_{n+1} \wedge ...\} \tag{4}$$

will be $f\!f$.

4.2 Adding Infinite Constraints

Let $E = \langle A, \vdash, tt, ff, \wedge, \vee \rangle$ be an elementary pre-constraint system. E is a *formal*, deductive system, where the entailment relation $\vdash$ is a preorder. We shall require a *semantic* model $\mathcal{C}(E) = \langle C, \sqsubseteq \rangle$ for E, where $\sqsubseteq$ is a partial order. Thus, we shall require a function:

$$\kappa(\cdot) : A \longrightarrow \mathcal{K}(C) \tag{5}$$

where $\mathcal{K}(C)$ are the compact elements of $\mathcal{C}(E)$. But what should the domain $\mathcal{C}(E)$ look like? One would expect to be able to prove, at least, soundness and completeness theorems. One way of constructing such a model is by taking the ideal completion (see Section 3.1 for basic definitions and facts) of the (Lindenbaum algebra) of E. We therefore adopt the following definition.

Define the interpretation $\kappa(\cdot) : A \longrightarrow \mathcal{K}(C)$ by:

$$\kappa(\phi) \quad = \quad \downarrow[\phi]$$

i.e. $\kappa(\cdot)$ assigns to each ϕ the principal ideal of the Lindenbaum algebra $\mathcal{LA}(E)$ determined by ϕ[5]. The ordering on $\mathcal{K}(C)$ is by subset inclusion.

Definition 4.7 *Let $E = \langle A, \vdash, tt, ff, \wedge, \vee \rangle$ be an elementary pre-constraint system. A* constraint system *$\mathcal{C}(E) = \langle C, \sqsubseteq \rangle$ determined by E is the ideal completion*

$$\langle \mathcal{I}(\mathcal{LA}(E)), \subseteq \rangle$$

of the Lindenbaum algebra of E. $\mathcal{C} = \langle C, \sqsubseteq \rangle$ is said to be a constraint system *iff it is the ideal completion of some elementary constraint system.*

We adopt the following notation convention. Elements of C will be referred to as *constraints*, and will be denoted by letters c, d, etc., or, where confusion does not arise, by their syntactic representation $\phi \in A$. For example, tt will be used to denote the principal ideal $\{tt\}$ determined by the proposition tt.

We shall now motivate the choice of ideals for the construction of a constraint system. Suppose I is an ideal in $\mathcal{LA}(E)$, then it is also an $\vdash$-ideal in E, and hence by definition:

1. I must contain tt;
2. if $\phi \in I$ and $\phi \vdash \psi$ then ψ must also be in I;
3. if $\phi, \psi \in I$ then $\phi \wedge \psi$ must be in I.

[5]Strictly speaking, the mapping is $\phi \mapsto [\phi] \mapsto \downarrow[\phi]$, where $\phi \mapsto [\phi]$ is the canonical valuation. The usual inductive definition of $\kappa(\cdot)$ has been replaced with the direct definition because it can be shown that if atomic formulae are mapped onto their canonical valuation, then so are all the formulae.

Thus, I constitutes a *deductively closed theory* in E.

The following result is immediate from the construction.

Proposition 4.8

1. *$\langle C, \sqsubseteq\rangle$ is an algebraic lattice.*

2. *The Lindenbaum algebra $\langle \mathcal{LA}(E), \sqsubseteq\rangle$ is isomorphic to $\langle \mathcal{K}(C), \sqsubseteq\rangle$ via the isomorphism $[\phi] \mapsto \downarrow[\phi]$.*

Proof: Direct from Lemma 3.2. $\square$

The importance of the above theorem lies in the fact that concurrent constraint systems are freely constructed from elementary constraint systems. In other words, there is just enough (*i.e.* no more, no less) structure in $\langle C, \sqsubseteq\rangle$. The isomorphism between the Lindenbaum algebra of an elementary pre-constraint system and the compact elements states that each point of the constraint system C is uniquely determined by a theory in E.

Furthermore, we can show that the constraint system $\langle C, \sqsubseteq\rangle$ admits join and meet operations, which can be characterized as follows. Note that, although the join of two constraints exists, it does not coincide with their set-theoretic union.

Proposition 4.9

1. *For $c, d \in C$, $c \sqcap d$ is their intersection.*

2. *For $c, d \in C$, $c \sqcup d$ is their union closed under finite joins.*

Proof:

1. Assume c, d respectively denote ideals $I, J \in \mathcal{I}(\mathcal{LA}(E))$. It is easy to see that $I \cap J$ is also an ideal (it is a lower set and $a, b \in I \cap J$ implies $a \sqcup b \in I \cap J$). Since for any ideal $I' \subseteq I, J$ we have $I' \subseteq I \cap J$, it follows that $I \cap J$ is the largest such ideal, and hence the meet of I and J.

2. Assume c, d respectively denote ideals $I, J \in \mathcal{I}(\mathcal{LA}(E))$. Then it is easy to see that $(I \cup J)^{\sqcup}$, *i.e.* the set $I \cup J$ closed under finite joins, is an ideal. Take any $I' \supseteq I, J$, then $I' \supseteq I \cup J$, and hence $I' = I'^{\sqcup} \supseteq (I \cup J)^{\sqcup}$ as required.

□

We can now show the following result.

Theorem 4.10 (Soundness and completeness) *For all* $\phi, \psi \in A$*:*

$$\phi \vdash \psi \iff \kappa(\phi) \sqsupseteq \kappa(\psi).$$

Proof:

1. We verify that axioms are satisfied. (TR), (RF), ($\approx$-I) and ($\approx$-E) follow by transitivity, reflexivity and antisymmetry of $\subseteq$. (tt-I) holds since $\kappa(tt) = \downarrow[tt] = \{tt\} \subseteq \downarrow[\phi]$ for all ϕ. Likewise, (tt-E) holds since $\kappa(ff) = \downarrow[ff] = A/_{\approx} \supseteq \downarrow[\phi]$ for all ϕ. We have $\kappa(\phi \wedge \psi) = \downarrow[\phi \wedge \psi] = \downarrow([\phi] \sqcup [\psi]) = \downarrow[\phi] \cap \downarrow[\phi] = \kappa(\phi) \sqcup \kappa(\psi)$, so $\kappa(\phi) \sqsupseteq \kappa(\psi)$ and $\kappa(\phi) \sqsupseteq \kappa(\theta)$ iff $\kappa(\phi) \sqsupseteq \kappa(\psi) \sqcup \kappa(\theta) = \kappa(\psi \wedge \theta)$, and the $\wedge$ axioms follow. The case of $\vee$ is similar. Finally, distributivity axiom follows from distributivity of the underlying lattice.
2. Suppose $\kappa(\phi) \sqsupseteq \kappa(\psi)$. Then by definition $\downarrow[\phi] \supseteq \downarrow[\psi]$, which implies $\phi \vdash \psi$.

