Disjunction of LOTOS specifications

M.W.A. Steen, H. Bowman, J. Derrick and E.A. Boiten
University of Kent at Canterbury
Computing Lab., The University, Canterbury, Kent CT2 7NF, UK.
Email: mwas@ukc.ac.uk

Abstract

LOTOS is a formal specification language, designed for the precise *description* of open distributed systems and protocols. The definition of, so called, implementation relations has made it possible also to use LOTOS as a *specification technique* for the design of such systems. These LOTOS based specification techniques usually (ab)use non-determinism to achieve *implementation freedom*. Unfortunately, this is unsatisfactory when specifying non-deterministic processes. We, therefore, propose to extend LOTOS with a disjunction operator in order to achieve more implementation freedom while maintaining the possibility to describe non-deterministic processes. In contrast with similar proposals we maintain the operational semantics.

Keywords

LOTOS, process algebra, specification, disjunction, operational semantics

1 INTRODUCTION

In this paper we investigate the extension of the formal specification language LOTOS with a disjunction operator. Such a specification construct could play a role in achieving a more expressive specification technique. As in logic, disjunction can be used to specify a choice between implementation options. If p_1 is an implementation of s_1, and p_2 is an implementation of s_2, then the specification $s_1 \bigvee s_2$ can be implemented by either p_1 or p_2. Thus, disjunction in specifications leads to greater implementation freedom. This is useful both in the specification of standards, which often describe a number of implementation classes, and in the development of distributed systems, where we do not want to tie the hands of the implementors in the initial specification.

1.1 Interpreting LOTOS specifications

LOTOS is a process algebraic language influenced by the earlier process calculi CCS [Mil89] and CSP [Hoa85]. For example, it has inherited the powerful idea of multi-way synchronisation, enabling constraint-oriented specification, from CSP. On the other hand, the language has been given an operational semantics much in the style of CCS.

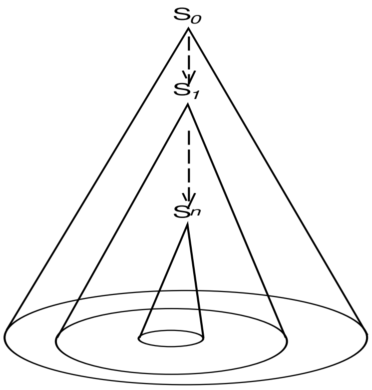


Figure 1 Design process

The operational semantics associates a labelled transition system (LTS) with each process description. The usual interpretation of a LOTOS "specification" is the set of processes that are indistinguishable (w.r.t. some notion of equivalence) from the LTS associated with it. However, this view requires that the observational behaviour of implementations is completely determined already by their initial specifications.

An alternative, more relaxed, view is, that implementations are related to specifications by a, so called, *implementation relation**. Implementation relations are usually not equivalences and, therefore, allow the behaviour of implementations to be somehow more determined than the behaviour described by their specifications. Moreover, they induce a *refinement* ordering between specifications, which enables an incremental design process as depicted in figure 1. An initial abstract specification, allowing many possible implementations, goes through a series of consecutive refinement steps, each restricting the implementation space, until a final implementation is reached.

1.2 The problem

Several researchers have investigated the use of implementation relations with LOTOS to obtain a *specification technique* for concurrent processes (see section 3). Most of these approaches are inspired by CSP's failures/divergences semantics, or have been derived from testing theory. *Non-determinism* is usually (ab)used to achieve implementation freedom. We argue that this is not always satisfactory. In particular, we show that it becomes impossible to specify inherently non-deterministic processes adequately, and the wide-spread use of internal actions as an abstraction mechanism can lead to counter-intuitive implementations.

---

*Implementation relations are sometimes also referred to as *conformance relations* or *satisfaction relations*.

2 LOTOS: SYNTAX AND SEMANTICS

In order not to clutter the presentation of our main ideas, we will only consider a small subset of the operators that LOTOS offers for the structuring of process descriptions. The subset we use is inductively defined by the following grammar:

$$P ::= \mathbf{stop} \mid a;P \mid i;P \mid P \,[]\, P \mid P \,|[G]|\, P \mid X$$

Here we assume that a set of action labels $\mathcal{L}$ is given. Then, $a \in \mathcal{L}$; i is the unobservable, or internal, action; $G \subseteq \mathcal{L}$; and X is a process name. We will assume that a definition exists for each process name used. Process definitions are written $X := P$, where P is a behaviour expression that can again contain process names, including possibly X itself, thus making the definition recursive. The set of all processes is denoted by $\mathcal{P}$, elements of $\mathcal{L}$ by $a, b, c \ldots$, and elements of $\mathcal{L} \cup \{i\}$ by μ.

The operational semantics for LOTOS associates a labelled transition system with each behaviour description through the axioms and inference rules given in table 1.

Table 1 Inference rules

	$\vdash$	$a;P \xrightarrow{a} P$
	$\vdash$	$i;P \xrightarrow{i} P$
$P \xrightarrow{\mu} P'$	$\vdash$	$P \,[]\, Q \xrightarrow{\mu} P'$
$Q \xrightarrow{\mu} Q'$	$\vdash$	$P \,[]\, Q \xrightarrow{\mu} Q'$
$P \xrightarrow{\mu} P', \mu \notin G$	$\vdash$	$P \,|[G]|\, Q \xrightarrow{\mu} P' \,|[G]|\, Q$
$Q \xrightarrow{\mu} Q', \mu \notin G$	$\vdash$	$P \,|[G]|\, Q \xrightarrow{\mu} P \,|[G]|\, Q'$
$P \xrightarrow{a} P', Q \xrightarrow{a} Q', a \in G$	$\vdash$	$P \,|[G]|\, Q \xrightarrow{a} P' \,|[G]|\, Q'$
$P \xrightarrow{\mu} P', X := P$	$\vdash$	$X \xrightarrow{\mu} P'$

The LTS for a process p is $\langle D_p, L_p, T_p, p\rangle$. Here T_p is the smallest set of transitions that can be inferred from p under the given inference rules; D_p is the set of processes derivable from p under the transitions in T_p; $L_p = \{a \mid (s, a, s') \in T_p\}$, the set of action labels.

2.1 Further notation

For the rest of the paper we need some more derived notation. Let $\mathcal{L}^*$ denote strings over $\mathcal{L}$. The constant $\epsilon \in \mathcal{L}^*$ denotes the empty string, and the variables σ, σ_i are used to range over $\mathcal{L}^*$. Elements of $\mathcal{L}^*$ are also called traces. In table 2 the notion of transition is generalised to traces. We further define *Tr*(p), the set of traces of p, and *Ref*(p, σ), the sets of actions refused by p after the trace σ:
Tr$(p) = \{\sigma \in L^* \mid p \stackrel{\sigma}{\Longrightarrow} \}$
Ref$(p, \sigma) = \{X \subseteq L \mid \exists p' : p \stackrel{\sigma}{\Longrightarrow} p' \text{ and } \forall a \in X : p' \stackrel{a}{\not\Longrightarrow} \}$.

