URL: http://www.elsevier.nl/locate/entcs/volume40.html 31 pages

May Testing, Non-interference, and Compositionality

Steve Schneider

Department of Computer Science
Royal Holloway, University of London
Egham, Surrey, TW20 0EX

Abstract

This paper uses CSP to introduce a characterisation of non-interference in terms of the deductions that may be made about high level processes by low level tests. May testing yields classic noninference, and has a concise formulation in CSP. It is preserved by a wider range of composition operators than are normally considered in the context of non-interference, and thus also composes under the operators traditionally studied with non-interference. The CSP characterisation of may non-interference also permits some attractive and simple compositionality proofs.

1 Introduction

Compositionality of non-interference properties is essential for constructing secure systems out of secure components. This argument was made by McCullough [8] as motivation for his definition of *restrictiveness*, a property which ensures non-interference and which is also preserved by system composition. In the literature, there are a number of formulations of non-interference (see for example [4,8,18,9,2]), which tend to use state machines or event systems as their system models, and which define notions of system composition with respect to the semantic framework they have set up.

Process algebra provides a mature theory for the modelling of systems built out of a number of components, and offers a number of compositional operators for constructing systems. It is naturally suited to the description of concurrent systems, and provides a framework for their analysis with respect to particular requirements, with established techniques for reasoning.

This paper investigates the notion of non-interference from the point of view of may testing, a technique used in process algebra to characterise process equivalence. One motivation for starting from may testing is that it provides a natural and intuitive understanding of the property that we aim to investigate, and of the systems to which this investigation will apply.

In the context of non-interference, we might wish to claim that a system provides non-interference between a high level interface H and a low level interface L if no interactions at the low level can provide any information about any user at the high level. This is formalised as the requirement that no low level test on the system can distinguish between any two high level users.

This definition is intuitively appealing, but is too cumbersome to work with directly in the analysis of alleged non-interfering systems. We are able to characterise this non-interference property in terms of conditions on the system without explicit reference to high level users. It turns out that corresponds to the property 'noninference' considered by O'Halloran [11] and more recently investigated by Focardi and Gorrieri [3], though we have given a more direct motivation for it, and thus provide additional insight into their formulations.

We investigate its composability through a wide range of CSP composition operators, including forms of parallel combination, sequencing, and hiding. The process-algebraic framework we use yields surprisingly simple proofs of compositionality, generally four or five lines of algebra. The external combinators typically used in studies of non-interference—cross product, cascade, and feedback—can be expressed in terms of the CSP compositions, and so results about these external combinators are easily derived.

2 Notation

Process algebras provide a particular approach to the study of concurrency and interaction. This paper bases its discussion within the framework of the process algebra CSP (Communicating Sequential Processes). A full account of this process algebra can be found in [14,17]. It provides a language for describing interacting systems, together with a semantic theory for understanding them. This section provides a brief reminder of those aspects most relevant to this paper.

The language of CSP is constructed around *events*: instantaneous synchronisations which provide the communication primitive. Events may have some structure, the most common communication being a channel communication of the form $c.v$, where c is the channel name, and v is the value communicated. The set of all events is denoted Σ. If c is a channel name, and T is a set of messages, then $c.T$ denotes the set of all channel communications of messages from T passing along c: $c.T = \{c.v \mid v \in T\}$

Processes are used to describe possible patterns of interaction. In CSP, a process P has an alphabet, or interface, $\alpha(P) \subseteq \Sigma$: the set of events it is able to synchronise on. The process $a \rightarrow P$ describes a process which is initially prepared to engage in the event a, and then subsequently behave as P. The process $c!v \rightarrow P$ describes a process which is prepared to output v on channel c, and behave subsequently as P. The input $c?x : T \rightarrow P(x)$ may take in some value v of type T along channel c and behave subsequently as $P(v)$.

The choice $P \sqcap Q$ may behave non-deterministically either as P or as Q. The choice $P \Box Q$ offers a choice between all the events offered by the processes P and Q. The choice is resolved on the occurrence of the first event in favour of the process that performs it.

Processes may be put in parallel: $P \underset{A}{\parallel} Q$ behaves as P running concurrently with Q, synchronising on events in A, and performing other events independently. Often, A will be the intersection of the alphabets of P and Q. Values are passed between parallel processes by means of synchronisations on the channels, linking an output channel of one to an input channel of another. An interleaving of two processes, $P \,|||\, Q$, simply executes P and Q concurrently without any communication occurring between them. The abstraction mechanism $P \setminus A$ describes the process P with all occurrences of A occurring internally in the resulting process. The process *Stop* can perform no events at all. Thus $P \underset{A}{\parallel} \mathit{Stop}$ behaves as the process P with all occurrences of A blocked. This is different from $P \setminus A$ in which all occurrences of events in A are made internal. The process Run_A has alphabet A, and it is always ready to perform any event from the set A. Processes may also be recursively defined, by giving equations which contain the name of the process being defined as a subterm of the process expression. For example, the process

$$\mathit{Copy} = \mathit{in}?x : \mathbb{N} \rightarrow \mathit{out}!x \rightarrow \mathit{Copy}$$

defines a buffer process *Copy* as one which repeatedly alternates input and output. Indexed processes may also be mutually recursively defined using families of equations.

The semantics of processes are given in terms of observations. A process is identified with the set of behaviours that may possibly be observed of it, where the kind of behaviour considered determines the nature of the model.

The *Traces Model* is concerned with the traces of CSP processes: the (finite) sequences of events that they can perform during some execution. A trace is written as a finite sequence of events within angled brackets. The empty trace is denoted $\langle\rangle$, and $tr_1 \frown tr_2$ is the concatenation of traces tr_1 and tr_2. For example,

$$\begin{aligned}
&\mathit{traces}(\mathit{Copy}) = \\
&\{\langle\rangle, \} \\
&\cup \{\langle \mathit{in}.v\rangle \mid v \in V\} \\
&\cup \{\langle \mathit{in}.v, \mathit{out}.v\rangle \mid v \in V\} \\
&\cup \{\langle \mathit{in}.v, \mathit{out}.v, \mathit{in}.w\rangle \mid v \in V, w \in W\} \\
&\vdots
\end{aligned}$$

where V is the type of the channels *in*, *out*. The Traces Model and trace semantics for the CSP operators introduced here is included in Appendix A.