□

Notice that the soundness and completeness theorem makes use of the order inherent in the lattice $\mathcal{C}(E)$, rather than an appropriate set-theoretic operation (*i.e.* containment) as required by Stone duality. Also note the order reversal. This reversal can be understood on the grounds of the Scott topology of $\mathcal{C}(E)$. Recall that the Scott topology (of an algebraic cpo) is determined by the basis $\{\uparrow b \mid b \in \mathcal{K}(C)\}$. It is easy to see that for any poset:

$$c \sqsubseteq d \iff \uparrow c \subseteq \uparrow d \tag{6}$$

and hence we can reformulate soundness as follows.

Corollary 4.11 (Soundness wrt Scott topology) *For all* $\phi, \psi \in A$*:*

$$\phi \vdash \psi \ \Rightarrow\ \uparrow\kappa(\phi) \subseteq \uparrow\kappa(\psi)$$

Moreover, we have the following results. The definability theorem states that every compact semantic constraint can be expressed as a formula in E. Although this does not, in general, hold for an arbitrary semantic constraint, we can characterize such a constraint as the join of the compact constraints below it. In other words, each constraint c is the join of all the (equivalence classes of) formulae in E which are below c.

Theorem 4.12 (Definability) $\forall c \in \mathcal{K}(C), \exists \phi \in A.\kappa(\phi) = c$.

Proof: Direct from Lemma 3.2. $\square$

Theorem 4.13 (Approximation) $\forall d \in C.d = \bigsqcup\{c \in \mathcal{LA}(E) \mid c \sqsubseteq d\}$.

Proof: Direct from Lemma 3.2 since $\mathcal{LA}(E)$ is isomorphic with the compact elements of the algebraic lattice $\mathcal{C}(E)$. $\square$

Once again, the above theorem makes use of the inherent order in the lattice C, whereas one would expect conjunction to be interpreted in terms of a set-theoretic intersection, as required by Stone duality. As in the case of the entailment - containment correspondence, we can show that this view can be understood on the grounds of the Scott topology. Note that for any poset we have:

$$\uparrow(b \sqcup c) = \uparrow b \cap \uparrow c \tag{7}$$

whenever $b \sqcup c$ is defined, so the following can be shown.

Corollary 4.14 (Approximation wrt Scott topology)

$$\forall d \in C.\uparrow d \quad = \quad \bigcap\{\uparrow c \mid c \sqsubseteq d\}.$$

Observe that the approximation theorem only guarantees that each constraint c in a concurrent constraint system $\langle C, \sqsubseteq\rangle$ can be approximated as a join of (a possibly infinite) set of (equivalence classes of) formulae of A. It does not, however, guarantee that *every* constraint $c \in C$ can be syntactically expressed as a formula in A.

The remaining question is to find a complete model for E, in which the operations of $\wedge$ and $\vee$ have been replaced by the set-theoretic intersection and union. We can utilize Stone duality for this purpose, thus interpreting formulae as the compact-open subsets of $Spec\mathcal{LA}(E)$. Recall that the space $Spec\mathcal{LA}(E)$, the prime filters over $\mathcal{LA}(E)$ with inclusion of filters as the specialization ordering, is isomorphic to the points (completely prime filters) of the ideal completion of E (Theorem 3.6), which allows us to identify $Spec\mathcal{LA}(E)$ with $Pt\,\mathcal{C}(E)$. Since *ff* is coprime, and every element of the lattice $\mathcal{LA}(E)$ is expressible as $\sqcup_{i=1}^{n}\psi_i$ for ψ_i coprime, there are enough coprimes, and hence $Spec\mathcal{LA}(E)$ is a coherent domain in its Scott topology. It follows (Theorem 3.7) that the usual topology on $Spec\mathcal{LA}(E)$ coincides with the Scott topology $\sigma\Omega Spec\mathcal{LA}(E)$.

This leads us to the following definition. Note that the basic open sets of the Scott topology $\sigma\Omega Spec\mathcal{LA}(E)$ have the form:

$$\uparrow_{Spec\mathcal{LA}(E)}(\uparrow_{\mathcal{LA}(E)}(\phi))$$

for $\phi \in \mathcal{LA}(E)$ coprime, *i.e.* they are upper-closed sets of prime filters over $\mathcal{LA}(E)$.

Let $E = \langle A, \vdash, tt, ff, \wedge, \vee \rangle$ be a pre-constraint system, and let $\mathcal{C}(E) = \langle C, \sqsubseteq \rangle$ be the constraint system determined by E. Define an interpretation map:

$$[\![\cdot]\!] : A \longrightarrow \mathcal{K}\Omega(Spec\mathcal{LA}(E))$$

by

$$\begin{array}{rcl} [\![ff]\!] & = & \emptyset \\ [\![tt]\!] & = & Spec\mathcal{LA}(E) \\ [\![\phi]\!] & = & \uparrow_{Spec\mathcal{LA}(E)}(\uparrow_{\mathcal{LA}(E)}[\phi]) \quad \phi \text{ coprime} \\ [\![\phi \wedge \psi]\!] & = & [\![\phi]\!] \cap [\![\psi]\!] \\ [\![\phi \vee \psi]\!] & = & [\![\phi]\!] \cup [\![\psi]\!] \end{array}$$

This is the localic side of Stone duality.

Theorem 4.15 (Stone duality) *Let $E = \langle A, \vdash, tt, ff, \wedge, \vee \rangle$ be a pre-constraint system, $\mathcal{C}(E) = \langle C, \sqsubseteq \rangle$ be the constraint system determined by E. Then:*

$$\mathcal{LA}(E) \;\; \cong \;\; \mathcal{K}\Omega(Spec\mathcal{LA}(E)).$$

Proof: First observe that from coherence the Scott-compact opens form a distributive lattice under the converse of inclusion, with $\emptyset$ acting as 1 and $Spec\mathcal{LA}(E) = \uparrow tt$ as 0. Thus, $\langle \mathcal{K}\Omega(Spec\mathcal{LA}(E)), \supseteq, Spec\mathcal{LA}(E), \emptyset, \cap, \cup \rangle$ is an elementary constraint system.

It remains to show that $[\![\cdot]\!]$ is a bijection preserving the distinguished elements and the lattice operations. It is easy to see that $[\![\cdot]\!]$ is injective and preserves the distinguished elements and both operations. That $[\![\cdot]\!]$ is surjective follows from the fact that every compact-open in $Spec\mathcal{LA}(E)$ (which is coherent algebraic) has a unique, irredundant, and finite coprime decomposition (see (22)). $\square$