Table 2 Derived transition denotations

Notation	Meaning
$p \xrightarrow{\mu}$	$\exists p' : p \xrightarrow{\mu} p'$
$p \not\xrightarrow{\mu}$	$\not\exists p' : p \xrightarrow{\mu} p'$
$\overset{\epsilon}{\Longrightarrow}$	reflexive and transitive closure of $\xrightarrow{i}$
$p \overset{a\sigma}{\Longrightarrow} p'$	$\exists q, q' : p \overset{\epsilon}{\Longrightarrow} q \xrightarrow{a} q' \overset{\sigma}{\Longrightarrow} p'$
$p \overset{\sigma}{\Longrightarrow}$	$\exists p' : p \overset{\sigma}{\Longrightarrow} p'$
$p \overset{\sigma}{\not\Longrightarrow}$	$\not\exists p' : p \overset{\sigma}{\Longrightarrow} p'$

2.2 Equivalence

Often transition systems are considered to be too discriminating in the sense that processes that are intuitively considered to be equivalent may have different representations. The processes $a; b; \mathbf{stop}$ and $a; (b; \mathbf{stop}\ []\ b; \mathbf{stop})\ []\ a; b; \mathbf{stop}$, for example, have different transition systems, but can both only perform the sequence of actions ab and then deadlock. For this reason several abstracting equivalences have been defined over the LTS model. In this paper, we consider only the strongest of the behavioural equivalences: *strong bisimulation equivalence*. Processes are equivalent iff they can simulate each other. This is indeed the case for the two processes above.

Definition 1 (bisimulation equivalence)
Bisimulation equivalence, $\sim \subseteq \mathcal{P} \times \mathcal{P}$, *is the largest relation such that,* $p \sim q$ *implies*
(i) *Whenever* $p \xrightarrow{\mu} p'$ *then, for some* q', $q \xrightarrow{\mu} q'$ *and* $p' \sim q'$*; and*
(ii) *Whenever* $q \xrightarrow{\mu} q'$ *then, for some* p', $p \xrightarrow{\mu} p'$ *and* $p' \sim q'$.

The choice of equivalence is fairly arbitrary. We could just as well have chosen *weak bisimulation equivalence* or *testing equivalence*. We are, however, interested in creating a specification technique that is as expressive as possible. Since bisimulation equivalence is the strongest behavioural equivalence on processes, and by defining satisfaction (see section 4) as an extension of it, we achieve precisely this.

3 LOTOS AS A SPECIFICATION TECHNIQUE

The "meaning" of a specification, i.e. the set of implementations that it describes, depends on the chosen *satisfaction relation*. Following [Lar90a] and [Led92], we define a *specification technique* to be a pair $\langle \Sigma, sat \rangle$, where Σ is the set of all specifications, and sat is some satisfaction relation. Using the notion of bisimulation from the previous section, we could instantiate sat with $\sim$. However, as argued in the introduction, this would leave very little room for manoeuvring during the implementation phase, because the behaviour of implementations would have to be equivalent to the behaviour of their specifications.

Several asymmetric instantiations for sat have been investigated for LOTOS [BSS87, Led91]. These, so called, *implementation relations* were either derived from CSP's denotational semantics [Hoa85], or from testing theory [NH84].

One of the simplest implementation relations, is the *trace preorder*. It only verifies that the implementation cannot perform sequences of observable actions (traces) that are not allowed by the specification.

Definition 2 (trace preorder) *Let* $p, s \in \mathcal{P}$. $p \leq_{tr} s$ *iff* $Tr(p) \subseteq Tr(s)$.

Example 1 *Let* $s := a; b; \mathbf{stop}[]a; c; \mathbf{stop}$, *then* $p_1 := a; b; \mathbf{stop}$, $p_2 := a; c; \mathbf{stop}$ *and* $p_3 := a; (b; \mathbf{stop}[]c; \mathbf{stop})$ *are all implementations of* s *according to* $\leq_{tr}$. *But, also* $\mathbf{stop}$ *and* $a; \mathbf{stop}$ *are correct, since* $\leq_{tr}$ *does not* require *any behaviour to be implemented.*

The trace preorder is a very weak implementation relation. We cannot use it to specify that anything *must* happen. Another notion of validity is, that for each trace of the specification, the implementation can only refuse whatever the specification refuses after that trace. This is captured by the **conf**-relation, which was derived from testing theory. Here we give an intensional definition in terms of traces and refusals.

Definition 3 (conf) *Let* $p, s \in \mathcal{P}$. $p\,\mathbf{conf}\,s$ *iff* $\forall \sigma \in Tr(s) : Ref(p, \sigma) \subseteq Ref(s, \sigma)$.

Example 2 *For the specifications and processes given in example 1,* p_1, p_2 *and* p_3 *are all correct implementations of* s *according to* ***conf***. *However,* $\mathbf{stop}$ *and* $a; \mathbf{stop}$ *are not, because* s *requires either* b *or* c *to happen after* a.

The relation **red** (sometimes referred to as testing preorder, or failure preorder), which is the intersection of $\leq_{tr}$ and **conf**, gives rise to a specification technique with which we can specify both that certain actions must happen and that certain traces are not allowed. This seems to give a suitable specification technique for concurrent processes.

Example 3 *Suppose we want to specify a class of drinks machines. All machines should initially accept a coin. After that, the implementations should give the user either coffee or tea, or a choice between both. With* $\langle \mathcal{P}, \mathbf{red} \rangle$ *we can capture this class of behaviours with the following specification:*

s := coin; (**i**; coffee; **stop** [] **i**; tea; **stop**)

In the example above, note that s also allows the implementation that non-deterministically offers either coffee or tea, after accepting a coin. Since the non-determinism is solely used for achieving implementation freedom in the specification, we could require that implementations are fully deterministic. In that case we have a specification technique that is suitable for specifying deterministic processes.

Unfortunately, non-determinism is not only used to specify implementation freedom. There are some inherently non-deterministic systems, such as gambling machines. More importantly, non-determinism is needed to model non-deterministic aspects of the environment that we do not control. Examples are lossy, or erroneous communication media. In addition, the LOTOS internal action is sometimes used to model certain implementation details that cannot be modelled in LOTOS. In the example below, we show how reduction of non-determinism can lead to intuitively incorrect implementations in these cases.

Example 4 *In the following specification of a transmission protocol, the internal action is used to abstract from the occurrence of a timeout, which is currently not explicitly expressible in* LOTOS*.*

TP_{spec} := send; (receive_ack; **stop** [] **i** (∗ timeout ∗); error; **stop**)

This protocol sends a packet and then waits for an acknowledgement. If the acknowledgement is not received within a certain time, the protocol gives an error signal.

According to $\mathbf{red}$*, this specification can be implemented by a process that gives an error straight away, which is counter-intuitive.*

TP_{error} := send; error; **stop**

Many more implementation relations exist, but most of them are also based on the assumption that implementations may be more deterministic than specifications. Implementation relations that require implementations to be as deterministic as their specifications are usually equivalence relations, which we have rejected for other reasons.

The solution we pursue in the next section separates the use of non-determinism to achieve implementation freedom from its other uses. A new specification construct is introduced for the specification of implementation options. An implementation is then a (possibly non-deterministic) specification in which all the implementation options have been resolved.

4 DISJUNCTION

In this section, we propose to extend LOTOS with a specification construct for explicitly specifying alternative implementation options. The construct we envisage has similarities to CSP's internal choice, but is closer to logical disjunction. In CSP the specification $P \sqcap Q$ could be implemented by $P[]Q$, but in logic, either P or Q would satisfy $P \vee Q$ (the choice is exclusive). The operator will be called disjunction, and denoted by $\bigvee$, because its properties are very much like those of logical disjunction. In the following $\mathcal{S}$ denotes the set of all specifications satisfying this extended syntax.