The traces model is sufficient for the purposes of this paper. More sophis-

(i) $(P \underset{A}{\parallel} Q) \underset{B}{\parallel} Stop = (P \underset{B}{\parallel} Stop) \underset{A}{\parallel} (Q \underset{B}{\parallel} Stop)$

(ii) $\alpha(P) \cap B = \varnothing \Rightarrow (P \underset{A}{\parallel} (Q \underset{B}{\parallel} Stop)) = P \underset{A \cup B}{\parallel} Q$

(iii) $P \setminus (A \cup B) = (P \setminus A) \setminus B$

(iv) $(P \underset{A}{\parallel} Stop) = (P \underset{A}{\parallel} Stop) \setminus A$

(v) $A \cap B = \varnothing \Rightarrow (P \underset{A}{\parallel} Q) \setminus B = (P \underset{B}{\parallel} Stop) \underset{A}{\parallel} (Q \underset{B}{\parallel} Stop)$

(vi) $P \setminus A \sqsubseteq P \underset{A}{\parallel} Stop$

(vii) $(P \setminus B) \underset{A}{\parallel} Stop \sqsubseteq (P \underset{A}{\parallel} Stop) \setminus B$

(viii) $(P \setminus B) \underset{A}{\parallel} (Q \setminus B) \sqsubseteq (P \underset{A}{\parallel} Q) \setminus B$

(ix) $(P \setminus B) \,|||\, (Q \setminus B) = (P \,|||\, Q) \setminus B$

(x) $P \setminus (A \cup B) \sqsubseteq (P \underset{A}{\parallel} Stop) \setminus B$

(xi) $\alpha(P) \cap B = \varnothing \Rightarrow (P \,|||\, Run_B) \underset{A \cup B}{\parallel} Q = P \underset{A}{\parallel} Q$

(xii) $P \sqsubseteq Q \wedge Q \sqsubseteq P \Rightarrow P = Q$

Fig. 1. Some algebraic laws of the CSP traces model

ticated models are used to handle phenomena such as deadlock, divergence, and nondeterminism.

Processes are considered equivalent in a semantic model if they have the same set of behaviours in that model. Thus if P and Q have the same traces, then they are equivalent in the traces model, written $P =_{traces} Q$. This means that if only their traces are examined, then they cannot be distinguished. A process P is refined by a process Q in the traces model if all the traces of Q are also traces of P. This is written $P \sqsubseteq_{traces} Q$. All of the CSP operators are *monotonic* with respect to refinement: refinement of components in a CSP combination will result in a refinement of the combination. In this paper we are concerned only with the traces model, and so we will drop the subscript from the equality symbol and the refinement symbol. The traces model supports a number of algebraic equations and refinement relationships. Those used in the proofs later in this paper are given in Figure 1.

One approach to comparing processes is in terms of *testing*. A test T is a particular kind of process, with some *Success* states. We consider the execution of T in conjunction with a process P and if $(P \underset{\Sigma}{\parallel} T) \setminus \Sigma$ can reach a success state then we say that 'P **may** T'. If P **may** T whenever Q **may** T and vice versa for all possible tests T, then P and Q are equivalent under may testing. It turns out that this will be the case precisely when P and Q have the same traces. For further information on testing see [5].

Finally, for the purposes of this paper it will be useful to characterise in

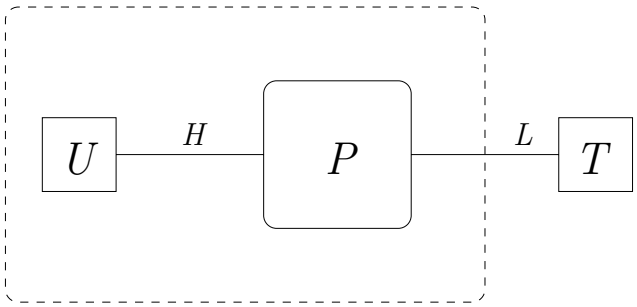

Fig. 2. Testing U through P

the traces model what is meant by a process always being willing to accept any event from a set A:

Definition 2.1 A process P is *open on* A if

$$\forall\, tr \in traces(P), a \in A \bullet tr \frown \langle a \rangle \in traces(P)$$

3 Non-interference and may testing

We are concerned with a system P whose interface is partitioned into a set of high level events H and a set of low level events L. The intention is that observing or interacting with the system only at the low level should not provide information about what is going on at the high level. In this paper, we adopt the convention that all pictures in the figures of systems P with partitioned interfaces will have high level events to the left, and low level events to the right of the process, as for P in Figure 2.

In this paper, non-interference will be considered from the point of view of whether a low level agent interacting with P may distinguish between two arbitrary processes U_1 and U_2 which are interacting with P at the high level. The high level user might or might not be able to observe low level activity directly, and we will consider both of these possibilities. In fact they turn out to be equivalent for may testing.

The first definition considers the high level agents to be concerned only with high level events. This is pictured in Figure 2. In defining a notion of non-interference on a process P with respect to a high level interface $H \subseteq \Sigma$ and a low level interface $L \subseteq \Sigma$, we will assume that $H \cap L = \varnothing$ and $\alpha(P) \subseteq H \cup L$. In this case, we make the following definition:

Definition 3.1 P is *may non-interfering with respect to (H,L)*, or *may-NI wrt (H,L)*, if for any test T, and any two high level users U_1 and U_2 :

$$(U_1 \underset{H}{\parallel} P) \setminus H \textbf{ may } T \Leftrightarrow (U_2 \underset{H}{\parallel} P) \setminus H \textbf{ may } T$$

An alternative definition is concerned with the situation in which high level agents are able to observe low level events, as long as they cannot prevent low level events from occurring. This condition is necessary since otherwise high

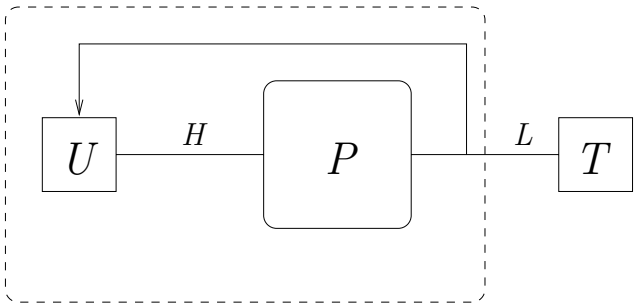

Fig. 3. Strong testing U through P: U can observe low level events

level agents could be distinguished by a low level test through no fault of P. For example, the test $l \rightarrow \mathit{Success}$ could tell the difference between two high level processes $U_1 = l \rightarrow \mathit{Stop}$ and $U_2 = \mathit{Stop}$ through system $P = l \rightarrow \mathit{Stop}$, since U_2 blocks l.