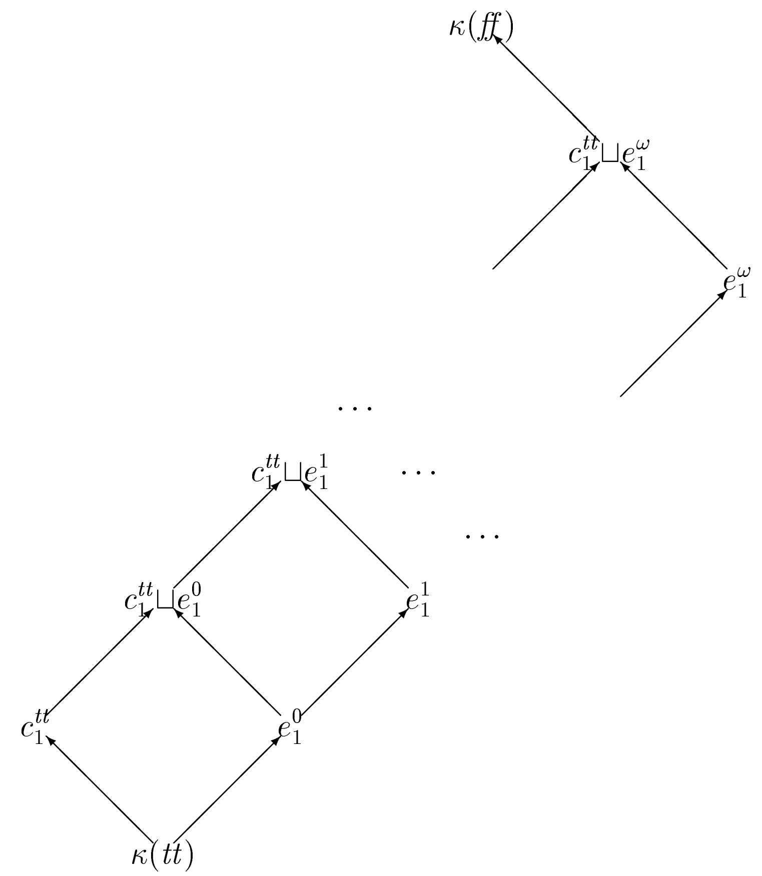
As a consequence, we obtain the following correspondence between entailment of formulae and inclusion of the interpreted constraints.

Corollary 4.16 $\forall \phi, \psi \in A. \phi \vdash \psi$ *iff* $[\![\phi]\!] \subseteq [\![\psi]\!]$.

Example 4.17 (Mutual Exclusion and Infinitely Often) *Consider the elementary pre-constraint system:*

$$E = \langle (A_M \cup A_I)^{\wedge}, \vdash, tt, ff, \wedge \rangle \tag{8}$$

where A_M is the set of formulae introduced in Example 4.4, and A_I is the set of formulae from Example 4.5. The following shows a part of the constraint system determined by E:



where c_1^{tt} *denotes* $\kappa(c_1 = tt)$, e_1^0 *denotes* $\kappa(e_1 = 0)$, $c_1^{tt} \sqcup e_1^0$ *denotes* $\kappa(c_1 = tt \wedge e_1 = 0)$, *etc. Note that* $c_1^{tt} \wedge e_1^{\omega}$ *and* e_1^{ω} *are* not *compact, while* $\kappa(ff)$ *is.*

4.3 Cylindric Constraint Systems

As many programming languages come equipped with a notion of hiding information within a module from the rest of the program, we shall introduce a kind of a constraint system which supports *hiding operators*. Our definition follows the approach of (19).

Definition 4.18 *A* cylindric pre-constraint system *is a tuple* $E = \langle A, Var, \vdash, tt, ff, \wedge, \vee \rangle$ *such that*

1. *A is a set of propositions over syntax:*

$$A \quad ::== \quad tt \mid ff \mid \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists X.\phi$$

2. *Var is an infinite set of variables,*

3. $\langle A, \vdash, tt, ff, \wedge, \vee \rangle$ *is an elementary pre-constraint system*

subject to the set of axioms shown in Table 2.

Table 2: Additional axioms for cylindric pre-constraint systems.

(∃-I)	$\phi \vdash \exists X.\phi$	(∃-⊢)	$\dfrac{\phi \vdash \psi}{\exists X.\phi \vdash \exists X.\psi}$
(∃-∧)	$\exists X.(\phi \wedge \exists X.\psi) \approx \exists X.\phi \wedge \exists X.\psi$	(∃-COM)	$\exists X.(\exists Y.\phi) \approx \exists Y.(\exists X.\phi)$

Definition 4.19 *Let $E = \langle A, \mathit{Var}, \vdash, tt, \mathit{ff}, \wedge\rangle$ be a cylindric pre-constraint system. A* cylindric constraint system *$\mathcal{C}(E) = \langle C, \sqsubseteq\rangle$ determined by E is the ideal completion*

$$\langle \mathcal{I}(\mathcal{LA}(E)), \subseteq\rangle$$

of the Lindenbaum algebra of E.

As observed in (19), cylindric pre-constraint systems determine constraint systems $\langle C, \sqsubseteq\rangle$ which are equipped with a family $\{\exists_X \mid X \in \mathit{Var}\}$ of operators $\exists_X : C \longrightarrow C$, which map a formula ϕ onto its existential closure $\exists X.\phi$.

Proposition 4.20 *Let $E = \langle A, \mathit{Var}, \vdash, tt, \mathit{ff}, \wedge, \vee\rangle$ be a cylindric pre-constraint system. Define $\exists^0_X : \mathcal{K}(C) \longrightarrow \mathcal{K}(C)$ by:*

$$\exists^0_X(\phi) \quad = \quad \exists X.\phi$$

Then $\exists^0_X$ is monotone, and $\exists_X$, the continuous extension of $\exists^0_X$ to C, is a kernel operator.

Proof: We have $\phi \vdash \psi$ implies $\exists X.\phi \vdash \exists X.\psi$ from axiom (∃-⊢), and hence $\exists^0_X(\phi) \sqsupseteq \exists^0_X(\psi)$.

Since $\exists^0_X$ is monotone on compact elements, its continuous extension $\exists_X$ to C exists and is unique. We show $\exists_X$ is idempotent. $\phi \vdash \exists X.\phi$ implies $\exists X.\phi \vdash \exists X.(\exists X.\phi)$ from axiom (∃-⊢), and hence $\exists_X(\phi) \sqsupseteq \exists_X(\exists_X(\phi))$. The other direction follows similarly. Finally, since $\phi \vdash \exists X.\phi$ we have $\phi \sqsupseteq \exists_X(\phi)$. $\square$

Another important kind of constraint systems are those with the "diagonal formulae" due to Tarski. These are terms of the form $X = Y$, where $X, Y \in \mathit{Var}$. They have been found useful when dealing with variable substitution, which in turn is used for parameter passing to and from procedures (19). Given a formula ϕ, the formula $\phi[Y/X]$, for $X, Y \in \mathit{Var}$, denotes ϕ with all the (free) occurrences of the variable X replaced with Y. Using existential quantification, $\phi[Y/X]$ can be modelled as $\exists X.(X = Y) \wedge \phi$. This leads us to the following definition.

Table 3: Additional axioms for diagonal pre-constraint systems.

($X = X$-I) $\quad \phi \vdash X = X$	(Z-I) $\quad X = Y \approx \exists Z.(X = Y \wedge Y = Z)$
($X = Y$-E) $\quad X = Y \wedge \exists X.(\phi \wedge X = Y) \vdash \phi$	

Definition 4.21 *A* diagonal pre-constraint system *is a tuple* $E = \langle A, \mathit{Var}, \vdash, tt, \mathit{ff}, \wedge, \vee \rangle$ *such that*

1. *A is a set of propositions over syntax:*

$$A \quad ::== \quad tt \mid \mathit{ff} \mid \phi \mid \phi \wedge \phi \mid \exists X.\phi \mid X = Y$$

2. *$(X, Y \in)\, \mathit{Var}$ is an infinite set of variables,*

3. *$\langle A, \vdash, tt, \mathit{ff}, \wedge, \vee \rangle$ is a cylindric pre-constraint system,*

subject to the set of axioms shown in Table 3.