Disjunction is an operation on specifications that can be used to compose requirements that do not have to be satisfied simultaneously. In order to satisfy the specification $s \bigvee t$ it is enough to implement either s or t. Disjunction is a specification construct. Disjunctions cannot occur in implementations. Therefore disjunctions should

Table 3 Inference rules for unlabelled transitions

$s \rightarrowtail s'$	$\vdash$	$s[]t \rightarrowtail s'[]t$
$t \rightarrowtail t'$	$\vdash$	$s[]t \rightarrowtail s[]t'$
$s \rightarrowtail s'$	$\vdash$	$s\,\vert[G]\vert\,t \rightarrowtail s'\,\vert[G]\vert\,t$
$t \rightarrowtail t'$	$\vdash$	$s\,\vert[G]\vert\,t \rightarrowtail s\,\vert[G]\vert\,t'$
$s \rightarrowtail s', x := s$	$\vdash$	$x \rightarrowtail s'$

gradually be eliminated from the specification during consecutive refinement steps. Refinement should not reduce non-determinism though.

In order to define the semantics for disjunction operationally, we augment labelled transition systems with a new, unlabelled, transition: $\rightarrowtail$. These unlabelled transitions can only be introduced by the disjunction operator through the following axioms:

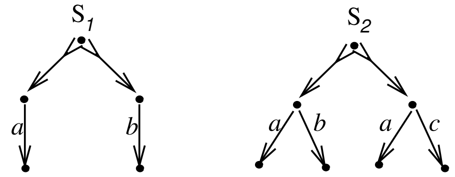
$$\frac{-}{s \bigvee t \rightarrowtail s} \qquad \frac{-}{s \bigvee t \rightarrowtail t}$$

i.e., a disjunction can be resolved through an unlabelled transition. The operational semantics of a specification is now given by an augmented labelled transition system.

Definition 4 (Augmented Labelled Transition System)
An augmented labelled transition system (ALTS) *is a structure $\langle S, L, \rightarrow, \rightarrowtail, s_0\rangle$, with S a set of states, L a set of action labels, $\rightarrow\ \subseteq S \times L \cup \{i\} \times S$ a set of labelled transitions, $\rightarrowtail \subseteq S \times S$ a set of unlabelled transitions, and $s_0 \in S$ the initial state.*

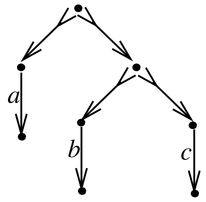
The ALTS for a specification is determined in the usual fashion by the axioms for disjunction given above, and a set of inference rules. The inference rules that determine the normal transition relation, $\rightarrow$, are the same as the normal transition rules for LOTOS given in table 1. The rules for unlabelled transitions are given in table 3. Note that unlabelled transitions are just passed through by all binary operators and recursion. The reason for this is that we do not want a choice, for example, to be resolved by the presence of a disjunction in one of its arguments.

Example 5 *Below we have depicted the transition systems for the specifications $S_1 := a; \mathbf{stop} \bigvee b; \mathbf{stop}$ and $S_2 := (a; \mathbf{stop}[]b; \mathbf{stop}) \bigvee (a; \mathbf{stop}[]c; \mathbf{stop})$.*



In case of nested disjunctions (see example 6) we will usually not be interested in the disjuncts that are again disjunctions themselves. Our interest will be in the "real" disjuncts, i.e., those states that can be reached through a sequence of unlabelled transitions, but which have no outgoing unlabelled transitions themselves. In the remainder of this paper, we therefore use a derived disjunction relation, defined below.

Example 6 *Depicted below is the transition system for* $a;\mathbf{stop} \bigvee (b;\mathbf{stop} \bigvee c;\mathbf{stop})$.



Definition 5 (derived disjunction relations)

1. *For a specification* s*, we define the following predicates:*
   $s \rightarrowtail$ *iff* $\exists s' : s \rightarrowtail s'$ *(s is a disjunction)*
   $s \not\rightarrowtail$ *iff* $\nexists s' : s \rightarrowtail s'$ *(s is not a disjunction)*
2. *For specifications* s *and* t*, we define the following relations:*
   $s \rightarrowtail^* t$ *iff* $t = s \vee \exists s' : s \rightarrowtail s' \wedge s' \rightarrowtail^* t$
   (i.e., the reflexive and transitive closure of $\rightarrowtail$*)*
   $s \rightarrowtail\!\!\!\rightarrow^* t$ *iff* $s \rightarrowtail^* t \wedge t \not\rightarrowtail$

The following lemma gives two useful properties for the $\rightarrowtail\!\!\!\rightarrow^*$-relation.

Lemma 1

1. $s \not\rightarrowtail \iff s \rightarrowtail\!\!\!\rightarrow^* s$;
2. $s \bigvee t \rightarrowtail\!\!\!\rightarrow^* x \iff s \rightarrowtail\!\!\!\rightarrow^* x \vee t \rightarrowtail\!\!\!\rightarrow^* x$.

Proof.

1.
$$
\begin{array}{ll}
 & s \rightarrowtail\!\!\!\rightarrow^* s \\
\Leftrightarrow & \{\text{ definition of } \rightarrowtail\!\!\!\rightarrow^* \} \\
 & s \rightarrowtail^* s \wedge s \not\rightarrowtail \\
\Leftrightarrow & \{\text{ definition of } \rightarrowtail^* \} \\
 & (s = s \vee \exists s' : s \rightarrowtail s' \wedge s' \rightarrowtail^* s) \wedge s \not\rightarrowtail \\
\Leftrightarrow & \{\ s \not\rightarrowtail\ \} \\
 & s = s \wedge s \not\rightarrowtail \\
\Leftrightarrow & \{\text{ reflexivity of } = \} \\
 & s \not\rightarrowtail
\end{array}
$$

2. $s \bigvee t \rightarrowtail\!\!\!\rightarrow^* x$
$\Leftrightarrow$ { definition of $\rightarrowtail\!\!\!\rightarrow^*$ }
$s \bigvee t \rightarrowtail^* x \wedge x \not\rightarrowtail$
$\Leftrightarrow$ { definition of $\rightarrowtail^*$ }
$(x = s \bigvee t \vee \exists x' : s \bigvee t \rightarrowtail x' \wedge x' \rightarrowtail^* x) \wedge x \not\rightarrowtail$
$\Leftrightarrow$ { $s \bigvee t \rightarrowtail s$ and $s \bigvee t \rightarrowtail t$ }
$(s \rightarrowtail^* x \vee t \rightarrowtail^* x) \wedge x \not\rightarrowtail$
$\Leftrightarrow$ { distribution of $\vee$ over $\wedge$ and definition of $\rightarrowtail\!\!\!\rightarrow^*$ }
$s \rightarrowtail\!\!\!\rightarrow^* x \vee t \rightarrowtail\!\!\!\rightarrow^* x$

So far, there is nothing much new. The unlabelled transitions could just as well have been internal actions. The relation $\rightarrowtail\!\!\!\rightarrow^*$ would then correspond to the relation given by $\{(s,t) \mid s \stackrel{\epsilon}{\Longrightarrow} t \wedge t \stackrel{i}{\not\longrightarrow}\}$. However, by introducing a different transition, we separate the specification of alternative implementation options from the use of internal actions and non-determinism. Note that rather than introducing an extra transition relation, we could have introduced another special action label like the i for internal actions.

In the following two sections, we define satisfaction and refinement as extensions of bisimulation equivalence. This is where we deviate from the usual approaches based on refusals.

4.1 Satisfaction

From here on we distinguish between *processes*, or *implementations*, which have no disjunctions, and *specifications*, which may have disjunctions. Processes are in the set $\mathcal{P}$, and specifications are drawn from the set $\mathcal{S}$.