Definition 3.2 P is *strong may non-interfering with respect to (H,L)* if for any test T, and any two high level users U_1 and U_2 which are open on L:

$$(U_1 \underset{H \cup L}{\|} P) \setminus H \textbf{ may } T \Leftrightarrow (U_2 \underset{H \cup L}{\|} P) \setminus H \textbf{ may } T$$

In fact as we shall see, the two definitions on a system P are equivalent for may testing, although they correspond to different situations.

These definitions state that P will prevent interference if no low level activity can ever distinguish between two different high level *processes*: if some low level activity is possible for one, then it is also possible for the other.

A key point is that high level *users* should be indistinguishable, rather than high level *activity*. This means that some possible high level traces can still be ruled out on observing low level behaviour, even if high level processes cannot be distinguished.

Example 3.3 The process

$$P_1 = (l \rightarrow \mathit{Stop} \,\Box\, h \rightarrow \mathit{Stop})$$

is may-NI wrt $(\{h\}, \{l\})$. Any low level view is going to see either an l or just the empty trace; and the possibility of the l cannot be prevented by any high level process. Thus no low level test will distinguish any two high level processes. On the other hand, observation of l informs us that h cannot have occurred in that same execution, so it does convey some information about the particular high level activity (i.e. that h has not occurred). Conversely, observation of the empty trace does not convey any information about the high level activity.

Example 3.4 The process

$$P_2 = h \rightarrow l \rightarrow \mathit{Stop}$$

is not may-NI wrt $(\{h\}, \{l\})$. In this case, there are two users, $h \rightarrow Stop$ and $Stop$, which can be distinguished by the test $l \rightarrow Stop$. If h is required for l to be possible, then observation of l allows the deduction that h must have occurred, and hence that the high level user was able to perform h.

Thus if high level activity is required for particular low level activity, then this definition states that there is interference. But if high level activity leads only to absence of or reduced low level activity (or equivalently that absence of high level activity is required for low level activity) then this definition allows that there is no interference. Any high level user is able to engage in absence of high level activity! This reflects the fact that, in the traces model, $U = U \sqcap Stop$.

Thus identifying a capacity for not performing high level events does not distinguish high level processes, since all high level users have that capacity.

3.1 Characterisation in the traces model

It turns out that may non-interference coincides with the property of non-inference proposed by O'Halloran [11]: that if $tr \in traces(P)$ then $tr \setminus H \in traces(P)$. This may also be expressed as the following equation on P: $(Stop \underset{H}{\parallel} P) = (P \setminus H)$.

Theorem 3.5 *P is may-NI wrt (H, L) if and only if $P \setminus H = P \underset{H}{\parallel} Stop$ in the traces model.*

Proof. It is a standard result that $\forall\, T \bullet P$ **may** $T \Leftrightarrow Q$ **may** T exactly characterises traces equivalence: $P =_{traces} Q$.

If P is may-NI wrt (H, L), then given any two users U_1 and U_2,

$$(U_1 \underset{H}{\parallel} P) \setminus H \textbf{ may } T \Leftrightarrow (U_2 \underset{H}{\parallel} P) \setminus H \textbf{ may } T$$

for any test T, and so $(U_1 \underset{H}{\parallel} P) \setminus H = (U_2 \underset{H}{\parallel} P) \setminus H$. This is therefore true for the two particular high level users $Stop$ and Run_H, and so

$$(Stop \underset{H}{\parallel} P) \setminus H = (Run_H \underset{H}{\parallel} P) \setminus H$$

which simplifies to

$$P \underset{H}{\parallel} Stop = P \setminus H$$

Conversely, suppose that $Stop \underset{H}{\parallel} P = P \setminus H$. Then any high level user U has that

$$Run_H \sqsubseteq U \sqsubseteq Stop$$

It follows by monotonicity of the CSP operators that

$$(Run_H \underset{H}{\parallel} P) \setminus H \sqsubseteq (U \underset{H}{\parallel} P) \setminus H \sqsubseteq (Stop \underset{H}{\parallel} P) \setminus H$$

and hence that

$$P \setminus H \sqsubseteq (U \underset{H}{\parallel} P) \setminus H \sqsubseteq Stop \underset{H}{\parallel} P$$

Since $P \setminus H = P \underset{H}{\parallel} Stop$ it follows that all three processes are equal for arbitrary U. Thus all high level users in parallel with P present the same low level view, and so

$$(U_1 \underset{H}{\parallel} P) \setminus H = (U_2 \underset{H}{\parallel} P) \setminus H$$

and so they may pass the same tests T. Thus P is may-NI wrt (H, L). □

Corollary 3.6 *P is may-NI wrt (H, L) if and only if*

$$(U_1 \underset{H}{\parallel} P) \setminus H = (U_2 \underset{H}{\parallel} P) \setminus H$$

for any two high level processes U_1 and U_2.

The same characterisation also holds for strong may non-interference with respect to (H, L):

Theorem 3.7 *P is strongly may non-interfering with respect to (H, L) if and only if*

$$P \setminus H = P \underset{H}{\parallel} Stop$$

Proof. The proof is similar to that for may non-interference with respect to (H, L).

Firstly, if P is strongly may non-interfering wrt (H, L), then the high level users (open on L) $Run_{H \cup L}$ and Run_L cannot be distinguished, so $(Run_{H \cup L} \underset{H \cup L}{\parallel} P) \setminus H$ and $(Run_L \underset{H \cup L}{\parallel} P) \setminus H$ have the same traces. These reduce to $P \setminus H$ and $P \underset{H}{\parallel} Stop$ respectively, so we obtain $P \setminus H = P \underset{H}{\parallel} Stop$.

Conversely, suppose that $P \setminus H$ and $P \underset{H}{\parallel} Stop$ have the same traces.

