A Logical View of Composition and Refinement

Martín Abadi[*] Gordon D. Plotkin[†]

Abstract

We define two logics of safety specifications for reactive systems. The logics provide a setting for the study of composition and refinement rules, and a framework for the use of the modular specification methods that these rules underpin. The two logics arise naturally from extant specification approaches; one of the logics is intuitionistic, while the other one is linear.

1 Introduction

Modular, hierarchical methods for specifying reactive systems [HP85] include rules for composing and refining specifications (e.g., [dBdRR90]). The form of the rules suggests a possible specification logic. In it, the propositions would be system specifications; the notations for combining specifications would become logical connectives; and the rules for composition and refinement would be formulated as sound inference rules. The logic would thereby provide a setting for the study of composition and refinement rules. It should also provide a framework for developing specifications and for verifying them using these rules.

In this paper, we define and develop such a logic. In fact two logics arise naturally. One of the logics is an intuitionistic logic, while the other one is linear [Gir87]. In the intuitionistic logic, a specification is a set of allowed behaviors, as in [Lam83a, BKP84]. In the linear logic, a specification is a set of allowed processes, much as in the sense of Abrahamson [Abr79].

Composition rules and refinement rules typically apply to safety properties, and also, sometimes with significant complication, to certain liveness properties. Here we treat only safety properties. With this restriction, the logics provide a new understanding of some current specification methods, and suggest extensions. They are intended as a basis for Lamport's transition-axiom method for reactive systems [Lam89].

A reactive system can be expected to operate correctly only when its environment operates correctly. For example, a concurrent program module can be expected to exhibit desirable behavior only when its inputs are of the proper types. The environment cannot be required to operate correctly, but the system's obligations are void when the environment operates incorrectly. An assumption-guarantee specification states that a reactive system satisfies a specification M if it operates in an environment that satisfies an assumption E; this specification is sometimes written $E \Rightarrow M$.

A Composition Principle gives a way of combining assumption-guarantee specifications while discharging their assumptions [MC81, Pnu84, Sta85, AL90]. A simple version of the Principle, applied to two reactive systems p_1 and p_2, says:

> If p_1 satisfies $M_2 \Rightarrow M_1$
> and p_2 satisfies $M_1 \Rightarrow M_2$,
> then when they are run in parallel
> p_1 satisfies M_1 and p_2 satisfies M_2.

As stated, the Composition Principle is not sound in general. The underlying propositional reasoning is obviously (and intriguingly) circular.

However, the Principle is sound when M_1 and M_2 are safety properties, and under some additional hypotheses. For instance, consider two processes p_1 and

[*]Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301, USA.

[†]Dept. of Computer Science, University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, UK. This work was completed while at Digital Equipment Corporation.

0

p_2 that communicate by the distributed, integer variables x_1 and x_2; it is assumed that only p_1 writes x_1 and that only p_2 writes x_2. Let M_1 be "x_1 never decreases" and M_2 be the corresponding assertion for x_2, and suppose that p_1 and p_2 satisfy $M_2 \Rightarrow M_1$ and $M_1 \Rightarrow M_2$, respectively. Then it is sound to conclude that M_1 and M_2 both hold, that is, that neither x_1 nor x_2 ever decreases.

An important test for a logic of specifications is whether it can be used to express and to illuminate the Composition Principle. Both of our logics are designed to satisfy this criterion. For example, the intuitionistic formulation of the Principle just given is:

$$(M_2 \to M_1) \wedge (M_1 \to M_2) \vdash M_1 \wedge M_2$$

with a proviso to guarantee that M_1 and M_2 are specifications of separate processes. The logics can express also other variants of the Composition Principle; they serve in comparing these variants and, occasionally, in discovering new ones.

Internal, hidden variables make the writing of specifications easier. In the specification of a buffer, for instance, it may be convenient to mention the sequence of elements that the buffer contains, which is not directly observable. Unfortunately, hidden variables can complicate the proof that one specification implements another, weaker specification. Various refinement rules have been proposed for these proofs; they are all based on instantiating internal variables. Logically, hiding is represented by existential quantification (e.g., [Lam89]). In order to prove that $\exists y.N$ implements $\exists x.M$, one gives an instantiation $g(y)$ for the variable x, and then proves that N implements M under this instantiation. The function g is known as a refinement mapping. For specifications of special forms, the basic refinement mappings have been complemented by the use of history and prophecy variables [AL88], and they have been generalized to multivalued mappings [LT87, Mer90].

In our logics, hiding is exactly existential quantification, and a simple refinement-mapping rule follows easily from the usual logical laws for quantifiers. It reads:

$$\frac{N \vdash M[g(y)/x]}{\exists y.N \vdash \exists x.M}$$

As we consider only safety properties, which are closed sets, we obtain an intuitionistic logic. In this we follow Hennessy and Plotkin [HP89] and, less directly, Abramsky with his proposal of a general logic of open sets [Abr89]. Parallel composition can be represented by conjunction, as in works of Lamport and Pnueli. Dam [Dam88] and Abramsky [Vic88] pointed out that in general parallelism will give extra, quantalic structure. This indeed happens when we take specifications to be sets of processes, and then the logic of specifications is linear. Our work may yield some evidence for the relevance of linear logic to concurrency. Other evidence can be found in work on Petri Nets (e.g., [MOM89]) and testing equivalence [AV90].

We introduce our logics in the following overview. Sections 3, 4, and 5 give formal definitions and develop the material further. The usual intuitionistic and linear logics need to be enriched in order to support the assumption-guarantee specification style. And they need to be enriched again, for example with prefixing and fixpoint operators, in order to obtain a basis for a complete specification language. The specification formalism is considered in section 6. Section 7 compares the logics.