A process intuitively satisfies a specification in case it is equivalent to one of its disjuncts. This intuition is reflected by the formal definition of satisfaction below. Since each disjunct can again have further disjuncts, the definition is inductive. Observe that we have used a "strong" interpretation. There is, however, no reason why this schema could not be applied to weaker interpretations of equivalence, provided they can be characterised inductively.

Definition 6 (Satisfaction)
Satisfaction, $\models \subseteq \mathcal{P} \times \mathcal{S}$, *is the largest relation such that,* $p \models s$ *implies* $\exists s' : s \rightarrowtail\!\!\!\rightarrow^* s'$ *and, for each* $\mu \in L \cup \{i\}$ *the following two conditions hold:*
($\models_1$) *Whenever* $p \stackrel{\mu}{\longrightarrow} p'$, *then* $s' \stackrel{\mu}{\longrightarrow} s''$ *for some* s'' *with* $p' \models s''$; *and*
($\models_2$) *Whenever* $s' \stackrel{\mu}{\longrightarrow} s''$, *then* $p \stackrel{\mu}{\longrightarrow} p'$ *for some* p' *with* $p' \models s''$.

Now, we can instantiate sat with $\models$ to obtain a powerful specification technique for both deterministic and non-deterministic processes.

Example 7 *Going back to the drinks machine specification of example 3, we can now specify the class of drinks machines that serve either coffee or tea as follows:*

S1 := coin; (coffee; **stop** $\bigvee$ tea; **stop**)

Possible implementations, according to $\models$*, are:* coin; coffee; **stop** *and* coin; tea; **stop**. *If we also want to allow the implementation that offers a choice between coffee and tea, after a coin has been accepted, then we should add this as a disjunct to the specification:*

S2 := coin; (coffee; **stop** $\bigvee$ tea; **stop** $\bigvee$ (coffee; **stop** [] tea; **stop**))

Specification S2 in the example above shows that we had to trade-in some conciseness of specifications for clarity of the semantics. We believe that the semantics of logical disjunction will be better understood by most specifiers than the semantics of non-determinism.

Example 8 *In example 4 of the transmission protocol, there was no intended implementation freedom. Since the specification* TP_{spec} *does not contain disjuncts, the only possible implementation (modulo bisimulation equivalence) is the specification itself.*

The following proposition confirms that the $\bigvee$-operator behaves like logical disjunction.

Proposition 7 *Let* $s, t \in \mathcal{S}$ *be specifications, and* $p \in \mathcal{P}$ *be a process. Then* $p \models (s \bigvee t) \Leftrightarrow (p \models s) \vee (p \models t)$.

Proof.

$$\begin{array}{ll}
 & p \models (s \bigvee t) \\
\Leftrightarrow & \{ \text{ definition of } \models \} \\
 & \exists x : (s \bigvee t) \rightarrowtail\!\!\!\rightarrow^* x \wedge (\text{conditions } \models_1 \text{ and } \models_2 \text{ hold for}(p, x)) \\
\Leftrightarrow & \{ \text{ lemma 1.2 } \} \\
 & \exists x : (s \rightarrowtail\!\!\!\rightarrow^* x \vee t \rightarrowtail\!\!\!\rightarrow^* x) \wedge (...) \\
\Leftrightarrow & \{ \text{ distr. of } \wedge \text{ over } \vee \text{ and distr. of } \exists \} \\
 & (\exists x : s \rightarrowtail\!\!\!\rightarrow^* x \wedge (...)) \vee (\exists x : t \rightarrowtail\!\!\!\rightarrow^* x \wedge (...)) \\
\Leftrightarrow & \{ \text{ definition of } \models \} \\
 & p \models s \vee p \models t
\end{array}$$

Because of this connection with logical disjunction, $\bigvee$ also enjoys the following properties.

Corollary 8 *Let* $r, s, t \in \mathcal{S}$ *be specifications, and let* $p \in \mathcal{P}$ *be a process. Then:*

1. $p \models s \Leftrightarrow p \models (s \bigvee s)$ *(idempotency);*
2. $p \models (s \bigvee t) \Leftrightarrow p \models (t \bigvee s)$ *(symmetry);*

3. $p \models (r \bigvee (s \bigvee t)) \Leftrightarrow p \models ((r \bigvee s) \bigvee t)$ *(associativity).*

It is not hard to see, that the equivalence over processes induced by the specification technique $\langle \mathcal{S}, \models \rangle$ is precisely strong bisimulation equivalence.

Proposition 9 (process equivalence)
Let $p, q \in \mathcal{P}$ be processes, then $p \sim q \Longleftrightarrow \forall s \in \mathcal{S} : (p \models s \Leftrightarrow q \models s)$.

Proof. (sketch) The proof for this proposition is similar to the proof that bisimulation equivalence is characterised by Hennessy-Milner logic in [Mil89, p.229]. It involves giving alternative characterisations of bisimulation and satisfaction as limits of descending chains of approximating relations. These are then used to prove the proposition by induction. □

We can also show that all other operators of the specification language distribute over disjunction. This will be a useful property when we want to establish a normal form for specifications.

Proposition 10 *Let $r, s, t \in \mathcal{S}$ be specifications, and let $p \in \mathcal{P}$ be a process. Then the following distributivity properties hold:*

1. $p \models ((s \bigvee t) \,[]\, r) \Leftrightarrow p \models ((s \,[]\, r) \bigvee (t \,[]\, r))$;
2. $p \models ((s \bigvee t) \,|[G]|\, r) \Leftrightarrow p \models ((s \,|[G]|\, r) \bigvee (t \,|[G]|\, r))$;
3. $p \models ((s \bigvee t) \bigvee r) \Leftrightarrow p \models ((s \bigvee r) \bigvee (t \bigvee r))$.

Proof.

1. From left-to-right: Assume $p \models ((s \bigvee t) \,[]\, r)$. Then, by definition 6, there exists an x such that $((s \bigvee t) \,[]\, r) \rightarrowtail\!\!\!\!\rightarrow^* x$ and conditions ($\models_1$) and ($\models_2$) hold for p and x. Inspection of the inference rules for $\bigvee$ and $[]$ results in the following cases:

   $x = s' \,[]\, r'$, where $s \rightarrowtail\!\!\!\!\rightarrow^* s'$ and $r \rightarrowtail\!\!\!\!\rightarrow^* r'$: Since $((s \,[]\, r) \bigvee (t \,[]\, r)) \rightarrowtail (s \,[]\, r)$ and the fact that $\rightarrowtail \circ \rightarrowtail\!\!\!\!\rightarrow^* = \rightarrowtail\!\!\!\!\rightarrow^*$, we also have $((s \,[]\, r) \bigvee (t \,[]\, r)) \rightarrowtail\!\!\!\!\rightarrow^* x$, and we are done.
   $x = t' \,[]\, r'$, where $t \rightarrowtail\!\!\!\!\rightarrow^* t'$ and $r \rightarrowtail\!\!\!\!\rightarrow^* r'$: Similarly.