For any process U open on L and with alphabet $H \cup L$ we have that

$$Run_{H \cup L} \sqsubseteq U \sqsubseteq Run_L$$

Hence by monotonicity

$$(Run_{H \cup L} \underset{H \cup L}{\|} P) \setminus H \sqsubseteq (U \underset{H \cup L}{\|} P) \setminus H \sqsubseteq (Run_L \underset{H \cup L}{\|} P) \setminus H$$

i.e.

$$P \setminus H \sqsubseteq (U \underset{H \cup L}{\|} P) \setminus H \sqsubseteq P \underset{H}{\|} Stop$$

and thus

$$P \setminus H = (U \underset{H \cup L}{\|} P) \setminus H = P \underset{H}{\|} Stop$$

Since this is true for any U open on L, it follows that no low level test can distinguish any two high level processes. □

Corollary 3.8 *Strong may non-interference with respect to* (H, L) *and may non-interference with respect to* (H, L) *are equivalent on any process.*

As a result of this corollary, we need only investigate one of these forms of non-interference for the rest of this paper; we will use may non-interference.

3.2 Signals and high level outputs

The above characterisation of may non-interference assumes a framework in which all high level events are synchronisations between the system and the high level user, and thus that the high level user has a veto over all such events. Low level evidence that such events have occurred thus indeed provide information about the activity of the high level user.

However, there are situations where high level events are required to pass information from the system to the high level process, but where the high level process cannot prevent such events from occurring. Examples include high level signals (such as writing to a screen) or outputs (in I/O automata). They cannot in themselves yield any information about high level user activity, yet their occurrence can be deduced from the low level events. In this paper, signals are another term for non-blockable output events.

For example, a high level output *ho* might precede low level activity:

$$P = ho \rightarrow l \rightarrow Stop$$

If *ho* cannot be refused by the high level, then the occurrence of l gives no information about the high level process. Yet the above characterisation of may-NI fails: $\langle l \rangle$ is possible for $P \setminus \{ho\}$ but not for $P \underset{\{ho\}}{\|} Stop$.

For a treatment which allows for high level signals, we must split the high level interface H into two disjoint sets: signals HO, and synchronisations HI. In this case, some restrictions must be introduced to the kind of high level user U_1 and U_2 that should be indistinguishable by low level may testing. In

particular, they should always be able to accept any high level output event HO. Thus U_1 and U_2 must be open on HO.

In the presence of high level signals, we enhance the interface information carried in the non-interference property.

Definition 3.9 P is *may non-interfering with respect to ((HI,HO),L)*, or *may-NI wrt ((HI, HO), L)*, if for any test T, and any two high level users U_1 and U_2 open on HO:

$$(U_1 \underset{H}{\parallel} P) \setminus H \textbf{ may } T \Leftrightarrow (U_2 \underset{H}{\parallel} P) \setminus H \textbf{ may } T$$

Observe that 'may-NI wrt $((H, \varnothing), L)$' is the same as 'may-NI wrt (H, L)'.

Theorem 3.10 *P is may-NI wrt $((HI, HO), L)$ if and only if*

$$(Stop \underset{HI}{\parallel} P) \setminus HO = P \setminus H$$

Proof. If P is may-NI wrt $((HI, HO), L)$, then the two HO-open high level users Run_{HO} and Run_H have that

$$(Run_{HO} \underset{H}{\parallel} P) \setminus H \textbf{ may } T \Leftrightarrow (Run_H \underset{H}{\parallel} P) \setminus H \textbf{ may } T$$

for any T, and so

$$(Stop \underset{HI}{\parallel} P) \setminus HO = P \setminus H$$

This means that if tr is a trace of P, then there is a trace of P with the same low level presentation, no high level inputs, and possibly different high level outputs: if $tr \in traces(P)$ then $tr \setminus H \in traces(P \setminus H)$ and so $tr \setminus H \in traces((Stop \underset{HI}{\parallel} P) \setminus HO)$ and so $tr \setminus H \in traces(P \setminus HO)$.

Conversely, assume $(P \underset{HI}{\parallel} Stop) \setminus HO = P \setminus H$. Any HO-open user U must have all traces of Run_{HO} as possible traces. Thus

$$traces(Run_H) \sqsubseteq traces(U) \sqsubseteq traces(Run_{HO})$$

and so by monotonicity

$$\begin{aligned} P \setminus H &= (Run_H \underset{H}{\parallel} P) \setminus H \\ &\sqsubseteq (U \underset{H}{\parallel} P) \setminus H \\ &\sqsubseteq (Run_{HO} \underset{H}{\parallel} P) \setminus H = (P \underset{HI}{\parallel} Stop) \setminus HO \end{aligned}$$

and thus

$$(U_1 \underset{H}{\parallel} P) \setminus H = (U_2 \underset{H}{\parallel} P) \setminus H$$

for any two HO-open users U_1 and U_2—so they may pass exactly the same tests. □

In principle, the low level interface L could also be divided into low level signals LO and low level synchronisations LI, with tests restricted to those that can always accept events from LO. In fact this makes no difference—this more restricted set of tests is as discriminating, since any test T that distinguishes two high level users can be transformed into a LO open test $T \mid\mid\mid Run_{LO}$ that also distinguishes them (since the component Run_{LO} does not introduce any more success states.)

4 Examples

In this section we consider some examples that illustrate various aspects of may non-interference.

Consider the following one-place buffer inputting on a channel l and outputting on a channel h:

$$B1 = l?x : T \rightarrow h!x \rightarrow B1$$

If h is a high level signal, then $B1$ has complete control over when messages on h are sent, and so $B1$ exhibits may-NI. In particular, $B1$ is may-NI wrt $((\{\}, \{h\}), \{l\})$.

On the other hand, if h is a high level synchronisation which can be blocked by the high level environment of $B1$, then $B1$ is not may non-interference. In other words, it is not may-NI wrt $(\{h\}, \{l\})$. In particular, the occurrence of a second low-level message indicates that the high level user has accepted a high level input. In testing terms, a low level test $l!0 \rightarrow l!0 \rightarrow Success$ can distinguish the high level user $Stop$ from $h?x : T \rightarrow Stop$. In terms of the the characterising trace equivalence, the process $B1 \setminus h.T = Run_{l.T}$, whereas $B1 \underset{h.T}{\parallel} Stop = l?x : T \rightarrow Stop$. Thus $B1 \setminus h.T \neq B_1 \underset{h.T}{\parallel} Stop$: $B1$ does not meet the characterising equation for may-NI.