2 Overview

We review the basic propositional intuitionistic and linear calculi. We describe the usual connectives, and motivate the addition of new constructs. The logic fragments covered are those needed for the study of composition rules.

2.1 A calculus of sets of behaviors

The intuitionistic logic is inspired by the work of Lamport, Pnueli, and others, where the specification of a system is a set of allowed behaviors. In turn, a behavior is a sequence of states, and a state is an assignment of values to state components, or variables. Each state transition is attributed to an agent, the environment process or system process that caused the state change. Thus, a behavior is a sequence

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots$$

where each s_i is a state and each a_i is an agent, and the sequence is either infinite or else ends in a state s_m for some $m \geq 0$.

The use of agents is motivated by the obvious need to distinguish between actions performed by the environment and those performed by the system. In any particular specification, it suffices to consider two agents: the environment and the system. However, it is preferable to allow arbitrary sets of agents, in order to ease the composition of specifications.

Since we are concerned only with safety properties, we restrict attention to finite behaviors. A safety property is then a prefix-closed set of behaviors. In the logic, the propositions denote safety properties, and $\vdash$ simply stands for $\subseteq$. The collection of safety properties forms a complete Heyting algebra [Joh82] and so the intuitionistic logical operations $\wedge$, $\vee$, and $\rightarrow$ are available. The first two are intersection and union.

Conjunction serves its usual logical role: a process p satisfies $M \wedge M'$ if and only if it satisfies both M and M'. Further, conjunction represents parallel composition: if p_1 satisfies M_1 and p_2 satisfies M_2 then p_1 and p_2 in parallel satisfy $M_1 \wedge M_2$. Disjunction corresponds to nondeterministic choice: if p_1 satisfies M_1 and p_2 satisfies M_2 then a processes that acts like either p_1 or p_2 satisfies $M_1 \vee M_2$.

Implication turns out to be a familiar and handy operation: $E \rightarrow M$ is the set of all behaviors that satisfy M at least as long as they satisfy E. The connective $\rightarrow$ has arisen in works on the Composition Principle (in [AL90], and implicitly in [MC81] and [Pnu84]). Under reasonable hypotheses, the specifications $E \Rightarrow M$ and $E \rightarrow M$ have the same implementations, and hence $\Rightarrow$ can be replaced with $\rightarrow$. It is encouraging that the logical formulation naturally yields this connective.

The specification of a system cannot require the environment to work properly, and so any environment action should be allowed. More precisely, if a property M is intended to specify the process represented by an agent (or set of agents) μ, then any behavior not in M should end with a μ state change. When this condition holds, we say that M constrains at most μ, and write $M \triangleleft \mu$.

With this notation, the Composition Principle reads: for any M_1 and M_2,

$$(M_2 \rightarrow M_1) \wedge (M_1 \rightarrow M_2) \vdash M_1 \wedge M_2$$

provided $M_1 \triangleleft \mu_1$, $M_2 \triangleleft \mu_2$, and the sets μ_1 and μ_2 are disjoint

The proviso expresses the requirement that M_1 and M_2 describe different processes. (The Principle is not sound otherwise, for example if M_1 and M_2 are the same.) Note how the logical approach obviates the need for explicit reference either to processes (as in [MC81, Pnu84]) or to the realizable parts of properties (as in [AL90]).

Many variants of the Composition Principle can be treated in this framework; for example, we easily obtain:

$$\frac{E \wedge M_2 \vdash E_1 \qquad E \wedge M_1 \vdash E_2}{(E_1 \rightarrow M_1) \wedge (E_2 \rightarrow M_2) \vdash (E \rightarrow M_1 \wedge M_2)}$$

where $M_1 \triangleleft \mu_1$, $M_2 \triangleleft \mu_2$, and μ_1 and μ_2 are disjoint. Some of these variants are well known, while others seem to be new. All of them can be proved equivalent using propositional reasoning and a few rules about the constrains relation.

2.2 A calculus of sets of processes

In the linear calculus, a proposition denotes a set of processes. We take a process to be a set of sequences of state pairs. Intuitively, a process that contains $(s_1, t_1)\,(s_2, t_2)\,(s_3, t_3)\,\ldots$ can change the state from s_1 to t_1, and later from s_2 to t_2, and later yet from s_3 to t_3, ...

In the study of safety, it suffices to consider finite sequences of state pairs. We require also that processes be prefix-closed. It turns out that the set of safety properties is isomorphic to the set of processes; thus, we may identify safety properties and processes.

The logical operations $\wedge$, $\vee$, and $\rightarrow$ are still meaningful. They arise as before from the complete Heyting algebra structure of the partial order of safety properties.

The property $M \wedge M'$ allows the processes that are allowed both by M and M'; conjunction does not have any particular relation with concurrency. Disjunction corresponds to nondeterministic choice, as before. Finally, $M \rightarrow M'$ includes the processes that behave like a process in M' at least as long as they behave like a process in M.

Intuitionistic linear logic arises when we consider the parallel composition of two processes. The parallel composition of p_1 and p_2 is the set of shuffles of p_1 sequences with p_2 sequences. At the level of specifications, this gives rise to a new logical operation, $\otimes$, which is the multiplicative conjunction in linear logic. A process satisfies $M_1 \otimes M_2$ if it is the parallel composition of an M_1 process with an M_2 process. Thus, if p_1 satisfies M_1 and p_2 satisfies M_2 then the parallel composition of p_1 and p_2 satisfies $M_1 \otimes M_2$.