   From right-to-left: similar.
2. $((s \bigvee t) \,|[G]|\, r)$ and $((s \,|[G]|\, r) \bigvee (t \,|[G]|\, r))$ have isomorphic transition systems. Both specifications have the following $\rightarrowtail$-derivatives: $s|[G]|r$ and $t|[G]|r$. Neither specification has any other derivatives.
3. Follows from the idempotency, symmetry and associativity of $\bigvee$. □

4.2 Refinement

The definition of satisfaction above, naturally induces a refinement ordering over specifications. A specification s refines a specification t in case the set of processes satisfying s is a subset of the set of processes satisfying t, i.e. $\{p \in \mathcal{P} \mid p \models s\} \subseteq \{p \in \mathcal{P} \mid p \models t\}$. However, generalising definition 6, we can also give an inductive characterisation of refinement:

Definition 11 (Refinement)
Refinement *is the largest relation* $\precsim \subseteq \mathcal{S} \times \mathcal{S}$ *such that,* $s \precsim t$ *implies that for each* s' *such that* $s \rightarrowtail\!\!\!\rightarrow^* s'$*, there exists a* t' *such that* $t \rightarrowtail\!\!\!\rightarrow^* t'$ *and, for each* $\mu \in L \cup \{i\}$ *the following holds:*
(i) *Whenever* $s' \xrightarrow{\mu} s''$ *then, for some* t'', $t' \xrightarrow{\mu} t''$ *and* $s'' \precsim t''$*; and*
(ii) *Whenever* $t' \xrightarrow{\mu} t''$ *then, for some* s'', $s' \xrightarrow{\mu} s''$ *and* $s'' \precsim t''$*.*

This definition simply states that s is a refinement of t if there is a disjunct t' in t for each disjunct s' in s, such that s' is "bisimilar" to t'. The following theorem shows that $\precsim$ is indeed a characterisation of refinement for the specification technique $\langle \mathcal{S}, \models \rangle$.

Theorem 12 *Let* $s, t \in \mathcal{S}$ *be specifications. Then*
$s \precsim t \Longleftrightarrow \{p \in \mathcal{P} \mid p \models s\} \subseteq \{p \in \mathcal{P} \mid p \models t\}$

Proof. (sketch) The proof for this theorem goes very much along the lines of the proof in [Mil89, p.229] that bisimulation is characterised by Hennessy-Milner logic. It involves giving alternative definitions for $\precsim$ and $\models$ as decreasing ω-sequences of approximating relations. We then use these to prove the given theorem by induction. □

Proposition 13 *Let* $s, t, r \in \mathcal{S}$ *be specifications, and let* $p \in \mathcal{P}$ *be a process. Then the following laws for disjunction will hold:*

1. $s \precsim s \bigvee t$;
2. $t \precsim s \bigvee t$;
3. *If* $s \precsim r$ *and* $t \precsim r$*, then* $s \bigvee t \precsim r$.

In other words, $s \bigvee t$ *is the least upper bound of s and t with respect to the refinement ordering.*

Proof. 1. and 2. follow immediately from definition 11, because $s \rightarrowtail\!\!\!\rightarrow^* s'$ implies $(s \bigvee t) \rightarrowtail\!\!\!\rightarrow^* s'$ (using lemma 1), and similarly for t.

3. We prove that the assumption that there is a specification r, such that $s \precsim r$ and $t \precsim r$, but $s \bigvee t \not\precsim r$ leads to a contradiction.

According to definition 11, $s \bigvee t \not\precsim r$ can only hold, if there exists an x, such that $(s \bigvee t) \rightarrowtail\!\!\!\rightarrow^* x$, and for all r' such that $r \rightarrowtail\!\!\!\rightarrow^* r'$ either of the two conditions of definition 11 does not hold. However, if $(s \bigvee t) \rightarrowtail\!\!\!\rightarrow^* x$, then (by lemma 1) either $s \rightarrowtail\!\!\!\rightarrow^* x$ or $t \rightarrowtail\!\!\!\rightarrow^* x$. Since we assumed that $s \precsim r$ and $t \precsim r$, there must exist an r' such that $r \rightarrowtail\!\!\!\rightarrow^* r'$ and x and r' satisfy the two conditions, which gives us the contradiction. $\square$

Next, we show that refinement, $\precsim$, is a (pre-)congruence. That is, refinement is preserved by all specification operators.

Proposition 14 *Let $s_1, s_2, t \in \mathcal{S}$ be specifications, such that $s_1 \precsim s_2$, then*

1. $a; s_1 \precsim a; s_2$
2. $s_1 \; [] \; t \precsim s_2 \; [] \; t$
3. $s_1 \; |[G]| \; t \precsim s_2 \; |[G]| \; t$
4. $s_1 \bigvee t \precsim s_2 \bigvee t$

Proof. The first case is trivial. The other cases can easily be proved by constructing a relation that contains the pair (LHS,RHS) and then showing that this relation is contained in $\precsim$. Here, we prove just the last case.

Consider the relation $\{(s_1 \bigvee t, s_2 \bigvee t) \mid s_1 \precsim s_2\} \cup \precsim$. Whenever $s_1 \bigvee t \rightarrowtail\!\!\!\rightarrow^* x$ then either of the following two cases holds:

$s_1 \rightarrowtail\!\!\!\rightarrow^* x$: Since $s_1 \precsim s_2$ there exists a y such that $s_2 \rightarrowtail\!\!\!\rightarrow^* y$ and x and y satisfy the two conditions of definition 11. Since $(s_2 \bigvee t) \rightarrowtail s_2$, also $(s_2 \bigvee t) \rightarrowtail\!\!\!\rightarrow^* y$.

$t \rightarrowtail\!\!\!\rightarrow^* x$: Since $(s_2 \bigvee t) \rightarrowtail t$, also $(s_2 \bigvee t) \rightarrowtail\!\!\!\rightarrow^* x$, and we are done. $\square$

5 APPLICATIONS

In [Hoa85], Hoare gives some examples in which the non-deterministic or, $\sqcap$, is used for loosely specifying change-giving machines in CSP. These specifications can be expressed equally well in our notation, although their interpretation is slightly different.

Example 9 *Consider the following specification of a change-giving machine, which always gives the right change in one of two combinations:*

```
CH1 := in5p;
        ( out1p; out1p; out1p; out2p; CH1
          ⋁
          out2p; out1p; out2p; CH1 )
```

This specification leaves open how *the change should be given. Valid implementations are those which always return one of two possible combinations of change, but also those which return different combinations on each invocation. For example, the implementation given by* CH_I1*, which alternates between the two possible combinations, satisfies* CH1.

CH_I1 := in5p;
out1p; out1p; out1p; out2p;
in5p;
out2p; out1p; out2p; CH_I1

Example 10 *We saw that* CH1 *allows implementations that give different combinations of change on each invocation. The following specification allows only implementations that always give the same combination, but it leaves open which combination it will be.*

CH2 := CH2A $\bigvee$ CH2B
where
CH2A := in5p; out1p; out1p; out1p; out2p; CH2A
CH2B := in5p; out2p; out1p; out2p; CH2B

Although CSP's $\sqcap$ is intended to play a similar role to logical disjunction, CSP's failures preorder allows also implementations that replace the non-deterministic choice by a deterministic one. This will then give the user a choice, at "run-time", which implementation s/he wants. For example, if the specifications CH1 and CH2 had been written with a non-deterministic choice between the alternatives, then both would have allowed the following implementation:

CH_I2 := in5p;
(out1p; out1p; out1p; out2p; CH_I2
[]
out2p; out1p; out2p; CH_I2)

which gives the user a chance to influence which combination of change s/he will get. However, the semantics of $\bigvee$ does not allow CH_I2 as an implementation of either CH1 or CH2, i.e. CH_I2 $\not\models$ CH1, CH2.

5.1 The most undefined specification

The disjunction operator can easily be generalised to work over a set of arguments. For S a set of specifications, $\bigvee S$ denotes the disjunction of all the specifications $s \in S$. The semantics is defined by the following family of axioms:

$$\frac{-}{\bigvee S \rightarrowtail s}(s \in S)$$

In the same fashion, choice, [], can be generalised to ΣS, with $\Sigma\{\} = \mathbf{stop}$.