Definition 4.22 *Let $E = \langle A, \mathit{Var}, \vdash, tt, \mathit{ff}, \wedge, \vee \rangle$ be a diagonal pre-constraint system. A* diagonal constraint system $\mathcal{C}(E) = \langle C, \sqsubseteq \rangle$ *determined by E is the ideal completion*

$$\langle \mathcal{I}(\mathcal{LA}(E)), \subseteq \rangle$$

of the Lindenbaum algebra of E.

We shall denote the semantic constraints corresponding to propositions of the form $X = Y$ by d_{XY}.

Unless otherwise stated, we shall assume that E denotes a cylindric, diagonal pre-constraint system.

5 CONSTRAINTS, FINITE OBSERVATIONS, AND PROPERTIES

Let $\mathcal{C}(E)$ be a constraint system. Define a *semantic property* Ψ to be any subset of C. This is an extensional, non-constructive view; note that we are not requiring at this stage that each property be expressed as a formula in E. By definition, a semantic property may contain any combination of compact elements contained in $\mathcal{K}(C)$, and the non-compact elements $C \setminus \mathcal{K}(C)$. We shall find that this is rather general, and thus will aim to restrict the approach later by allowing some "basic" properties and certain constructions from them.

It would be convenient to distinguish between properties that are detectable in finite time (so called "finitary") from those which cannot be detected at finite time ("infinitary"). One possible distinction would be by identifying subsets of $\mathcal{K}(C)$ with finitary properties, and subsets of $C \setminus \mathcal{K}(C)$ with infinitary ones (see *e.g.* (15)). This may be justified by the fact that, by definition of compactness, each compact element is the supremum of a finite chain of elements, and thus can be reached in a finite number of "computation steps", where a computation step corresponds to moving up to the next element in the chain. On the other hand, a non-compact element is a supremum of an infinite chain, and hence cannot be reached by a finite number of computation steps. However, we shall find that allowing infinite (*i.e.* non-compact) elements in finitary properties, and compact elements in infinitary properties is advantageous. Therefore, we define a semantic property $\Psi \subseteq C$ to be *finitary* iff:

$$c \in \Psi \cap (C \setminus \mathcal{K}(C)) \iff \exists c' \in (\Psi \cap \mathcal{K}(C)).c' \sqsubseteq c$$

Otherwise, the property Ψ is *infinitary*.

The intuition behind the above definition is as follows. A finitary property is one which may contain infinite elements *only* if some compact element below is also in the property. Thus, a finitary property can always be decided in finite time[6] (in the sense of semi-decidability). On the other hand, given an infinitary property, it is possible that it will not be detected in finite time. We shall find that properties like "mutual exclusion" or "termination" will be finitary, while fairness will not.

Example 5.1 *Consider the constraint system shown in Example 4.17. Then* $\downarrow(c_1^{tt} \wedge e_1^0)$, $\uparrow(c_1^{tt} \wedge e_1^0)$, $\{\kappa(ff)\}$ *are finitary,* $\uparrow c_1^{tt} \wedge e_1^{\omega}$ *is infinitary.*

Recall that safety properties state that "nothing bad will happen", while liveness that "something good will happen". It seems that we need to state what each such "thing" is, and also define what it means for each "thing" to be "good" or "bad". Following (21), observations must correspond to open sets in some topology over C. But which topology?

It is natural to require that a single "thing" must be detectable in finite time, and hence be (a subset of C) determined by an element of $\mathcal{K}(C)$. For $c \in \mathcal{K}(C)$ call such subset U_c a *basic observation* of c. If $d \sqsupseteq c$, then we would like d to belong to U_c; this guarantees that to observe something above c (*i.e.* stronger than c in the sense of the entailment relation), it is sufficient to observe c. It is easy to see that for $c \in \mathcal{K}(C)$ we can now define a basic observation U_c to be $\uparrow c$. Note that to observe c *and* to observe

[6] We are assuming that satisfaction corresponds to containment.

d means to be inside $U_c \cap U_d = U_{s \sqcup d}$, and hence basic observations should be closed under (finite) intersections.

Ideally, we would like to be able to observe collections of "things", in the sense that to observe $\{c, d\}$ means to observe either c or d. Thus, we would expect $U_{\{c,d\}} = U_c \cup U_d$. This immediately gives rise to an identification of observations with a topology (the Scott topology) on C. The problem is whether we should restrict the size of collections of "things" in any way – should they be finite, countable or arbitrary sets of compact elements? Since we would like observations to be syntactically expressible as formulae of E, and our axiomatization permits only finitely expressible constraints, which are moreover identified with the compact open subsets, we propose the following definition.

Definition 5.2 *Define a* finite observation *in the constraint system $\mathcal{C}(E) = \langle C, \sqsubseteq \rangle$ to be a Scott-compact open subset of C.*

Thus finite observations are closed under finite intersection and finite union. Moreover, by Stone duality, each finite observation has a syntactic form. Note that this definition does not allow arbitrary open sets as finite observations. This is not to say that open sets, which are finitary, are not observable - they are, in the sense of semi-decidability, but what concerns us here is both *observability* and *finiteness* of syntactic definition, and, in particular, finite disjunctions. However, we should point out that such a restriction may be considered too strong, and arbitrary, or at least countable, collections of "things" may be allowed (13).

As discussed earlier, we have We remark that, in the light of Stone duality, perhaps we should have identified finite observations with the Scott-compact open subsets $U \in \mathcal{K}\Omega(Pt\mathcal{C}(E))$. However, since we are working with coherent algebraic locales, any such U is an intersection of a $U' \in \mathcal{K}\Omega(\mathcal{C}(E))$ with $Pt, \mathcal{C}(E)$, so we can work with the Scott topology of $\mathcal{C}(E)$.

6 PROCESS LANGUGAGE

6.1 Operational Semantics

We now introduce a concurrent constraint language $\mathcal{L}$ with angelic non-determinism, similar to the language introduced in (8). The language is interpreted over the disjunctive, cylindric and diagonal constraint system $\langle C, \sqsubseteq \rangle$, and is given by the following syntax:

$$A ::== \mathit{tell}(c) \mid \mathit{ask}(c) \to A \mid A \oplus A \mid A \parallel A \mid \exists X.A \mid p(Y)$$

where $c \in \mathcal{K}(C)$. Each such c will denote the state of the store at some *finite* time, and we shall often use constraints to describe such finite stores. As we allow infinite constraints $c \in C$, we may also refer to *infinite* stores not reachable in finite time.

For $c \in \mathcal{K}(C)$, $tell(c)$ denotes the action which conjoins the constraint c to the current store. The guard $ask(c) \to A$ denotes the action, which will behave as the agent A if run in the store which entails d with $d \vdash c$; otherwise, the process will be suspended until such condition holds. $A \oplus A$ denotes angelic non-determinism, in the sense of (8); a choice will be made between the two agents, and the other agent will be discarded[7]. $A \parallel A$ will cause the agent to decompose into two concurrent subagents, which can proceed autonomously. $\exists X.A$ will be used for information hiding. $p(Y)$ denotes procedure call, where $Y \in Var$ is the actual parameter. It is straightforward to generalize procedure calls to vectors $\vec{X}$ of variables. To avoid having to use environments, we shall assume a fixed set W of procedure declarations of the form $p(X){:}-A \in W$, $X \in Var$ (X is fixed).