Associated with the connective $\otimes$ is a linear implication operation, $\multimap$. The property $M_1 \multimap M_2$ is the largest N such that $M_1 \otimes N$ is a subset of M_2. Thus, $p \in M_1 \multimap M_2$ if and only if the parallel composition of p with any $q \in M_1$ satisfies M_2.

Conjunction and disjunction are then the additive connectives of linear logic. The exponential ! is trivial, but instead a $(\ )^*$ construct can be added to represent the parallel composition of a number of like processes. In the next subsection, we propose an interpretation of the classical constructs.

The standard intuitionistic linear connectives do not suffice as a basis for assumption-guarantee specifications. In particular, $p \in E \multimap M$ is not equivalent to the desired "p satisfies M in any environment that satisfies E." The assertion $p \in E \multimap M$ means only that the composition of p with any E process q is an M process. It is possible that q is not the whole environment of p—there could be a third process running in parallel; it is also possible that p does not satisfy

M in this environment—the parallel composition of p and q does.

To remedy this deficiency, we introduce a connective $\multimap$. The property $M_1 \multimap M_2$ consists of the processes that, when run in parallel with an M_1 process (and with nothing else), behave like M_2 processes. The special case of $M_1 \multimap M_2$ where M_1 contains only the null process 1 is of particular interest; $\{1\} \multimap M$ is the set of all processes that behave like a process in M when run by themselves, with no interference from the environment. We denote this property by $M^\diamond$.

Now the Composition Principle goes:

$$(M_2 \multimap M_1) \otimes (M_1 \multimap M_2) \vdash (M_1 \otimes M_2)^\diamond$$

This formula is valid in our model, without any additional proviso. As in the intuitionistic case, a number of variants of the Composition Principle are available, and for example we have also the more general:

$$\frac{E \otimes M_2 \vdash E_1 \qquad E \otimes M_1 \vdash E_2}{(E_1 \multimap M_1) \otimes (E_2 \multimap M_2) \vdash (E \multimap M_1 \otimes M_2)}$$

2.3 Testing

The linear logic described so far is an intuitionistic one. It does not include a constant $\perp$ that resembles falsehood, or a negation-like involution $(\ )^\perp$. The notion of testing suggests useful $\perp$ and $(\ )^\perp$ constructs, and gives rise to a different account of assumption-guarantee specifications.

We can view the environment of a process as a tester for the process. We distinguish a state *fail* to represent the result of failed tests. A process p passes the test of q if q and p do not yield the state *fail* when they run in parallel, and q fails p otherwise. A process succeeds if it does not yield *fail* when it runs in isolation. Thus, p passes the test of q if the parallel composition of p and q succeeds.

Success is a safety property, and we write $\perp$ for the set of all processes that succeed. A sort of negation can also be defined: $M^\perp$ is the set of all processes that do not fail M processes. Naturally, we are particularly interested in the propositions M such that $M = (M^\perp)^\perp$, which are called facts. These are the specifications that have sound and complete testers; they can be characterized explicitly with a simple set of closure conditions.

Certain expressions in this classical linear logic are reminiscent of assumption-guarantee specifications. In particular, $(E \wedge M^\perp)^\perp$ is the set of processes that pass all of the tests that M processes pass, provided these tests are from E. In other words, $(E \wedge M^\perp)^\perp$ includes all of the processes that cannot be distinguished from M processes in E environments (by E tests). It is analogous to the assumption-guarantee specification $E \Rightarrow M$, but the obvious analogues of the Composition Principle do not hold.

A small correction solves this problem. Let

$$E^+ = E \cup \{u(s, \mathit{fail}) \mid u \in E,\ s \text{ a state}\}$$

The processes in E^+ behave like processes in E, except that they may fail the testee at any point. If E and M are facts, then

$$E \multimap M = (E^+ \cap M^\perp)^\perp$$

and the expected Composition Principle follows.

3 Intuitionistic Logic

The model that underlies the intuitionistic logic is a small variant of that in [AL90]; we refer the reader to this and previous works for additional motivation.

We assume given a nonempty set of states, $\mathbf{S}$, and a nonempty set of agents, $\mathbf{A}$. These sets are disjoint. A *behavior* is a finite sequence

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots s_{n-1} \xrightarrow{a_n} s_n$$

where each s_i is a state and each a_i is an agent. If s is a state then $\langle s \rangle$ denotes the sequence with s as its sole element. If σ is a sequence, a an agent, and s a state, the $\sigma \cdot (a, s)$ denotes the concatenation of a and s at the end of σ. The set of all behaviors is denoted by $\mathcal{B}$.

A *safety property* is a set of behaviors closed under prefixes. The set of all safety properties is denoted by $\mathcal{S}_b$. Safety properties, as we have defined them, are isomorphic to the safety properties of [AL90], for example, with the caveat that we have not yet treated invariance under stuttering. It is quite natural, and desirable, to add a straightforward condition of invariance under stuttering to our definitions, as first advocated by Lamport [Lam83b]. For simplicity, we do not do so at this point.

The *length* $|\sigma|$ of a behavior σ is the number of agents that occur in σ. If $0 \leq m \leq |\sigma|$ then $\sigma|_m$ is the prefix of σ of length m; if $m > |\sigma|$, then $\sigma|_m = \sigma$.

Proposition 1 *$\mathcal{S}_b$ is a complete Heyting algebra, where $\wedge$ is $\cap$, $\vee$ is $\cup$, and the associated $\rightarrow$ is*

$$M \rightarrow M' = \{\sigma \mid \forall n \geq 0.\ \text{if } \sigma|_n \in M \text{ then } \sigma|_n \in M'\}$$

Hence, the algebra of safety properties is a model for intuitionistic logic. The next two subsections discuss composition and refinement in this intuitionistic setting.