Using these generalised operators, we can define the most undefined specification, i.e. the specification that allows all processes as implementations, provided the alphabet of labels is finite.

$$U := \bigvee \{ \Sigma \{ \text{a}; U \mid \text{a}{\in}\text{A} \} \mid \text{A}{\subseteq} \mathcal{L} \}$$

Example 11 *Let $\mathcal{L} = \{a, b\}$ be the alphabet. Then the most undefined specification U is given by:*

$$U := \textbf{stop} \bigvee \text{a}; U \bigvee \text{b}; U \bigvee (\text{a}; U \;[]\; \text{b}; U)$$

This most undefined specification is very useful for partial specification. Whenever we want to leave open the behaviour at a certain point, we can just plug-in U. Later on, this can be refined to anything, thus achieving complete implementation freedom.

6 CONCLUSION

Many others before us have recognised the limited expressiveness of process algebras for the specification of non-deterministic, concurrent processes. A common approach has been to define a logic, separate from the process description language, for the specification of properties of processes (e.g. Hennessy-Milner Logic (HML) [HM85] and modal μ-calculus [Koz83]). A clear drawback is that specifications and implementations are in different notations. Step-wise refinement is not possible, and verification can only be done *a posteriori*. In order to alleviate this problem, there have been some attempts to introduce the process structuring operators into these logics. In [Hol89], HML is extended with the CCS operators, and in [BGS89], the same is done for a fragment of the μ-calculus. Unfortunately, these languages have a denotational semantics: each specification is associated with the set of processes that satisfy it. Verifying whether a process satisfies a specification amounts to checking whether it is in that set. Alternatively, the correctness of an implementation can be verified through (in-)equational reasoning.

Another way to increase the expressive power of process algebraic specifications is introduced in [Lar90b], where transitions are decorated with modalities. A distinction is made between *required* and *allowed* transitions. Bisimulation equivalence is then generalised to a refinement relation that ensures that the more concrete specification requires more and allows less. It is also possible to define the equivalent of logical *conjunction* operationally in this model [LSW95]. In fact, it has been shown that the specification technique thus obtained is as expressive as a restricted version of HML [BL92]. The restriction is caused by the inability to adequately express disjunction. However, modal transition systems can be extended with disjunction in the same way we have extended labelled transition systems with disjunction in this paper. Would this then create a specification technique with the full power of HML?

ACKNOWLEDGEMENTS

We would like to thank Rom Langerak for discussing ideas that led to this paper, and the anonymous referees for their useful comments.

REFERENCES

[BGS89] A. Bouajjani, S. Graf, and J. Sifakis. A logic for the description of behaviours and properties of concurrent systems. LNCS 354, pages 398–410, 1989.

[BL92] G. Boudol and K.G. Larsen. Graphical versus logical specifications. *Theoretical Computer Science*, 106:3–20, 1992.

[BSS87] E. Brinksma, G. Scollo, and C. Steenbergen. LOTOS specifications, their implementations and their tests. In *PSTV VI*, pages 349–360, 1987.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, January 1985.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Hol89] S. Holmström. A refinement calculus for specifications in Hennessy-Milner logic with recursion. *Formal Aspects of Computing*, 1(3):242–272, 1989.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, December 1983.

[Lar90a] K.G. Larsen. Ideal specification formalism = expressivity + compositionality + decidability + testability + $\cdots$. In *CONCUR'90*, LNCS 458, pages 33–56, 1990.

[Lar90b] K.G. Larsen. Modal specifications. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems: Proceedings*, LNCS 407, pages 232–246. Springer-Verlag, 1990.

[Led91] G. Leduc. *On the Role of Implementation Relations in the Design of Distributed Systems using LOTOS*. PhD thesis, University of Liège, Belgium, June 1991.

[Led92] G. Leduc. A framework based on implementation relations for implementing LOTOS specifications. *Computer Networks and ISDN Systems*, 25:23–41, 1992.

[LSW95] K.G. Larsen, B. Steffen, and C. Weise. A constraint oriented proof methodology based on modal transition systems. In E. Brinksma, editor, *TACAS'95*, LNCS 1019, pages 17–40, 1995.

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

[NH84] R. de Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

Ir M.W.A. Steen obtained an MSc(Eng) in Computer Science from the University of Twente, The Netherlands, in 1993. He is currently completing a PhD degree in Computer Science at the University of Kent at Canterbury. His current research focuses on partial specification in process algebra, in particular on techniques for consistency checking and composition. Furthermore he has worked on application of these, and other, formal techniques in the area of Open Distributed Processing.

Dr. H. Bowman, **Dr. J. Derrick** and **Dr.Ir. E.A. Boiten** are lecturers in the Computing Laboratory at the University of Kent.

APPENDIX

This appendix contains a complete proof for theorem 12. Our strategy is similar to the proof that bisimulation is characterised by Hennessy-Milner logic in [Mil89, chap. 10]. It involves alternative characterisations for satisfaction, $\models$, and refinement, $\sqsubseteq\!\!\!\sim$, as limits of descending chains of approximating relations.

Firstly, we define functions $\mathcal{F}_{\models} : \wp(\mathcal{P} \times \mathcal{S}) \to \wp(\mathcal{P} \times \mathcal{S})$, and $\mathcal{F}_{\underset{\sim}{\sqsubset}} : \wp(\mathcal{S} \times \mathcal{S}) \to \wp(\mathcal{S} \times \mathcal{S})$, as follows:

Definition 15 *If $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{S}$, then $(p, s) \in \mathcal{F}_{\models}(\mathcal{R})$ iff there exists an s' such that $s \rightarrowtail\!\!\!\rightarrow^* s'$ and, for each $a \in L \cup \{i\}$:*

($\models_1$) *Whenever $p \xrightarrow{a} p'$, then $s' \xrightarrow{a} s''$ for some s'' with $p' \mathcal{R} s''$; and*
($\models_2$) *Whenever $s' \xrightarrow{a} s''$, then $p \xrightarrow{a} p'$ for some p' with $p' \mathcal{R} s''$.*

Definition 16 *If $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$, then $(s, t) \in \mathcal{F}_{\underset{\sim}{\sqsubset}}(\mathcal{R})$ iff for each s' such that $s \rightarrowtail\!\!\!\rightarrow^* s'$, there exists a t' such that $t \rightarrowtail\!\!\!\rightarrow^* t'$ and, for each $a \in \widetilde{L} \cup \{i\}$:*

($\underset{\sim}{\sqsubset}_1$) *Whenever $s' \xrightarrow{a} s''$, then $t' \xrightarrow{a} t''$ for some t'' with $s'' \mathcal{R} t''$; and*
($\underset{\sim}{\sqsubset}_2$) *Whenever $t' \xrightarrow{a} t''$, then $s' \xrightarrow{a} s''$ for some s'' with $s'' \mathcal{R} t''$.*

Observe that the conditions in these definitions are the same as in definitions 6 and 11 of satisfaction and refinement. In fact, it can be shown that $\models$ is the greatest fixed-point of $\mathcal{F}_{\models}$, and that $\underset{\sim}{\sqsubset}$ is the greatest fixed-point of $\mathcal{F}_{\underset{\sim}{\sqsubset}}$.

Proposition 17

1. *$\models$ is the greatest fixed-point of $\mathcal{F}_{\models}$; and*
2. *$\underset{\sim}{\sqsubset}$ is the greatest fixed-point of $\mathcal{F}_{\underset{\sim}{\sqsubset}}$.*

Proof. We outline the proof for 1. The other proof is similar.