In order to regain may-NI in the case where h is not a signal, it is necessary to allow further inputs along l even if output on h has not yet occurred. One way of achieving this is to give the buffer infinite capacity:

$$\begin{aligned} B2 &= B2_{\langle\rangle} \\ B2_{\langle\rangle} &= l?x : T \rightarrow B2_{\langle x \rangle} \\ B2_{\langle v \rangle^\frown s} &= l?x : T \rightarrow B2_{\langle v \rangle^\frown s^\frown \langle x \rangle} \\ &\quad \Box\ h!v \rightarrow B2_s \end{aligned}$$

This process reliably passes all messages from l to h, and is may-NI wrt $(\{h\}, \{l\})$. Any test at the low level cannot tell whether a high level user has picked up any messages or not.

However, if the buffer is restricted to any finite capacity, so that further inputs are blocked when the buffer is full, then may-NI is lost.

To make a finite-capacity buffer may-NI, it is necessary to allow for some messages to be lost when inputs are received into a full buffer, since inputs must always be possible. For example, the following one-place buffer allows

its contents to be overwritten:

$$\begin{aligned} B3 &= l?x : T \rightarrow B3(x) \\ B3(x) &= l?y : T \rightarrow B3(y) \\ &\quad \Box\ h!x \rightarrow B3 \end{aligned}$$

This process is may-NI on $(\{h\}, \{l\})$, but at the cost of possibly losing messages. Worse, a high level user cannot tell from the messages passed along h whether any messages have been lost.

This latter problem can to some extent be addressed by including additional information with the high level output. For example, the number of messages that have been lost since the last high output might be of use to the high level process:

$$\begin{aligned} B4 &= l?x : T \rightarrow B4(x, 0) \\ B4(x, n) &= l?y : T \rightarrow B4(y, n+1) \\ &\quad \Box\ h!(x.n) \rightarrow B4 \end{aligned}$$

The type of channel h is $T \times \mathbb{N}$. In this example, messages can still be lost, but the high level user will at least know how many (if any) between any two messages. $B4$ is also may-NI wrt $(\{h\}, \{l\})$.

5 Composability

The characterisation of may non-interference allows us to examine which CSP composition operators preserve it. In the non-interference literature, systems are generally composed so as to preserve the classification level of events (see e.g. [9,19]). In this paper we consider such compositions, but we go further and also consider compositions in which parts of one process' high level interface are connected to parts of another's low level interface. Thus each process P will be associated with its own classification of events from its interface into high level events $H_P = HI_P \cup HO_P$, and low level events L_P.

5.1 Interleaving

Two processes are interleaved if they run side by side without any direct interaction. This situation is pictured in Figure 4. The sets H_P and H_Q can overlap, as can L_P and L_Q. However, high and low level sets must be disjoint from each other: $(H_P \cup H_Q) \cap (L_P \cup L_Q) = \varnothing$).

Theorem 5.1 *If P is may-NI wrt (H_P, L_P) and Q is may-NI wrt (H_Q, L_Q), then $P \,|||\, Q$ is may-NI wrt $(H_P \cup H_Q, L_P \cup L_Q)$.*

Proof. We show that $(P \,|||\, Q) \setminus H = (P \,|||\, Q) \underset{H}{\parallel} \mathit{Stop}$.

Let $H = H_P \cup H_Q$.

$$\begin{aligned} (P \,|||\, Q) \setminus H &= (P \setminus H) \,|||\, (Q \setminus H) \\ &= (P \setminus H_P) \,|||\, (Q \setminus H_Q) \end{aligned}$$

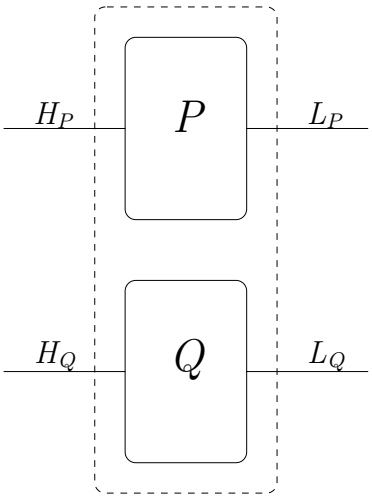

Fig. 4. P and Q side by side

$$
\begin{aligned}
&= (P \underset{H_P}{\parallel} \mathit{Stop}) \;|||\; (Q \underset{H_Q}{\parallel} \mathit{Stop}) \\
&= (P \underset{H}{\parallel} \mathit{Stop}) \;|||\; (Q \underset{H}{\parallel} \mathit{Stop}) \\
&= (P \;|||\; Q) \underset{H}{\parallel} \mathit{Stop}
\end{aligned}
$$

□

The more general case additionally considers high level signals. It requires the additional property that high level events must not be treated as signals by one process and synchronisations by the other.

Theorem 5.2 *If P is may-NI wrt $((HI_P, HO_P), L_P)$ and Q is may-NI wrt $((HI_Q, HO_Q), L_Q)$, and $(HI_P \cup HI_Q) \cap (HO_P \cup HO_Q) = \varnothing$, then $P \;|||\; Q$ is may-NI wrt $((HI_P \cup HI_Q, HO_P \cup HO_Q), L_P \cup L_Q)$.*

Proof. Let $H_P = HI_P \cup HO_P$, and $H_Q = HI_Q \cup HO_Q$.

We show that $(P \;|||\; Q) \setminus H_P \cup H_Q = (P \;|||\; Q) \underset{HI_P \cup HI_Q}{\parallel} \mathit{Stop} \setminus (HO_P \cup HO_Q)$.

$$
\begin{aligned}
(P \;|||\; Q) \setminus H_P \cup H_Q &= (P \setminus H_P) \;|||\; (Q \setminus H_Q) \\
&= (P \underset{HI_P}{\parallel} \mathit{Stop}) \setminus HO_P \;|||\; (Q \underset{HI_Q}{\parallel} \mathit{Stop}) \setminus HO_Q \\
&= ((P \;|||\; Q) \underset{HI_P \cup HI_Q}{\parallel} \mathit{Stop}) \setminus HO_P \cup HO_Q
\end{aligned}
$$

as required. The last step used the condition that $(HI_P \cup HI_Q) \cap (HO_P \cup HO_Q) = \varnothing$. □

5.2 Chaining

In a chaining composition $P \gg Q$, the low level interface of one process is connected to the high level interface of the other. Their resulting common

interface is then hidden. The situation is pictured in Figure 5. The sets H, M, and L are pairwise disjoint. P has alphabet $H \cup M$ with H as high level events and M as low level, and Q has alphabet $M \cup L$ with M as its high level events and L as low level. The chain $P \gg Q$ is defined to be $(P \underset{M}{\parallel} Q) \setminus M$.

In this situation, it is in fact sufficient for either one of these processes to be may-NI for their combination to be may-NI.