3.1 Composition

We say that the safety property M *constrains at most* the set of agents μ, and write $M \lhd \mu$, if: (i) if $s \in \mathbf{S}$ then $\langle s \rangle \in M$; and (ii) if $\sigma \in M$, $s \in \mathbf{S}$, and $a \in \bar{\mu}$, then $\sigma \cdot (a, s) \in M$. Note that if $M \lhd \mu$ then $(N \rightarrow M) \lhd \mu$ for every N, and that if $\mu \subseteq \nu$ and $M \lhd \mu$ then $M \lhd \nu$.

Further, let M_μ be the smallest superset of M that constrains at most μ. The definition of "constrains at most," in the form of a monotone closure condition, guarantees that such an M_μ exists. In fact, a behavior in M_μ is either a behavior in M extended with arbitrary $\bar{\mu}$ steps, or simply a behavior that consists exclusively of $\bar{\mu}$ steps.

We now have enough machinery for formulating the Composition Principle. As an example, we briefly study two variants. Many other variants are possible. In particular, some of them take initial conditions into account; initial conditions are useful, but pose no major problem in this calculus.

The most basic law that we call a Composition Principle applies to a single specification:

$$(M_{\bar{\mu}} \rightarrow M) \vdash M \tag{1}$$

where $M \lhd \mu$. The soundness of this law can be proved by induction on the length of behaviors. Clearly, if $s \in \mathbf{S}$ then $\langle s \rangle \in M$, because $M \lhd \mu$. As an induction hypothesis, assume that if $\sigma \in (M_{\bar{\mu}} \rightarrow M)$ then $\sigma \in M$, and suppose that $\sigma \cdot (a, s) \in (M_{\bar{\mu}} \rightarrow M)$, in order to prove that $\sigma \cdot (a, s) \in M$. Since M is a safety property, $\sigma \in (M_{\bar{\mu}} \rightarrow M)$, and the induction hypothesis yields $\sigma \in M$. In the case where $a \in \mu$, it follows that $\sigma \cdot (a, s) \in M_{\bar{\mu}}$, and then $\sigma \cdot (a, s) \in (M_{\bar{\mu}} \rightarrow M)$ implies $\sigma \cdot (a, s) \in M$. In the case where $a \in \bar{\mu}$, we obtain $\sigma \cdot (a, s) \in M$ from $M \lhd \mu$.

An apparently more general law applies to two specifications:

$$(M_2 \rightarrow M_1) \wedge (M_1 \rightarrow M_2) \vdash M_1 \wedge M_2 \tag{2}$$

where $M_1 \lhd \mu_1$, $M_2 \lhd \mu_2$, and the sets μ_1 and μ_2 are disjoint. This yields (1) as a special case, by letting $M_1 = M$, $\mu_1 = \mu$, $M_2 = M_{\bar{\mu}}$, $\mu_2 = \bar{\mu}$, and observing that $M_1 \vdash M_2$.

Somewhat surprisingly, the converse holds as well: (2) can be derived from (1). Suppose that $M_1 \lhd \mu_1$ and $M_2 \lhd \mu_2$, with μ_1 and μ_2 disjoint, and assume that $(M_2 \rightarrow M_1) \wedge (M_1 \rightarrow M_2)$. Let $N = (M_1 \rightarrow M_2) \rightarrow M_2$. Since $M_2 \lhd \mu_2$ and $\mu_2 \subseteq \bar{\mu_1}$, $N \lhd \bar{\mu_1}$. Moreover, $M_1 \vdash N$, by propositional reasoning, and hence $(M_1)_{\bar{\mu_1}} \vdash N$. Therefore, $N \rightarrow M_1 \vdash M_1$ follows by (1), and then M_1 follows from the assumption. The proof of M_2 is exactly analogous. Therefore, $(M_2 \rightarrow M_1) \wedge (M_1 \rightarrow M_2) \vdash M_1 \wedge M_2$.

This logical style of proof is pleasant, we found. It has helped us in deriving all of our variants of the Composition Principle for safety properties from (1), which is the simplest.

3.2 Refinement

To discuss refinement, we need to particularize the choice of the set of states. Let Var be a countably infinite set of variables, let $\mathbf{V}$ be a set of values, and let $\mathbf{S}$ be (Var $\rightarrow \mathbf{V}$), the functions from Var to $\mathbf{V}$. For two states s and s', we write $s =_x s'$ if $s(y) = s'(y)$ for all y different from x. The relation $=_x$ is extended to behaviors by requiring the behaviors to have the same length and the corresponding states to be in the relation $=_x$.

A safety property M is *x-invariant* if for every $\sigma \in M$ if $\sigma =_x \tau$ then $\tau \in M$. Further, the *existential quantification* $\exists x.M$ of M along x is the least x-invariant safety property containing M.