First, observe that the function $\mathcal{F}_{\models}$ is monotonic, i.e. it preserves the standard subset-ordering on relations. Then, by the Knaster-Tarksi fixed-point theorem, it has a greatest fixed-point given by

$$\mathsf{gfp}(\mathcal{F}_{\models}) = \bigcup \{\mathcal{R} \mid \mathcal{R} \subseteq \mathcal{F}_{\models}(\mathcal{R})\}$$

Finally, if we change the word 'implies' in definition 6 to '$\subseteq$', then we obtain that $\models$ is the largest relation such that $\models \subseteq \mathcal{F}_{\models}(\models)$. Hence, $\models = \mathsf{gfp}(\mathcal{F}_{\models})$.

Now, we are ready to give alternative characterisations for $\models$ and $\underset{\sim}{\sqsubset}$ as the limits of descending chains of approximating relations. Definitions 18 and 19 define sequences of relations $\models_0, \models_1, \cdots, \models_\alpha, \cdots$ and $\underset{\sim}{\sqsubset}_0, \underset{\sim}{\sqsubset}_1, \cdots, \underset{\sim}{\sqsubset}_\alpha, \cdots$ for each ordinal number $\alpha \in \mathcal{O}$, starting with the universal relation.

Definition 18 (satisfaction)
$\models_0 = \mathcal{P} \times \mathcal{S}$; $\models_{\beta+1} = \mathcal{F}_{\models}(\models_\beta)$; and $\models_\alpha = \bigcap_{\beta<\alpha} \models_\beta$ for each limit ordinal α.

Definition 19 (refinement)
$\underset{\sim}{\sqsubset}_0 = \mathcal{S} \times \mathcal{S}$; $\underset{\sim}{\sqsubset}_{\beta+1} = \mathcal{F}_{\underset{\sim}{\sqsubset}}(\underset{\sim}{\sqsubset}_\beta)$; and $\underset{\sim}{\sqsubset}_\alpha = \bigcap_{\beta<\alpha} \underset{\sim}{\sqsubset}_\beta$ for each limit ordinal α.

Another way of looking at these definitions is, that $\models_\alpha$ is equal to the α-fold application of $\mathcal{F}_{\models}$ to $\mathcal{P} \times \mathcal{S}$, i.e. $\models_\alpha = \mathcal{F}^\alpha_{\models}(\mathcal{P} \times \mathcal{S})$.

The following proposition shows that the sequences of relations thus defined, form non-strictly decreasing chains.

Proposition 20 *If $\alpha > \beta$ then*

1. $\models_\alpha \subseteq \models_\beta$*;*
2. $\precsim_\alpha \subseteq \precsim_\beta$.

Proof. 1. By transfinite induction and the monotonicity of $\mathcal{F}_{\models}$.

Certainly, $\models_1 = \mathcal{F}_{\models}(\models_0) \subseteq \models_0$. For all successor ordinals, $\beta + 1$, we have $\mathcal{F}^{\beta+1}_{\models}(\models_0) \subseteq \mathcal{F}^{\beta}_{\models}(\models_0)$, because of the monotonicity of $\mathcal{F}_{\models}$. For each limit ordinal α, we have $\mathcal{F}^\alpha_{\models}(\models_0) = \bigcap_{\beta<\alpha} \mathcal{F}^\beta_{\models}(\models_0)$. Since $\langle \wp(\mathcal{P} \times \mathcal{S}), \subseteq\rangle$ is a complete lattice, the glb $\bigcap\{\mathcal{F}^\beta_{\models}(\models_0) \mid \beta < \alpha\}$ exists and is below each $\mathcal{F}^\beta_{\models}(\models_0)$, i.e. $\forall \beta < \alpha : \models_\alpha \subseteq \models_\beta$. Hence, we have a descending chain

$$\cdots \subseteq \models_\alpha \subseteq \cdots \subseteq \models_{\beta+1} \subseteq \models_\beta \subseteq \cdots \subseteq \models_1 \subseteq \models_0$$

for all $\alpha \in \mathcal{O}$.

The other proof is similar.

With results from fixed-point theory, it can be shown that the limit of these decreasing chains are indeed the relations $\models$ and $\precsim$. For completeness sake we provide the proof here though.

Proposition 21

1. $\models = \bigcap_{\alpha \in \mathcal{O}} \models_\alpha$*;*
2. $\precsim = \bigcap_{\alpha \in \mathcal{O}} \precsim_\alpha$.

Proof. We show that $\bigcap_\alpha \models_\alpha$, the limit of the chain of relations $\models_\alpha$, is a fixed-point of $\mathcal{F}_{\models}$, and equal to the greatest fixed-point of $\mathcal{F}_{\models}$, $\models$.

Let's assume that there is no α for which $\models_\alpha$ is a fixed-point of $\mathcal{F}_{\models}$, i.e. $\forall \alpha \in \mathcal{O} : \mathcal{F}^\alpha_{\models}(\models_0) \neq \mathcal{F}^\alpha_{\models}(\mathcal{F}^\alpha_{\models}(\models_0))$. Then

$$\cdots \subset \models_\alpha \subset \cdots \subset \models_{\beta+1} \subset \models_\beta \subset \cdots \subset \models_1 \subset \models_0$$

is a strictly-descending chain. Eventually this chain must reach the bottom element of the lattice of relations on $\mathcal{P} \times \mathcal{S}$, the empty relation $\emptyset$. In other words, there must be an α such that $\mathcal{F}^\alpha_{\models}(\models_0) = \emptyset$. But, this is a fixed-point of $\mathcal{F}_{\models}$. Contradiction! Hence we can conclude that for some α, $\mathcal{F}^\alpha_{\models}(\models_0)$ is a fixed-point of $\mathcal{F}_{\models}$.

If $\mathcal{F}^\alpha_{\models}(\models_0)$ is a fixed-point of $\mathcal{F}_{\models}$, then for all $\beta > \alpha$, $\mathcal{F}^\beta_{\models}(\models_0) = \mathcal{F}^\alpha_{\models}(\models_0)$. So, the limit of the descending chain $\bigcap_\alpha \models_\alpha = \models_\alpha$, and is therefore a fixed-point of $\mathcal{F}_{\models}$. Since $\models$ is the greatest fixed-point of $\mathcal{F}_{\models}$, we have $\bigcap_\alpha \models_\alpha \subseteq \models$.

It remains to be proved that $\mathcal{F}^\alpha_{\models}(\models_0)$ is the greatest fixed-point $\models$. This can be proved using transfinite induction.

Clearly, $\models\subseteq\models_0$. Assume that $\models\subseteq\models_\beta$, then $\mathcal{F}_{\models}(\models) \subseteq \mathcal{F}_{\models}^{\beta+1}(\models_0)$, by the monotonicity of $\mathcal{F}_{\models}$, which implies that $\models\subseteq \mathcal{F}_{\models}^{\beta+1}(\models_0)$, since $\models$ is a fixed-point of $\mathcal{F}_{\models}$. Finally, assume that $\models\subseteq\models_\beta$ for all $\beta < \alpha$, where α is a limit ordinal. Then $\mathcal{F}_{\models}^{\alpha}(\models_0) = \bigcap\{\mathcal{F}_{\models}^{\beta}(\models_0) \mid \beta < \alpha\} \supseteq \mathcal{F}_{\models}(\models) = \models$, by the monotonicity of $\mathcal{F}_{\models}$. Knowing that $\models\subseteq\models_\alpha$ for all $\alpha \in \mathcal{O}$, we can conclude that $\models\subseteq \bigcap_\alpha \models_\alpha$.