Theorem 5.3 *If P is may-NI wrt (H, M), or Q is may-NI wrt (M, L), then $P \gg Q$ is may-NI wrt (H, L).*

Proof. We are concerned with the may-NI wrt (H, L) properties of $(P \underset{M}{\parallel} Q) \setminus M$.

$$\begin{aligned}((P \underset{M}{\parallel} Q) \setminus M) \setminus H &= ((P \setminus H) \underset{M}{\parallel} Q) \setminus M \\ &= ((P \underset{H}{\parallel} \mathit{Stop}) \underset{M}{\parallel} Q) \setminus M \\ &= ((P \underset{M}{\parallel} Q) \setminus M) \underset{H}{\parallel} \mathit{Stop}\end{aligned}$$

The step

$$((P \setminus H) \underset{M}{\parallel} Q) \setminus M = ((P \underset{H}{\parallel} \mathit{Stop}) \underset{M}{\parallel} Q) \setminus M$$

is justified in one of two ways, depending on whether it is P or Q that is may-NI:

- If it is P, then $(P \setminus H) = (P \underset{H}{\parallel} \mathit{Stop})$ and the result follows.
- If it is Q, then $(U \underset{M}{\parallel} Q) \setminus M = (U' \underset{M}{\parallel} Q) \setminus M$ for any U and U' by Corollary 3.6, and the step provides a particular instantiation.

In the presence of signals, where $H = HI \cup HO$ as before, if P is may-NI on $((HI, HO), M)$, then $P \gg Q$ is may-NI on $((HI, HO), L)$:

$$\begin{aligned}(P \underset{M}{\parallel} Q) \setminus M \setminus H &= (P \setminus H) \underset{H}{\parallel} Q \setminus M \\ &= ((P \underset{HI}{\parallel} \mathit{Stop}) \setminus HO \underset{M}{\parallel} Q) \setminus M \\ &= ((P \underset{M}{\parallel} Q) \setminus M \underset{HI}{\parallel} \mathit{Stop}) \setminus HO\end{aligned}$$

Furthermore, if Q is may-NI wrt $((MI, MO), L)$ (where $M = MI \cup MO$), and P is open on MO, then $P \gg Q$ is may-NI on (H, L) for any H for which $\alpha(P) \subseteq H \cup M$ and $H \cap M = \varnothing$. The following reasoning establishes this:

$$\begin{aligned}(P \underset{M}{\parallel} Q) \setminus M \setminus H &= ((P \setminus H) \underset{M}{\parallel} Q) \setminus M \\ &= ((P \underset{H}{\parallel} \mathit{Stop}) \underset{M}{\parallel} Q) \setminus M \quad \text{since } Q \text{ is may-NI} \\ &= (P \underset{M}{\parallel} Q) \setminus M \underset{H}{\parallel} \mathit{Stop}\end{aligned}$$

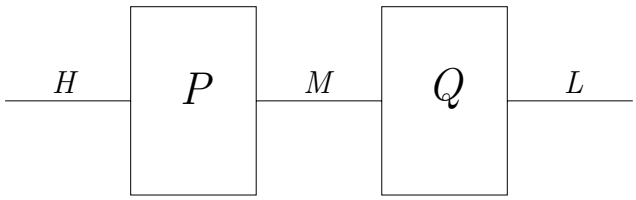

Fig. 5. P and Q in a chain

since if P is open on MO then so too is $P \setminus H$ and $P \underset{H}{\parallel} \mathit{Stop}$.

However, if P is not open on MO then $P \gg Q$ need not be may-NI wrt (H, L), even if Q is. For example

$$P = h \rightarrow mo \rightarrow \mathit{Stop}$$
$$Q = mo \rightarrow l \rightarrow \mathit{Stop}$$

Here, Q is may-NI wrt $((\varnothing, \{mo\}), \{l\})$. However, the fact that P can block mo until after h occurs allows information to flow from high to low. In this case $P \gg Q = h \rightarrow l \rightarrow \mathit{Stop}$, which is not may-NI wrt (H, L), as observed in Example 3.4.

Of course, if P is also may-NI then we have previously shown that $P \gg Q$ must be, so any counterexample must make use of some P which is not itself may-NI. □

5.3 Synchronisation

In the general case, P and Q can synchronise on both high level and low level events. Let $H = H_P \cup H_Q$ and $L = L_P \cup L_Q$ with $H \cap L = \varnothing$. Observe that H_P and H_Q can overlap, as can L_P and L_Q. Let $I \subseteq H \cup L$ be any synchronisation set between P and Q. Then P in parallel with Q on this set will be may-NI whenever P and Q both are.

Theorem 5.4 *If P is may-NI wrt (H_P, L_P) and Q is may-NI wrt (H_Q, L_Q), then $P \underset{I}{\parallel} Q$ is may-NI wrt $(H_P \cup H_Q, L_P \cup L_Q)$.*

Proof.

$$\begin{aligned}
(P \underset{I}{\parallel} Q) \setminus H_P \cup H_Q &\sqsupseteq (P \setminus H_P \cup H_Q) \underset{I}{\parallel} (Q \setminus H_P \cup H_Q) \\
&= (P \setminus H_P) \underset{I}{\parallel} (Q \setminus H_Q) \\
&= (P \underset{H_P}{\parallel} \mathit{Stop}) \underset{I}{\parallel} (Q \underset{H_Q}{\parallel} \mathit{Stop}) \\
&= (P \underset{H_P \cup H_Q}{\parallel} \mathit{Stop}) \underset{I}{\parallel} (Q \underset{H_P \cup H_Q}{\parallel} \mathit{Stop}) \\
&= (P \underset{I}{\parallel} Q) \underset{H_P \cup H_Q}{\parallel} \mathit{Stop}
\end{aligned}$$

But $(P \underset{I}{\parallel} Q) \underset{H_P \cup H_Q}{\parallel} \mathit{Stop} \sqsubseteq (P \underset{I}{\parallel} Q) \setminus H_P \cup H_Q$, so they must be equal. □

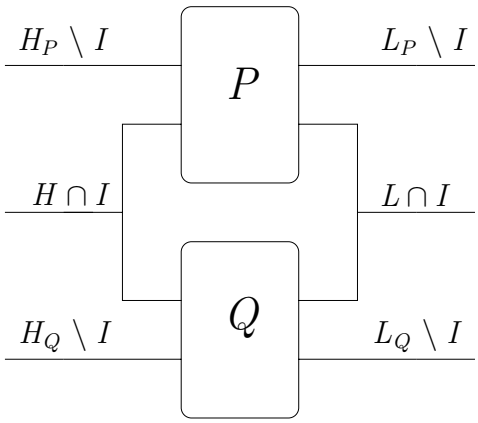

Fig. 6. Some synchronisation between P and Q

This composition covers a variety of cases. In general, P and Q may synchronise on some events and interleave on others. They may have events in common on which they do not synchronise: interleaving (above) is a special case where $I = \varnothing$.