Clearly, if N is x-invariant, then we have:

$$\frac{M \vdash N}{\exists x.M \vdash N} \tag{3}$$

and the converse rule is sound too. In a specification logic, the x-invariance hypothesis would be signalled syntactically by a free-variable proviso. Taking N to be $\exists x.M$, the converse yields that:

$$M \vdash \exists x.M \tag{4}$$

Now we need substitution in order to derive a refinement rule. Let $f : \mathbf{S} \rightarrow \mathbf{S}$ be any function from states to states, and extend it to behaviors in the evident way so that $f(\sigma \cdot (a, s)) = f(\sigma) \cdot (a, f(s))$. Then we have $f^{-1} : \mathcal{S}_b \rightarrow \mathcal{S}_b$. Let $g : \mathbf{V} \rightarrow \mathbf{V}$; the substitution of $g(y)$ for x in a safety property M is defined by $M[g(y)/x] = f^{-1}(M)$, where $f(s)(z)$ equals $g(s(y))$ if $z = y$, and $s(z)$ otherwise. As any f^{-1} is monotone, we get in particular that for any safety properties M and N,

$$\frac{M \vdash N}{M[g(y)/x] \vdash N[g(y)/x]}$$

If N is x-invariant then $N[g(y)/x] = N$, that is, N is unchanged by substitution for x. We apply this in (4) and get:

$$M[g(y)/x] \vdash \exists x.M$$

Then combining this with (3) we obtain a refinement rule:

$$\frac{N \vdash M[g(y)/x]}{\exists y.N \vdash \exists x.M} \tag{5}$$

There is an evident generalization to multiple quantifications.

Refinement principles in the literature deal with properties of special forms. For example, in [Lam89], these properties are given by an initial condition and a next-state relation (and a progress property, which relates only to liveness). Our intention is that such principles can be derived from the treatment of existential quantification above, plus considerations relating to the definition of M using the prefixing construction and recursion, treated below.

Refinement functions are not enough in general: even if the conclusion of (5) holds, it may be that no suitable g exists. The various known remedies for this incompleteness rely on auxiliary variables, on multivalued mappings, or on a combination of these.

In a logical setting, auxiliary variables can be introduced with the help of two more general forms of semantic substitution. A certain *forward* substitution enables the use of *history* variables and a *backward* substitution enables the use of *prophecy* variables. Some complications arise as—unlike the standard kind of substitution dealt with above—the new substitutions can only be approximated in the syntax of the specification logic (given in section 6). For practical applications, one should also take account of stuttering and, perhaps, other closure conditions suggested by the work on classical linear logic (section 5). We intend to treat these issues in more detail in the full version of this paper.

4 Intuitionistic Linear Logic

In this section we develop the intuitionistic linear logic proposed in the overview. The study of classical linear logic is postponed to the next section.

We assume given only a set of states $\mathbf{S}$; there is no notion of agent in this calculus. A *transition* is a pair of states. A *process* is a prefix-closed set of sequences of transitions. (Note that the empty sequence ϵ is allowed.) The set of all processes is denoted by $\mathcal{P}$.

A set of processes M is *left-closed* if $q \in M$ whenever there exists $p \in M$ such that $q \subseteq p$. A *safety property* is a set of processes which is both left-closed and closed under arbitrary unions. The set of all safety properties is denoted by $\mathcal{S}_p$.

Every safety property has a largest process, and in fact:

Proposition 2 *The function from $\mathcal{S}_p$ to $\mathcal{P}$ that maps a safety specification to its largest element is an isomorphism of partial orders.*

Below, then, we define some operations on $\mathcal{P}$ rather than on $\mathcal{S}_p$.

The complete Heyting algebra operations $\wedge$ and $\vee$ are intersection and union, and the associated $\rightarrow$ is

$$M \rightarrow M' = \{u \mid \forall n \geq 0.\ \text{if } u|_n \in M \text{ then } u|_n \in M'\}$$

where the prefix $u|_n$ is defined as usual for sequences. The empty set (falsehood) is written $\mathbf{0}$.

If p_1 and p_2 are two processes, their parallel composition is $p_1 \,||\, p_2$, where $||$ is the language shuffle operator. Since processes and specifications are identified, the parallel composition of specifications is defined as shuffle too: $M \otimes M' = M \,||\, M'$.

Proposition 3 $(\mathcal{S}_p, \bigcup, \mathbf{1}, \otimes)$ *is a commutative quantale, where $\mathbf{1} = \{\epsilon\}$.*

The associated quantalic implication is then given by

$$M_1 \multimap M_2 = \{u \mid (\{u\} \;\, || \, M_1) \subseteq M_2\}$$

It follows immediately that the algebra of safety specifications provides a model of intuitionistic linear logic [Yet90]. Parallel composition is the multiplicative conjunction operation, while $\wedge$ and $\vee$ are the additives.

The exponential ! can be treated too, but it is not interesting: $!M = \mathbf{1}$ for all $M \neq \mathbf{0}$, and $!\mathbf{0} = \mathbf{0}$. Instead, a $(\ )^*$ operation is available; M^* is defined as $\bigvee_i M^i$, where M^i is the i-fold parallel composition of M with itself, and it represents an arbitrary number of M processes running in parallel.

We have only started to consider existential quantifiers and refinement in a linear logic; they seem to pose no new problems. We omit their discussion here.

Composition

A transition sequence is *chained* if it is of the form

$$(s_1, s_2)\,(s_2, s_3)\ \ldots\ (s_{n-2}, s_{n-1})\,(s_{n-1}, s_n)$$

(The sequences ϵ and (s_1, s_2) are chained.) Intuitively, chained transition sequences correspond to runs of a system by itself, with no interference from the environment. We write $u \smile v$ (and say that u *runs* with v) if u and v have a chained shuffle.

Assumption-guarantee specifications are made possible by a new connective $\multimap\!\!\diamond$. We set:

$$M^\dagger = \{u \mid \exists v \in M.\ u \smile v\}$$

and

$$M_1 \multimap\!\!\diamond\, M_2 = M_1^\dagger \rightarrow M_2$$

The definition says that if a prefix u of a sequence in $M_1 \multimap\!\!\diamond M_2$ has a chained shuffle with a sequence in M_1, then u is in M_2. Hence, the sequences in $M_1 \multimap\!\!\diamond M_2$

cannot be distinguished from sequences in M_2 by an M_1 environment.