Operational semantics is given in the form of a transition system. The transition system is of the form $\langle Conf, \longrightarrow \rangle$, where $Conf$ is the set of *configurations*, that is pairs consisting of a process and a constraint $c \in \mathcal{K}(C)$. $(P, c) \longrightarrow (P', c')$ means that the agent P when initiated in the store c will upgrade the store to c' and subsequently behave like P'. The transition rules are shown in Table 4. As usual, the transition relation $\longrightarrow$ is defined to be the smallest relation satisfying the axioms. The rules are based on the language (8), with the exception of the procedure call, which has been simplified. The reader should note that, to aid hiding, transition relation has been extended to agents of the form $\exists X.(d, A)$, where d denotes an internal store holding information about X which is hidden outside $\exists X.(d, A)$. An agent of the form $\exists X.A$ evolves into the agent $\exists X.(d, B)$, with d storing the information about the hidden X for b to use.

Unlike the treatment of (4), we shall not distinguish between *successful* and *deadlocking* termination.

6.2 Computations and Observations

Let P be a process (term) in the language $\mathcal{L}$ interpreted over the constraint system $\langle C, \sqsubseteq \rangle$. An *execution sequence of P from $c \in \mathcal{K}(C)$* is a finite or infinite sequence of pairs from $\mathcal{L} \times \mathcal{K}(C)$ of the form:

$$\langle P_1, c_1 \rangle \langle P_2, c_2 \rangle \cdots \langle P_i, c_i \rangle \cdots$$

such that $\forall i$:

- $P_1 = P, c_1 = c$
- $\langle P_i, c_i \rangle \longrightarrow \langle P_{i+1}, c_{i+1} \rangle$

We shall sometimes write execution sequences as:

$$P_1 \stackrel{\langle c_1, c_2 \rangle}{\longrightarrow} P_2 \stackrel{\langle c_2, c_3 \rangle}{\longrightarrow} \cdots P_i \stackrel{\langle c_i, c_{i+1} \rangle}{\longrightarrow} \cdots$$

[7] The reader should note that this is *not* demonic non-determinism, as will be made clear later.

Table 4: Transition rules for $\mathcal{L}$

T1	$\langle tell(d), c\rangle \longrightarrow \langle tell(tt), c \sqcup d\rangle$	**if** $c \neq tt$
T2	$\langle ask(d) \rightarrow A, c\rangle \longrightarrow \langle A, c\rangle$	**if** $d \vdash c$
T3	$\langle p(Y), c\rangle \longrightarrow \langle \exists Y.(tell(X = Y) \parallel A), c\rangle$	**if** $p(X){:}\, -A \in W$
T4	$\dfrac{\langle A, \exists_X c\rangle \longrightarrow \langle B, d\rangle}{\langle \exists X.A, c\rangle \longrightarrow \langle \exists X.(d, B), c \sqcup \exists_X d\rangle}$	
T5	$\dfrac{\langle A, d \sqcup \exists_X c\rangle \longrightarrow \langle B, d'\rangle}{\langle \exists X.(d, A), c\rangle \longrightarrow \langle \exists X.(d', B), c \sqcup \exists_X d'\rangle}$	
T6	$\dfrac{\langle A, c\rangle \longrightarrow \langle A', c'\rangle}{\begin{array}{c}\langle A \parallel B, c\rangle \longrightarrow \langle A' \parallel B, c'\rangle \\ \langle B \parallel A, c\rangle \longrightarrow \langle B \parallel A', c'\rangle\end{array}}$	
T7	$\dfrac{\langle A, c\rangle \longrightarrow \langle A', c'\rangle}{\begin{array}{c}\langle A \oplus B, c\rangle \longrightarrow \langle A', c'\rangle \\ \langle B \oplus A, c\rangle \longrightarrow \langle A, c'\rangle\end{array}}$	

If an execution sequence s is finite of length n and there is no d such that $\langle P_n, c_n\rangle \longrightarrow \langle P', d\rangle$ then s is called *terminating*; otherwise, it is *non-terminating*.

Define an execution sequence s to be *weakly fair* iff no subagent remains enabled continuously from some point on. s is *strongly fair* iff no subagent remains enabled infinitely often. Note that, since a subagent enabled infinitely often implies it must be enabled continuously from some point on, strong and weak fairness coincide. This does not hold in general, but here it is a result of the calculus having a restricted form of non-determinism[8].

The set of all execution sequences of P from c will be denoted $S(P, c)$, and the fair ones $FS(P, c)$.

In the previous section we have identified finite observations with the Scott-compact open subsets $U \in \mathcal{K}\Omega(\mathcal{C}(E))$. Given U Scott compact-open, there must exist a finite subset $F \subseteq \mathcal{K}(C)$ such that $U = \uparrow F$.

Definition 6.1 *A constraint (store) $c \in C$ is said to* satisfy *an observation U (notation $c \models U$) iff $c \in U$.*

Note that this means that $c \sqsupseteq d$ for some $d \in Min(F)$.

Definition 6.2 *For a process P and a constraint $c \in \mathcal{K}(C)$, the* observations of P at *c, denoted $Obs(P)(c)$ are:*

$$Obs(P)(c) \quad = \quad \Big\{U \in \mathcal{K}\Omega(\mathcal{C}(E)) \mid \forall s = \langle P_i, c_i\rangle_{i\in\omega} \in FS(P, c), \exists k.c_k \in U\Big\}$$

A failed computation is one that leads to a store that satisfies *ff*. As $ff \in U$ for any $U \in \mathcal{K}\Omega(\mathcal{C}(E))$ (by $\uparrow$-closure), it follows that an inconsistent store (*i.e.* a store satisfying the constraint *ff*) will satisfy all finite observations. However, note that the observation $\{ff\} \in \mathcal{K}\Omega(\mathcal{C}(E))$ is satisfied by a program P only if all (fair) computations lead to failure; hence, failure can be distinguished from a non-failing computation. Also, failed computations do not matter (in the sense of not being observable) as long as there exist non-failing computations. This explains the nature of *angelic non-determinism* as discussed in (8) – we can think of it as allowing *all* the non-deterministic sub-agents to proceed, and then discarding those which lead to a failure.

Note that, unlike the treatment of (4), we cannot distinguish between deadlocked and non-deadlocked computations.

[8]We do not admit a mechanism for synchronization between *tell*()/*ask*() actions to be refused, but an extension is possible, see *e.g.* (4).

6.3 Denotational Semantics

As the language $\mathcal{L}$ is non-deterministic, we shall require a suitable *powerdomain* construction over the given constraint system $\mathcal{C}(E)$ to allow for situations in which a number of possible results is produced by a non-deterministic process. For example, the process

$$ask(X = 1) \rightarrow tell(Y = 1) \oplus ask(X = 1) \rightarrow tell(Y = 2)$$

when initiated in a store satisfying the constraint $X = 1$, will *either* lead to the store satisfying $X = 1 \wedge Y = 1$ *or* to the store satisfying $X = 1 \wedge Y = 2$. We shall thus require a *set* of constraints to model the result of a non-deterministic process.