When the component processes allow high level signals, the composition preserves may-NI provided none of the signals are involved in any synchronisation. In this case, $H \cap I$ must be a subset of $HI = HI_P \cup HI_Q$, and we obtain the following theorem (in which $HO = HO_P \cup HO_Q$):

Theorem 5.5 *If P is may-NI wrt $((HI_P, HO_P), L_P)$ and Q is may-NI wrt $((HI_Q, HO_Q), L_Q)$, and $I \cap (HO_P \cup HO_Q) = \varnothing$ then $P \underset{I}{\parallel} Q$ is may-NI wrt $((HI_P \cup HI_Q, HO_P \cup HO_Q), L_P \cup L_Q)$.*

Proof. We compare $(P \underset{I}{\parallel} Q) \setminus H$ and $((P \underset{I}{\parallel} Q) \underset{HI_P \cup HI_Q}{\parallel} Stop) \setminus HO_P \cup HO_Q$ and show that each is a refinement of the other.

$$
\begin{aligned}
&((P \underset{I}{\parallel} Q) \underset{HI_P \cup HI_Q}{\parallel} Stop) \setminus HO_P \cup HO_Q \\
&= ((P \underset{HI_P \cup HI_Q}{\parallel} Stop) \underset{I}{\parallel} (Q \underset{HI_P \cup HI_Q}{\parallel} Stop)) \setminus HO_P \cup HO_Q \\
&= ((P \underset{HI_P}{\parallel} Stop) \underset{I}{\parallel} (Q \underset{HI_Q}{\parallel} Stop)) \setminus HO_P \cup HO_Q \\
&= (P \underset{HI_P}{\parallel} Stop) \setminus HO_P \cup HO_Q \underset{I}{\parallel} (Q \underset{HI_Q}{\parallel} Stop) \setminus HO_P \cup HO_Q \\
&\qquad \text{since } I \cap (HO_P \cup HO_Q) = \varnothing \\
&= (P \setminus H_P) \underset{I}{\parallel} (Q \setminus H_Q) \\
&= (P \setminus H_P \cup H_Q) \underset{I}{\parallel} (Q \setminus H_P \cup H_Q) \\
&\sqsubseteq (P \underset{I}{\parallel} Q) \setminus H_P \cup H_Q
\end{aligned}
$$

but we also have that

$$(P \underset{I}{\parallel} Q) \setminus HI_P \cup HI_Q \cup HO_P \cup HO_Q$$
$$\sqsubseteq ((P \underset{I}{\parallel} Q) \underset{HI_P \cup HI_Q}{\parallel} Stop) \setminus HO_P \cup HO_Q$$

and hence the two sides are equal, and $P \underset{I}{\parallel} Q$ is may-NI with respect to $((HI_P \cup HI_Q, HO_P \cup HO_Q), L_P \cup L_Q)$. □

However, the result does not hold when the system allows synchronisation on high level signals.

Example 5.6 For example, the following processes have that P is may-NI wrt $((\varnothing, \{ho\}), \{l\})$ and Q is may-NI wrt $((\{hi\}, \{ho\}), \varnothing)$:

$$P = ho \rightarrow l \rightarrow Stop$$
$$Q = hi \rightarrow ho \rightarrow Stop$$

When these two processes are required to synchronise on $\{ho\}$, then the resulting process behaves as $hi \rightarrow ho \rightarrow l \rightarrow Stop$ which can perform l only after the occurrence of the high level input hi, allowing information to flow from high to low.

5.4 Restricted synchronisation

Two special cases of synchronisation obtain when a process Q restricts the behaviour of a process P which is may-NI wrt (H_P, L_P), on just one of its interfaces. These two special cases are pictured in Figure 7.

Restriction of the high level behaviour of P is an example of synchronisation in which $\alpha(Q) = I \subseteq H_P$. It immediately follows that Q is may-NI wrt $(I, \varnothing)$, whatever its definition, since it can perform only high level events. Hence $P \underset{I}{\parallel} Q$ will be may-NI wrt (H_P, L_P) whenever P is. Introducing restrictions to the high level behaviour of P preserves non-interference.

However, the result for this special case does not hold in general in the presence of high level signals, unless there is no synchronisation on them, as was the case for synchronisation composition given above. The processes given in Example 5.6 provide a counterexample here as well.

Similarly, introducing restrictions to the low level behaviour of P also preserves non-interference. If $\alpha(Q) = I \subseteq L_P$, then Q is may-NI wrt $(\varnothing, I)$, and so $P \underset{I}{\parallel} Q$ will be may-NI wrt (H_P, L_P) whenever P is. This is also true in the presence of high level signals, since in this case Q is only concerned with low level events, and so there can be no synchronisation on high level signals. Hence Theorem 5.4 is applicable here.

5.5 Hiding

The hiding operator abstracts some of the channels in a process interface. In general, these can be both high and low level. This abstraction will always