Various formulations of the Composition Principle are now available. One of the most powerful ones we have considered is:

$$\frac{E \otimes M_2 \vdash M_1 \multimap E_1 \qquad E \otimes M_1 \vdash M_2 \multimap E_2}{(E_1 \multimap M_1) \otimes (E_2 \multimap M_2) \vdash (E \multimap M_1 \otimes M_2)}$$

Another variant applies to the composition of an arbitrary number of like processes that depend on one another, in an environment E:

$$\frac{(E \otimes M^* \multimap M)^*}{E \multimap M^*}$$

As in the intuitionistic calculus, the Composition Principle can be extended to handle initial conditions. This extension is less elegant than in the intuitionistic calculus, however, as it requires a generalization of the $\multimap$ connective.

5 Classical Linear Logic

With a notion of testing, the linear logic of specifications goes from intuitionistic to classical. The notion of testing we adopt is an internal one: a distinguished state *fail* represents the result of failed tests. As *fail* represents a fatal failure, we restrict attention to transition sequences that do not contain *fail* except possibly as their last state. In particular, the shuffle operation is redefined to respect this restriction; when we shuffle the sequences (s_0, fail) and (s_1, s_2), for example, we drop (s_1, s_2) from the shuffle $(s_0, \mathit{fail})(s_1, s_2)$, and say that (s_0, fail) is a shuffle of (s_0, fail) and (s_1, s_2). For simplicity, we also assume given a fixed initial state $\mathit{start} \neq \mathit{fail}$, and require chained transition sequences to start at *start*.

A chained transition sequence *succeeds* if it does not end with *fail*, and *fails* otherwise. A process *succeeds* if all of its chained transition sequences succeed. The set of processes that succeed is left-closed and closed under arbitrary unions, and hence it is a safety property; we denote this safety property by $\perp$. A process p *passes* a test q if $p \,||\, q$ succeeds. Note that p passes q if and only if q passes p, and there is no formal distinction between tester and testee.

Abramsky has suggested related but opposite definitions in a CCS framework [Vic88]. He proposed taking $\perp$ as the set of processes that fail. Our choice of $\perp$, the processes that do not fail, is more appropriate for a calculus of safety properties; in our model, Abramsky's $\perp$ is not left-closed.

A negation-like connective $(\ )^\perp$ is habitually defined from $\perp$. For an arbitrary set of processes M, the standard definition gives:

$$M^\perp = \{p \mid \forall q \in M.\ (p \,||\, q) \in \perp\}$$

In other words, $M^\perp$ is the largest (the hardest) set of tests that M processes pass, and it equals $M \multimap \perp$. Clearly, $M^\perp$ is always a safety property. When M is a safety property, it is easy to understand $M^\perp$ as a process:

$$M^\perp = \{u \mid \forall v \in M.\ \text{not } u \frown v\}$$

where we write $u \frown v$ (and say that u *crashes* with v) if u and v have a chained shuffle that fails.

As usual, we focus on the *facts*—the propositions M such that $(M^\perp)^\perp = M$. A useful characterization of the facts can be given in terms of a relation $\sqsupseteq$. The relation $\sqsupseteq$ is defined to be the smallest preorder such that:

$$uv \sqsupseteq u \qquad u(r,s)(s,t)v \sqsupseteq u(r,t)v \qquad uv \sqsupseteq u(s,s)v$$

$$(\mathit{start}, \mathit{start})u \sqsupseteq u \qquad u(s, \mathit{fail}) \sqsupseteq u(s,t)v$$

A safety property M is *downward-closed* if $v \in M$ whenever there exists $u \in M$ such that $u \sqsupseteq v$; as we shall see, M is a fact if and only if it is downward-closed.

Given a transition sequence $u = (s_1, t_1) \ldots (s_n, t_n)$, let $u^\flat = (t_1, s_2) \ldots (t_{n-1}, s_n)$ if $s_1 = \mathit{start}$, and $u^\flat = (\mathit{start}, s_1)(t_1, s_2) \ldots (t_{n-1}, s_n)$ otherwise. Now let $u^\sharp = u^\flat$ if $t_n = \mathit{fail}$, and $u^\sharp = u^\flat(t_n, \mathit{fail})$ otherwise. These definitions apply when $n = 1$, and for example $(s_1, t_1)^\flat$ equals either ϵ or (start, s_1), depending on whether $\mathit{start} = s_1$. In the special case $n = 0$, let $\epsilon^\flat = \epsilon$ and $\epsilon^\sharp = (\mathit{start}, \mathit{fail})$. The operations $(\ )^\flat$ and $(\ )^\sharp$ have some convenient properties:

Proposition 4 *If $w \smile u$ then $w \sqsupseteq u^\flat$.*

Proposition 5 *If $w \frown u^\sharp$ then $w \sqsupseteq u$. If $u \frown v$ and $w \sqsupseteq v$ then $u \frown w$. Moreover, $u \frown u^\sharp$.*

It follows that $(\ )^\perp$ behaves almost like set-theoretic complementation:

Proposition 6 *If M is a downward-closed set of transition sequences, then $u^\sharp \notin M^\perp$ if and only if $u \in M$.*

We obtain also the desired characterization of facts:

Proposition 7 *The set of transition sequences M is a fact if and only if it is downward-closed.*

In classical linear logic, the product $\otimes$ is required to map facts to facts. Thus, from now on, $M_1 \otimes M_2$ denotes $((M_1 \,||\, M_2)^\perp)^\perp$. Similarly, $\mathbf{1}$ is $(\{\epsilon\}^\perp)^\perp$.