One such construction is the *Smyth* (or *upper*) powerdomain, which, as shown in (8), allows to reduce the representation of a process to a *set* of constraints, instead of a subset of a set of sets of constraints. We shall briefly explain the construction here (the reader is referred to (7) for more details). Given a constraint system $\mathcal{C} = \langle C, \sqsubseteq \rangle$, which is an algebraic lattice with a countable set $\mathcal{K}(C)$ of compact elements, and hence also an ω-algebraic cpo, we can define the *Smyth preorder* $\preceq \subseteq \wp_{\mathrm{f}}^{\mathrm{ne}}(\mathcal{K}(C)) \times \wp_{\mathrm{f}}^{\mathrm{ne}}(\mathcal{K}(C))$ on finite sets of compact elements by

$$u \preceq v \iff \forall c \in u, \exists d \in v . c \sqsupseteq d.$$

Note that this order identifies all subsets containing tt. For the remaining subsets, the fewer constraints in the set, the higher it is in the ordering. Now define the *Smyth powerdomain* $(\mathcal{P}_{\mathrm{u}}(\mathcal{C}), \sqsubseteq)$ as the ideal completion of $(\wp_{\mathrm{f}}^{\mathrm{ne}}(\mathcal{K}(C)), \preceq)$, *i.e.*

$$(\mathcal{I}(\wp_{\mathrm{f}}^{\mathrm{ne}}(\mathcal{K}(C))), \subseteq).$$

An alternative representation for the Smyth powerdomain (up to isomorphism) is as the family of all non-empty $\uparrow$-closed Scott-compact subsets of the constraint system $\mathcal{C}$ ordered by superset (17).

It can be shown that, for a constraint system $\mathcal{C} = \langle C, \sqsubseteq \rangle$, the Smyth powerdomain $\mathcal{P}_{\mathrm{u}}(\mathcal{C})$ is also a constraint system (*i.e.* it is an algebraic lattice, possibly satisfying additional properties depending on the underlying constraint system). Moreover, the Smyth powerdomain is closed under set-theoretic union (*i.e.* for any $u, v \in \mathcal{P}_{\mathrm{u}}(\mathcal{C})$, $u \cup v \in \mathcal{P}_{\mathrm{u}}(\mathcal{C})$), and there exists a "singleton" mapping $\{\!|\cdot|\!\} : \mathcal{C} \longrightarrow \mathcal{P}_{\mathrm{u}}(\mathcal{C})$, defined by $\{\!|c|\!\} = \uparrow c$ for $c \in C$. The supremum $\bigsqcup_{i \in \omega} x_i$ for an increasing chain $x_1 \supseteq x_2 \cdots$ is $\bigcap_{i \in \omega} x_i$.

Following (8), we identify deterministic processes with continuous closure operators on $\mathcal{C}$, and non-deterministic ones with linear, continuous closure operators on $\mathcal{P}_{\mathrm{u}}(\mathcal{C})$. Such closure operators can be represented by sets of fixed-points, *i.e.* $\{c \in \mathcal{C} \mid f(c) = c\}$ for $f : \mathcal{C} \longrightarrow \mathcal{C}$ a closure operator. For linear closure operators $f : \mathcal{P}_{\mathrm{u}}(\mathcal{C}) \longrightarrow \mathcal{P}_{\mathrm{u}}(\mathcal{C})$ on the Smyth powerdomain, we can use the set of singleton fixed-points, *i.e.* elements $c \in \mathcal{C}$ such that $f(\{\!|c|\!\}) = c$.

Theorem 6.3 (8) *Let $\mathcal{C}(E)$ be a constraint system. $S \subseteq \mathcal{C}(E)$ is the set of singleton fixed-points of a linear closure operator $f : \mathcal{P}_{\mathrm{u}}(\mathcal{C}(E)) \longrightarrow \mathcal{P}_{\mathrm{u}}(\mathcal{C}(E))$ iff it is Scott-compact. f is the linear extension of $\lambda x.(\uparrow x \cap S)$. Moreover, f is continuous iff for all chains $(c_i)_{i\in\omega}$, $S \cap \uparrow(\bigsqcup_{i\in\omega} c_i) = \bigcap_{i\in\omega}(S \cap \uparrow c_i)$.*

Given the set S_f of singleton fixed-points, the linear closure operator can be recovered through the equation $f(x) = \uparrow(x \cap S_f)$. *From now on, we shall identify linear closure operators on the Smyth powerdomain of $\mathcal{C}(E)$ with the Scott-compact subsets of $\mathcal{C}(E)$.*

The following are semantic equations, compositional in style, defining inductively the sets of singleton fixed-points of a process. The process $tell(c)$ is a fuction of x which returns $\{\!|c|\!\}\sqcup x$. The process $ask(c) \to A$ behaves like a function of x, which tests if x is above $\{\!|c|\!\}$ returning the result of A on c if that is the case, and $\{\!|x|\!\}$ otherwise. The parallel and choice operators correspond to intersection and union of the sets of fixed-points respectively. The hiding operator $\exists X.A$ is a kernel operator (behaves like a projection); the result of $\exists X.A$ on c should be $c\sqcup\exists_X(A(\exists_X c))$ (here $c\sqcup X$ means $\{c\sqcup x \mid x \in X\}$).

$$
\begin{array}{rcl}
[\![tell(c)]\!] & = & \uparrow c \\
[\![ask(c) \to A]\!] & = & \overline{\uparrow c} \cup [\![A]\!] \\
[\![A \parallel B]\!] & = & [\![A]\!] \cap [\![B]\!] \\
[\![A \oplus B]\!] & = & [\![A]\!] \cup [\![B]\!] \\
[\![\exists X.A]\!] & = & \exists_X^{-1}(\exists_X([\![A]\!])) \\
[\![p(Y)]\!] & = & \exists_Y(\uparrow d_{XY} \cap [\![A]\!]) \quad \text{if } p(Y)\text{: } -A \in W
\end{array}
$$

Proposition 6.4

1. *If $\exists_X$ preserves Scott-compactness under inverse image then the set of fixed-points of a possibly infinite process is Scott compact.*

2. *If $\exists_X$ preserves Scott-compact opens then the set of fixed-points of a finite process is Lawson clopen.*

3. *If $\exists_X$ preserves Scott-compact opens then the set of fixed-points of a possibly infinite process is Lawson closed.*

Proof: By induction. Observe that $\uparrow c, \overline{\uparrow c}$ are Scott open and closed respectively, and hence Lawson subbasic open and Lawson clopen. Continuous functions preserve Scott compactness; also, Scott open and closed sets are preserved under inverse image. The open closed and clopen sets are closed under finite union and intersection. An infinite process is a result of recursive procedure calls; through unwindings these can be reduced inductively to countable intersections of finite processes of the form:

$$
\begin{array}{rcl}
[\![p^0(Y)]\!] & = & \uparrow tt \\
[\![p^{k+1}(Y)]\!] & = & \exists_Y(\uparrow d_{XY} \cap \exists_Y(\uparrow d_{XY} \cap [\![p^k(Y)]\!])).
\end{array}
$$

In an algebraic lattice, countable intersections of Scott compact sets (Lawson closed) are compact (Lawson closed) respectively. $\square$

It is interesting that the interpretation of *ask*() gives rise to a subset which is neither Scott-open nor Scott-closed (but it is Lawson clopen). Since the clopen subsets form a Boolean algebra, this would mean that we are extracting classical logic from the constraint system. It would be interesting to find out if a suitable (intuitionistic) interpretation could be found in the Scott topology.