The other proof is similar.

After the ground work above, we now come to the proof of theorem 12. We actually prove a slightly stronger proposition, of which theorem 12 is a corollary.

Proposition 22 *For each $\alpha \in \mathcal{O}$, and specifications s, t:*

$$s \precsim_\alpha t \Leftrightarrow \forall p : p \models_\alpha s \Rightarrow p \models_\alpha t \tag{1}$$

Proof. By transfinite induction over α. Assume that (1) holds for all $\beta < \alpha$.

Base case $(\alpha = 0)$**:** $s \precsim_0 t \Leftrightarrow \forall p : p \models_0 s \Rightarrow p \models_0 t$, which holds trivially, since $s \precsim_0 t \equiv p \models_0 s \equiv p \models_0 t \equiv \mathit{true}$.

Induction step $(\alpha = \beta + 1)$**:** $s \precsim_{\beta+1} t \Leftrightarrow \forall p : p \models_{\beta+1} s \Rightarrow p \models_{\beta+1} t$.

We do a ping-pong proof:

"⇒" Assuming that $s \precsim_{\beta+1} t$ and $p \models_{\beta+1} s$ hold for an arbitrary process p, we prove that $p \models_{\beta+1} t$. That is, we need to prove that $\exists t' : t \rightarrowtail^* t'$ such that, for all $a \in A$:

1. $\forall p' : p \xrightarrow{a} p' \Rightarrow \exists t'' : t' \xrightarrow{a} t'' \wedge p' \models_\beta t''$; and
2. $\forall t'' : t' \xrightarrow{a} t'' \Rightarrow \exists p' : p \xrightarrow{a} p' \wedge p' \models_\beta t''$.

Using $p \models_{\beta+1} s$, we derive that $\exists s' : s \rightarrowtail^* s'$ such that, for all $a \in A$:

$$\forall p' : p \xrightarrow{a} p' \Rightarrow \exists s'' : s' \xrightarrow{a} s'' \wedge p' \models_\beta s'' \tag{2}$$

$$\forall s'' : s' \xrightarrow{a} s'' \Rightarrow \exists p' : p \xrightarrow{a} p' \wedge p' \models_\beta s'' \tag{3}$$

Using $s \precsim_{\beta+1} t$, we next derive that $\exists t' : t \rightarrowtail^* t'$ such that, for all $a \in A$:

$$\forall s'' : s' \xrightarrow{a} s'' \Rightarrow \exists t'' : t' \xrightarrow{a} t'' \wedge s'' \precsim_\beta t'' \tag{4}$$

$$\forall t'' : t' \xrightarrow{a} t'' \Rightarrow \exists s'' : s' \xrightarrow{a} s'' \wedge s'' \precsim_\beta t'' \tag{5}$$

Next, we distinguish two cases:

Case $p \xrightarrow{a} p'$**:**
From (2) we derive that $\exists s'' : s' \xrightarrow{a} s'' \wedge p' \models_\beta s''$. And, by (4), we derive next that $\exists t'' : t' \xrightarrow{a} t'' \wedge s'' \precsim_\beta t''$. And, finally, the induction hypothesis gives us $p' \models_\beta t''$.

Case $t' \xrightarrow{a} t''$**:**
From (5) we derive that $\exists s'' : s' \xrightarrow{a} s'' \wedge s'' \precsim_\beta t''$. And, by (3), we derive next that $\exists p' : p \xrightarrow{a} p' \wedge p' \models_\beta s''$. And, finally, the induction hypothesis gives us $p' \models_\beta t''$.

"$\Leftarrow$" We turn the proposition around and assume that $s \not\precsim_{\beta+1} t$. Next, we look for a process p such that $p \models_{\beta+1} s$ and $p \not\models_{\beta+1} t$.
Since there always is a t' such that $t \rightarrowtail^* t'$ (remember that t' could be t itself), $s \not\precsim_{\beta+1} t$ can only hold if there exists an s' such that $s \rightarrowtail^* s'$ and for each t' such that $t \rightarrowtail^* t'$ either of the following two predicates holds:

1. $\exists s'' : s' \xrightarrow{a} s'' \wedge (\forall t'' : t' \xrightarrow{a} t'' \Rightarrow s'' \not\precsim_{\beta} t'')$
2. $\exists t'' : t' \xrightarrow{a} t'' \wedge (\forall s'' : s' \xrightarrow{a} s'' \Rightarrow s'' \not\precsim_{\beta} t'')$

We consider both cases in turn:

Case 1: For simplicity, we assume that $s' \not\xrightarrow{b}$ for any $b \in A$. Let $\{t_i : i \in I\}$ be the set of all $\xrightarrow{a}$-derivatives of t'. Then for each $i \in I$, since $s'' \not\precsim_{\beta} t_i$, there is by induction a process p_i such that $p_i \models_{\beta} s''$ and $p_i \not\models_{n} t_i$. Now define p to be the process $\mathcal{S}_{i \in I}\, a; p_i$. Then, whenever $p \xrightarrow{a} p_i$, we have $p_i \models_{\beta} s''$, and since $s' \not\xrightarrow{b}$, we have $p \models_{\beta+1} s$. On the other hand, no $\xrightarrow{a}$-derivative of any t', such that $t \rightarrowtail^* t'$, is satisfied by p_i, so $p \not\models_{\beta+1} t$.

Case 2: Again for reasons of simplicity, we assume that $s' \not\xrightarrow{b}$ for any $b \neq a$, and that $s' \xrightarrow{a} s'' \Rightarrow s' \xrightarrow{a} s''$. Let $\{s_i : i \in I\}$ be the set of all $\xrightarrow{a}$-derivatives of s'. Then for each $i \in I$, since $s_i \not\precsim_{\beta} t''$, there is by induction a process p_i such that $p_i \models_{\beta} s_i$ and $p_i \not\models_{n} t''$. Now define p to be the process $\mathcal{S}_{i \in I}\, a; p_i$. Then, whenever $p \xrightarrow{a} p_i$, there is an s_i such that $p_i \models_{\beta} s_i$, and whenever $s' \xrightarrow{a} s_i$ there is a p_i such that $p_i \models_{\beta} s_i$. Hence, we have $p \models_{\beta+1} s$. On the other hand, no a-derivative of p will satisfy t'', so $p \not\models_{\beta+1} t$.

Induction step (α is a limit ordinal):

$$
\begin{array}{cl}
 & s \precsim_{\alpha} t \\
\Leftrightarrow & \{ \text{ definition 19 } \} \\
 & \forall \beta < \alpha : s \precsim_{\beta} t \\
\Leftrightarrow & \{ \text{ induction hypothesis } \} \\
 & \forall \beta < \alpha : \forall p : p \models_{\beta} s \Rightarrow p \models_{\beta} t \\
\Leftrightarrow & \{ \text{ predicate logic } \} \\
 & \forall p : (\forall \beta < \alpha : p \models_{\beta} s) \Rightarrow (\forall \beta < \alpha : p \models_{\beta} t) \\
\Leftrightarrow & \{ \text{ definition 18 } \} \\
 & \forall p : p \models_{\alpha} s \Rightarrow p \models_{\alpha} t
\end{array}
$$

Theorem 12 now follows directly from the proposition above and the fact that $\models = \bigcap_{\alpha} \models_{\alpha}$ and $\precsim = \bigcap_{\alpha} \precsim_{\alpha}$.