A redefinition is not needed for the additives. If M and M' are facts then $M \cap M'$ is a fact, and $M \wedge M'$ is still defined to equal $M \cap M'$. It follows from proposition 7 that $((M \cup M')^\perp)^\perp = (M^\perp)^\perp \cup (M'^\perp)^\perp$, and hence we can keep $M \vee M' = M \cup M'$. These properties of the additives, together with proposition 6, are interestingly reminiscent of the set-theoretic models of relevance logic [Dun86].

This concludes the definition of a model of classical linear logic. Actually, the model can be viewed in two ways, as a quantale of safety properties or as a phase space of processes. The latter option makes us consider sets of processes which are not safety properties, but only safety properties are facts; thus, the choice of viewpoint seems unimportant.

Composition

As suggested in subsection 2.2, let

$$E^+ = E \cup \{u(s, \mathit{fail}) \mid u \in E,\ s \in \mathbf{S}\}$$

Note that E^+ is not a fact in general, even when E is a fact. We obtain:

Proposition 8 *If E and M are facts then*

$$E \multimap M = (E^+ \cap M^\perp)^\perp$$

This description of $\multimap$ leads to a general Composition Principle for the classical linear calculus:

$$\frac{E \otimes M_2 \vdash M_1 \multimap E_1 \qquad E \otimes M_1 \vdash M_2 \multimap E_2}{(E_1 \multimap M_1) \otimes (E_2 \multimap M_2) \vdash (E \multimap M_1 \otimes M_2)}$$

where E, E_1, E_2, M_1, and M_2 are facts. The Principle follows from its analogue in the intuitionistic linear logic (but is not identical to its analogue, because $\otimes$ has been redefined slightly).

In particular, the Composition Principle says that $(E \multimap M) \otimes (M \multimap E) \vdash (E \otimes M)^\circ$. If $E \otimes M \vdash \perp$ then it also follows that $(E \multimap M) \otimes (M \multimap E) \vdash \perp$, and hence $(E \multimap M) \vdash (M \multimap E)^\perp$. Since the converse can be proved too, if $E \otimes M \vdash \perp$ then $(E \multimap M) = (M \multimap E)^\perp$. When $E = M$, this implies that if $M \otimes M \vdash \perp$ then $(M \multimap M) = (M \multimap M)^\perp$, and so $(M \multimap M)$ is self-dual. Furthermore, all self-duals arise thus: if N is self-dual, then $N \otimes N \vdash \perp$ and $N \multimap N = N$.

6 A Specification Language

Here we give a language for specifying safety properties. The language is rather minimal, and is intended as a basis for more practical languages. For example, certain temporal operators are expressible in the language, and it would be quite natural to add abbreviations for them. Similarly, state machines can be coded and some syntax for them might be useful.

We treat the calculus of behaviors first. As in [Lam90], we assume a fixed first-order signature and assume that there are two kinds of first-order variables, primed and unprimed, with a one-to-one correspondence $x \leftrightarrow x'$ between the two; these serve for writing actions. The unprimed variables are those used in section 3.2 above to define states. We let A range over the formulas in this language, X over a countably infinite set of specification variables, and μ over sets of agents.

Formulas are ranged over by P and Q, and are given by the following grammar:

$$\begin{aligned} P \ ::= &\ \mathit{tt} \mid \mathit{ff} \mid P \wedge Q \mid P \vee Q \mid P \rightarrow Q \mid P_\mu \\ &\mid \exists x.P \mid \langle \mu : A\rangle.P \mid A \mathrel{-\!\!\triangleright} P \mid X \mid \mu X.P \end{aligned}$$

Here we have first the propositional connectives, then a μ-closure operator, then a hiding operator (the quantification $\exists x$ binds x' as well as x), then an agent-action prefix construction, then a guarding construction (where no primed variable appears in the guard formula A), and finally, after specification variables, a least fixpoint expression (where it is assumed that all occurrences of X are positive, that is, within an even number of $(\_ \rightarrow Q)$'s). The notation $M \triangleleft \mu$ could be adopted as part of the formal logic, but here it remains in the metalanguage.

All expressions denote safety properties, given values for their free variables. Rather than define the semantics formally, we just point out the semantic operations corresponding to the syntactic ones. The denotations of the propositional connectives are obtained by the appropriate Heyting operations, and that of μ-closure as in section 3. Existential quantification is as in subsection 3.2. For prefixing, we define for any μ, $R \subseteq \mathbf{S}^2$, and $M \in \mathcal{S}_b$:

$$\begin{aligned} \langle \mu : R\rangle.M \ = &\ \{\langle s\rangle \mid s \in \mathbf{S}\} \cup \{\langle s, a, s'\rangle \mid a \in \mu, sRs'\} \\ &\cup \{\langle s, a, s'\rangle \cdot \sigma \mid a \in \mu, sRs', \sigma \in M \text{ begins with } s'\} \end{aligned}$$

where $\langle s, a, s'\rangle \cdot (\sigma \cdot (a', s")) = (\langle s, a, s'\rangle \cdot \sigma) \cdot (a', s'')$ and $\langle s, a, s'\rangle \cdot \langle s'\rangle = \langle s\rangle \cdot (a, s')$. For guarding, we define for any $G \subseteq \mathbf{S}$ and $M \in \mathcal{S}_b$:

$$G \mathrel{-\!\!\triangleright} M = \{\sigma \in M \mid \sigma \text{ begins with an element of } G\}$$

Finally, $\mu X.P[X]$ is interpreted as the least M such that $P[M] = M$.