6.4 Observables, Properties and Satisfacton

Given a process $P \in \mathcal{L}$ define the *observables* function $\mathcal{O} : \mathcal{L} \times \mathcal{K}(C) \to \mathcal{P}(C)$ as follows.

Definition 6.5 *For a process P and a constraint $c \in \mathcal{K}(C)$, the* observables of P at c, *denoted $\mathcal{O}(P)(c)$, is the linear continuous extension of:*

$$\mathcal{O}(P)(c) \quad = \quad \bigcap \{U \mid U \in Obs(P)(c)\}$$

to $\mathcal{P}_{\mathrm{u}}(\mathcal{C}(E))$.

Note that $\mathcal{O}(P)$ is monotone, *i.e.* $c \sqsubseteq d$ implies $\mathcal{O}(P)(c) \subseteq \mathcal{O}(P)(d)$, so the continuous extension exists.

A process P initiated in the store c *satisfies* the observation U (notation $P \models U$) iff $\mathcal{O}(P)(c) \subseteq U$. Given an arbitrary property $\Psi \subseteq C$, we say P satisfies Ψ iff

$$\forall c \in \mathcal{K}(C). \mathcal{O}(P)(c) \subseteq \Psi$$

This means that $\bigcup_{c \in \mathcal{K}(C)} \mathcal{O}(P)(c)$ (disjunction on all possible inputs) must be included in Ψ.

It can be shown that operational semantics (the observables) and the denotational semantics agree (as functions).

Theorem 6.6 (8) $\mathcal{O}(P) = [\![P]\!]$.

7 SEMANTIC PROPERTIES

Given a language $\mathcal{L}$ interpreted over a constraint system $\langle C, \sqsubseteq \rangle$, processes written in this language "compute" subsets of C (the *observables*). This is formally stated in terms of a function $\mathcal{O} : \mathcal{L} \times \mathcal{K}(C) \to \mathcal{P}_{\mathrm{u}}(C)$. Recall that each (well-formed) process expression determines some subset $\mathcal{O}(P)(c)$ of C. In Section 6.3 we shall give a more specific definition of the observables function, and also characterize those subsets of C which correspond to the process expressions.

The aim of this section is to describe the subsets of C which correspond to *safety*, *liveness* and *fairness* properties in the intuitive sense explained in Section 2.

7.1 Safety Properties

Intuitively, a safety property states that nothing bad must happen. The "thing" must be observable, and hence there must exist a corresponding finite observation $U \subseteq C$. It must also be "bad", that is, the process is prohibited from ever entering the set U, or, in other words, must remain entirely within the set-theoretic complement of the open set U.

Definition 7.1 *$S \subseteq C$ is a* safety property *iff S is a Scott closed set.*

Note that this means that S is a safety property iff there exists a (countable) family of finite observations $\{U_i \subseteq \mathcal{K}\Omega(C) \mid i \in \omega\}$ such that for all $c \in C$, $c \in S$ iff $\forall i \in \omega.\ c \not\models U_i$. The countability restriction is sufficient because we have an ω-algebraic lattice.

The following statement says: (1) a safety property contains those, and only those, constraints which do not entail the "bad" thing; (2) if ϕ is safe, then so are all formulae entailed by it; (3) "once lost, a safety property is never regained"; and (4) safety is inaccessible through directed sups.

Proposition 7.2 *Let S be a safety property.*

1. *$\phi \in S$ iff $\phi \not\vdash \psi$ some $\psi \in C$*
2. *If $\phi \in S$ and $\phi \vdash \psi$ then $\psi \in S$.*
3. *If $\phi \notin S$ then $\forall \psi \in C.\psi \vdash \phi$ we have $\psi \notin S$.*
4. *If $D \subseteq S$ then $\bigsqcup D \in S$.*

Proof: Direct from definitions. $\Box$

Example 7.3 *In Example 4.4, $c_1 = t \wedge c_2 = t$ is "bad", so $A_M^{\wedge} \setminus \{\uparrow(c_1 = t \wedge c_2 = t)\}$ is the corresponding safety property.*

Proposition 7.4 *The class of all safety properties over the constraint system $\mathcal{C}(E)$ is closed under finite unions and countable intersections.*

The following statement is an often repeated intuitive property of safety, which allows to deduce that a safety property holds for some system if it can be proved (inductively) that it holds for every finite computation.

Proposition 7.5 *Let S be a safety property in the constraint system $\mathcal{C}(E)$, P a closed term in $\mathcal{L}$. Then P satisfies S iff $\mathcal{O}(P)(c) \cap \mathcal{K}(C) \subseteq S$ for all $c \in \mathcal{K}(C)$.*

Proof:

1. ($\Rightarrow$) P satisfies S iff $\forall c \in \mathcal{K}(C).\mathcal{O}(P)(c) \subseteq S$, which certainly implies $\mathcal{O}(P)(c) \cap \mathcal{K}(C) \subseteq S$.
2. ($\Leftarrow$) Suppose for all $c \in \mathcal{K}(C)$, $\mathcal{O}(P)(c) \cap \mathcal{K}(C) \subseteq S$ then $Cl(\mathcal{O}(P)(c) \cap \mathcal{K}(C)) \subseteq Cl(S) = S$ where $Cl(\cdot)$ denotes Scott closure, and so $\mathcal{O}(P)(c) \cap \mathcal{K}(C) \subseteq S$ (because Scott closure is prefix closure and closure under directed sups).

$\square$

7.2 Liveness Properties

On the other hand, a liveness property requires that something "good" happens. It is often the case that countable families of such good things are necessary, just consider the property "infinitely often a". Each "good" thing ϕ must be observable, and therefore there must exist a family $\{U_i \mid i \in \mathbf{N}\}$ of observations such that $\phi \models U_i$ for all i. This motivates the following:

Definition 7.6 *$L \subseteq C$ is a* finitary liveness *property iff L is a Scott open set. $L \subseteq C$ is an* infinitary liveness *property iff L is a Scott G_δ.*

Unlike safety, liveness properties may be infinitary. Note that for every finitary liveness property there exists a (countable) family of finite observations $\{U_i \subseteq \mathcal{K}\Omega(C) \mid i \in \omega\}$ such that for all $c \in C$, $c \in L$ iff $\exists i \in \omega \ . \ c \models U_i$. On the other hand, for an infinitary property there exists a countable family of finitary liveness properties $\{L_i \mid i \in \omega\}$ such that for all $c \in C$, $c \in L$ iff $\forall i \in \omega \ . \ c \models L_i$.

Finitary liveness can be useful to model properties such as $odd(n)$ where $n \in \omega$ in the flat domain (we either observe $\uparrow 1$, or $\uparrow 3$, etc).

Example 7.7 *In Example 4.17, for a process to enter its critical section is a "good" thing, and the set* $\uparrow(e_1 \geq 1)$ *is therefore a (finitary) liveness property stating "process no 1 entered its critical section at least once". An infinitary liveness property ("process no 1 has entered its critical section infinitely often") is* $\bigcap_{i\in\omega} \uparrow(e_1 \geq i)$.

The following statement says: (1) if ϕ is "good', then so are all formulae which entail it ("once gained, never lost"); (2) every finitary liveness property must be determined at finite time; and (3) safety is a notion dual to finitary liveness.

Proposition 7.8 *Let* L *be a liveness property.*

1. *If* $\phi \in L$ *and* $\psi \vdash \phi$ *then* $\psi \in L$.
2. *If* L *is finitary then* $\forall \psi \in L$, $\exists \phi \in \mathcal{K}(C)$ *such that* $\psi \vdash \phi$;
3. *Every safety property is a complement of a finitary liveness property, and vice-versa.*

Proof: Direct from definitions. $\square$

Proposition 7.9 *The class of all finitary liveness properties over the constraint system* $\mathcal{C}(E)$ *is closed under countable unions and finite intersections. The class of all liveness properties is closed under finite intersections and unions.*

7.3 Fairness Properties

Fairness properties are viewed either as a subclass of liveness. We define a fairness property to be a subset of the maximal elements of $\mathcal{C}(E) \setminus \{\mathit{ff}\}$ (maximal after ff has been removed). However, not every such subset corresponds to fairness in the intuitive sense – only those subsets whose $\downarrow$-closure includes *all* the compact elements $\mathcal{K}(C)$ are. This is to guarantee that: (1) every terminating computation is fair; (2) every finite computation has some fair computation above it.

Definition 7.10 $F \subseteq \mathit{Max}(C \setminus \{\mathit{ff}\})$ *is a* fairness *property iff* F *is a Scott* G_δ *set dense in* $C \setminus \{\mathit{ff}\}$.

Example 7.11 *In Example 4.6, the set of lists which contain a finite number of* b *and an infinite number of* a *is not a fairness property (because the list* b^{n+1} *does not have a fair computation above), while the set of lists containing an infinite number of both* a *and* b *is a fairness property.*

We have the following statement directly from definition.

Proposition 7.12 *Let F be a fairness property. Then for all $\psi \in \mathcal{K}(C)$ there exists $\phi \in F$ such that $\phi \vdash \psi$.*

Proof: By maximality and density. $\Box$

8 CONCLUSION AND FURTHER WORK

We have set out a basic framework for an axiomatic presentation of constraint systems, derived their semantic model, interpreted a process language in it, and characterized certain properties semantically.

Note that not all semantic properties can be expressed syntactically – only a subclass of finitary liveness corresponding to the Scott-compact opens is. The restriction to the compact-opens is a consequence of Stone duality, and is rather severe. We should point out, however, that the compact-opens are simply "building blocks", from which the more general properties can be built. We have shown how these "building blocks" can be composed semantically; what seems to be needed is a suitable syntax. As already suggested in (2), we would need least and greatest fixpoints as modalities of formulae, interpreted as $\bigsqcup_{i\in\omega}\phi_i$ and $\prod_{i\in\omega}\phi_i$. We are currently investigating a version of Stone duality for a formal system allowing fixpoint operators. The reason we have not considered it in this paper is that finite axiomatization is no longer sufficient. We would require an enhancement of the theory with suitable modalities and an axiomatization in terms of schemas. Finally, an investigation of the correspondence between the constructed concurrent constraint system and the extended elementary constraint system would have to be carried out.

It should be mentioned that we have considered a distributive lattice structure, partly because distributivity is essential to Stone duality. One can work with semilattices, which are more common in concurrent constraint programming, by suitably extending them. We have not included implication – complete Heyting algebras admit implication, but unfortunately implication does not preserve compactness, and it is not preserved by morphisms. A further point worth investigating is the relationship between implication and guards.

9 ACKNOWLEDGEMENTS

The author would like to acknowledge the AP1 group at CWI, especially Jaco de Bakker, Jan Rutten, Katuscia Palamidessi, Frank de Boer and Jeroen Warmerdam for a stimulating environment. Paul Taylor's diagrams package was used to produce the figures in this paper.

References

[1] S. Abramsky. Domain theory in logical form. In *Proceedings, Annual Symposium on Logic in Computer Science*, pages 47–53. IEEE CS, 1987.

[2] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[3] B. Davey and H. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[4] F. de Boer and C. Palamidessi. A fully abstract model for concurrent constraint programming. In S. Abramsky and T. Maibaum, editors, *Proc. of TAP-SOFT/CAAP*, volume 493 of *Lecture Notes in Computer Science*, pages 296–319. Springer-Verlag, 1991.

[5] N. Francez. *Fairness.* Springer-Verlag, 1986.

[6] G. Gierz, K. Hofmann, K. Keimel, J. Lawson, M. Mislove, and D. Scott. *A Compendium of Continuous Lattices.* Springer-Verlag, 1980.

[7] C. Gunter and D. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science.* Elsevier Science Publishers, 1990.

[8] R. Jagadeesan, V. A. Saraswat, and V. Shanbogue. Angelic non-determinism in concurrent constraint programming. Technical report, System Sciences Laboratory, Xerox PARC, 1991.

[9] P. T. Johnstone. *Stone Spaces.* Cambridge University Press, 1982.

[10] M. Z. Kwiatkowska. Event fairness and non-interleaving concurrency. *Formal Aspects of Computing*, 1(3):213–228, 1989.

[11] M. Z. Kwiatkowska. Survey of fairness notions. *Information and Software Technology*, 31(7):371–386, 1989.

[12] M. Z. Kwiatkowska. Defining process fairness for non-interleaving concurrency. In K. Nori and Veni-Madhavan, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 472 of *Lecture Notes in Computer Science*, pages 286–300. Springer-Verlag, 1990.

[13] M. Z. Kwiatkowska. On topological characterization of behavioural properties. In G. Reed, A. Roscoe, and R. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 153–177. Oxford University Press, 1991.

[14] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977.

[15] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proceedings, 9th ACM Symposium on Principles of Distributed Computing*, pages 377–408. ACM Press, 1990.

[16] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[17] G. D. Plotkin. Post-graduate lecture notes in advanced domain theory. Incorporating the "Pisa Notes", 1981.

[18] V. Saraswat and M. Rinard. Concurrent constraint programming. In *Proc. of the seventeenth ACM Symposium on Principles of Programming Languages*, pages 232–245. ACM, New York, 1990.

[19] V. Saraswat, M. Rinard, and P. Panangaden. A fully abstract semantics for concurrent constraint programming. In *Proc. of the eighteenth ACM Symposium on Principles of Programming Languages*. ACM, New York, 1991.

[20] H. Simmons. Frames: the point-free approach to topology. Lecture notes for a Logic for IT course, 1989.

[21] M. B. Smyth. Powerdomains and predicate transformers: A topological view. In *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*. Springer, 1983.

[22] S. Vickers. *Topology via Logic*, volume 5 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.