There is no difficulty in giving a similar language for process safety properties. For example, one may simply add the linear operators $\otimes$ and $\multimap$, plus $\multimap$; one drops the μ-closure operator and also the reference to μ in prefixing.

7 Comparisons

The intuitionistic logic and the linear logic are based on different connectives, and on different semantic models, yet there is a fairly straightforward translation between them. Let σ be a behavior

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots s_{n-1} \xrightarrow{a_n} s_n$$

Let $t_\mu(\sigma)$ be the subsequence of $(s_0, s_1) \dots (s_{n-1}, s_n)$ such that the transition (s_{i-1}, s_i) appears in $t_\mu(\sigma)$ if and only if $a_i \in \mu$. The *runs of p with identity μ* are the behaviors σ such that $t_\mu(\sigma) \in p$. This yields a map $r_\mu : \mathcal{S}_p \to \mathcal{S}_b$. It has a left-inverse $s_\mu : \mathcal{S}_b \to \mathcal{S}_p$, which maps a set of behaviors to the most general process that implements this set of behaviors. The operations of the two calculi can then be related, and for example

$$M_1 \multimap M_2 = s_\mu(r_{\bar{\mu}}(M_1) \to r_\mu(M_2))$$

where μ is any nontrivial set of agents.

The intuitionistic logic captures an external view of processes, via their behaviors. The notation $M \lhd \mu$ makes it possible to express who is the subject of a specification. Linear logic specifications describe a process at a time, and hence the notion of "constrains at most" is unnecessary. On the other hand, it becomes more difficult to express that one process is the complete environment of another, and that the system that they form is closed. Such closed systems are essential in the notion of testing, which then helps in the analysis of assumption-guarantee specifications.

Acknowledgements

Luca Cardelli encouraged this work in its initial phase, and tried some of the ideas in a CCS setting. Cynthia Hibbard provided editorial help.

References

[Abr79] K. Abrahamson. Modal logic of concurrent nondeterministic programs. In *International Symposium on Semantics of Concurrent Computation*, Evian-les-Baines, July 1979.

[Abr89] Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 1989.

[AL88] Martín Abadi and Leslie Lamport. The existence of refinement mappings. Research Report 29, Digital Equipment Corporation Systems Research Center, 1988. To appear in *Theoretical Computer Science*. A preliminary version appeared in *Proceedings of the Third Annual Symposium on Logic In Computer Science*, pages 165-177, IEEE Computer Society, Edinburgh, Scotland, July 1988.

[AL90] Martín Abadi and Leslie Lamport. Composing specifications. Research Report 66, Digital Equipment Corporation Systems Research Center, 1990. A preliminary version appeared in [dBdRR90].

[AV90] Samson Abramsky and Steve Vickers. Quantales, observational logic, and process semantics. Technical report, Imperial College, January 1990.

[BKP84] Howard Barringer, Ruurd Kuiper, and Amir Pnueli. Now you may compose temporal logic specifications. In *Sixteenth Annual ACM Symposium on Theory of Computing*, pages 51–63. ACM, April 1984.

[Dam88] Mads Dam. Relevance logic and concurrent computation. In *Proceedings of the Third Symposium on Logic in Computer Science*, pages 178–185. IEEE, July 1988.

[dBdRR90] J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors. *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, volume 430 of *Lecture Notes in Computer Science*, Berlin, 1990. Springer-Verlag.

[Dun86] J. Michael Dunn. Relevance logic and entailment. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 3, pages 117–224. D. Reidel Publishing Co., 1986.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[HP85] David Harel and Amir Pnueli. On the development of reactive systems. In K. R. Apt, editor, *Logics and models of concurrent systems*, volume F13 of *NATO ASI Series*, pages 477–498. Springer-Verlag, 1985.

[HP89] Matthew Hennessy and Gordon Plotkin. Finite conjunctive nondeterminism. In *Petri Festschrift*, 1989.

[Joh82] P. T. Johnstone. *Stone Spaces*. Cambridge University Press, Cambridge, 1982.

[Lam83a] Leslie Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, April 1983.

[Lam83b] Leslie Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Information Processing 83: Proceedings of the IFIP 9th World Congress*, Paris, September 1983. IFIP, North Holland.

[Lam89] Leslie Lamport. A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1):32–45, January 1989.

[Lam90] Leslie Lamport. A temporal logic of actions. Research Report 57, Digital Equipment Corporation Systems Research Center, April 1990.

[LT87] Nancy Lynch and Mark Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Sixth Symposium on the Principles of Distributed Computing*, pages 137–151. ACM, August 1987.

[MC81] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, SE-7(4):417–426, July 1981.

[Mer90] M. Merritt. Completeness theorems for automata. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 544–560. Springer-Verlag, 1990.

[MOM89] Narciso Martí-Oliet and José Meseguer. From Petri nets to linear logic. Technical Report SRI-CSL-89-4R2, SRI International, December 1989.

[Pnu84] Amir Pnueli. In transition from global to modular temporal reasoning about programs. In Krzysztof R. Apt, editor, *Logics and Models of Concurrent Systems*, NATO ASI Series, pages 123–144, Berlin, October 1984. Springer-Verlag.

[Sta85] Eugene W. Stark. A proof technique for rely/guarantee properties. In S. N. Maheshwari, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 206 of *Lecture Notes in Computer Science*, pages 369–391, Berlin, 1985. Springer-Verlag.

[Vic88] Steve Vickers. Samson Abramsky on linear process logics. Foundation Workshop Notes, October-November 1988.

[Yet90] David N. Yetter. Quantales and (noncommutative) linear logic. *Journal of Symbolic Logic*, 55(1):41–64, March 1990.