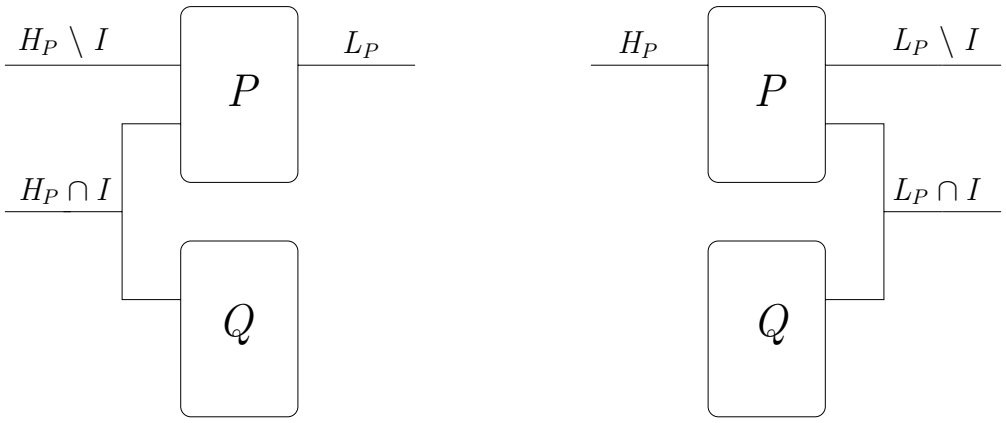

Fig. 7. Restricting P on high and low interfaces

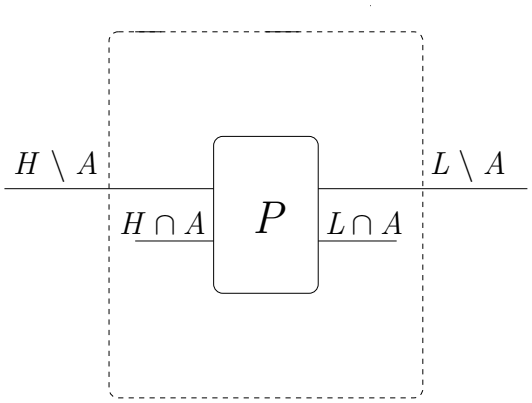

Fig. 8. Hiding inputs and outputs

preserve may-NI: by restricting the interface through which high level users can interact with P, and by restricting the interface through which low level tests can be carried out. This is pictured in Figure 8, where a set A of high and low level events is hidden.

Theorem 5.7 *If P is may-NI wrt (H, L), then so too is $P \setminus A$.*

Proof. The proof is straightforward.

$$\begin{aligned}(P \setminus A) \setminus H &= (P \setminus H) \setminus A \\ &= (P \underset{H}{\parallel} \mathit{Stop}) \setminus A \\ &\sqsupseteq (P \setminus A) \underset{H}{\parallel} \mathit{Stop}\end{aligned}$$

But also $(P \setminus A) \setminus H \sqsubseteq (P \setminus A) \underset{H}{\parallel} \mathit{Stop}$, and so the two are equal.

If P allows high level signals, then the result still holds, and the proof is as follows:

$$((P \setminus A) \underset{HI}{\parallel} \mathit{Stop}) \setminus HO \sqsubseteq (\mathit{Stop} \underset{HI}{\parallel} P) \setminus A \setminus HO$$

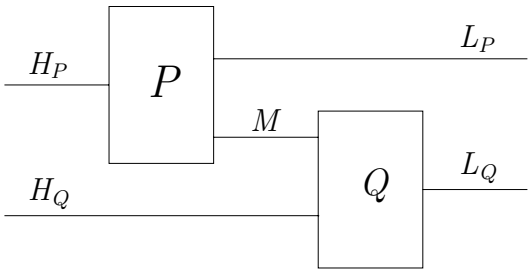

Fig. 9. P and Q in a high-low connection

$$\begin{aligned}&= (Stop \underset{HI}{\parallel} P) \setminus HO \setminus A\\ &= (P \setminus H) \setminus A\\ &= (P \setminus A) \setminus H\end{aligned}$$

But $P \setminus A \setminus H \sqsubseteq (Stop \underset{HI}{\parallel} (P \setminus A)) \setminus HO$ and so the two are equal. □

5.6 High-low connection

If some of the low level outputs of P are provided as some high level inputs to Q, then may-NI is not preserved in general. The situation is pictured in Figure 9. High-low connection allows some low level output from P as input into Q (with such channels hidden), but with some other low outputs from P remaining available to the low level (unlike chaining, a special case which we have already seen does preserve may-NI).

Example 5.8 The following processes provide an example of a pair of may-NI processes whose composition is not may-NI:

$$P = m \rightarrow l \rightarrow Stop$$
$$Q = h \rightarrow m \rightarrow Stop$$

P is may-NI wrt $(\varnothing, \{m, l\})$, since it has only low level events. Q is may-NI wrt $(\{h, m\}, \varnothing)$, since it has only high level events.

However, $(P \underset{\{m\}}{\parallel} Q) \setminus \{m\} = h \rightarrow l \rightarrow Stop$, which is not may-NI with respect to any (H, L) for which $h \in H$ and $l \in L$.

Observe that $P \underset{\{m\}}{\parallel} Q$ is not may-NI either—the visibility of $\{m\}$ makes no difference (either as a high level set, a low level set, or (for larger interface sets) divided across the two levels of the resulting process).

Although high-low connection in the general case does not preserve may-NI, this is not altogether surprising since it allows in some sense a downgrading of information from high to low.

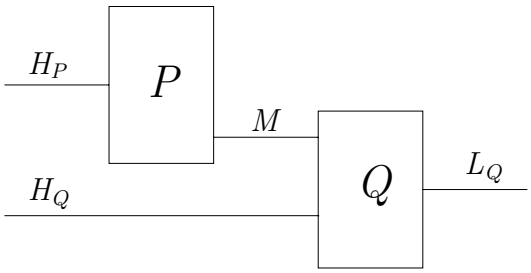

Fig. 10. P connected to Q's high level interface

5.7 Limited high-low connection 1

If P does not have any low level interface apart from that with Q, as pictured in Figure 10, then if Q is may-NI then so too is the combination of P and Q. P restricts a part of Q's high level interface (as in restricted synchronisation above) and also has some independent high level activity)

Theorem 5.9 *If H_P and H_Q are disjoint, P has alphabet $H_P \cup M$, and Q is may-NI wrt $(H_Q \cup M, L)$, then $(P \underset{M}{\parallel} Q) \setminus M$ is may-NI wrt $(H_P \cup H_Q, L)$.*

Proof. We are considering the system $((P \underset{M}{\parallel} Q) \setminus M)$.

$$
\begin{aligned}
&(P \underset{M}{\parallel} Q) \setminus M \underset{H_P \cup H_Q}{\parallel} Stop \\
&= ((P \underset{H_P}{\parallel} Stop) \underset{M \cup H_Q}{\parallel} Q) \setminus M \cup H_Q \\
&= ((P \setminus H_P) \underset{M \cup H_Q}{\parallel} Q) \setminus M \cup H_Q \\
&= (((P \setminus H_P) \,|||\, Run_{H_Q}) \underset{M \cup H_Q}{\parallel} Q) \setminus M \cup H_Q \quad \text{since } Q \text{ is may-NI} \\
&= ((P \setminus H_P) \underset{M}{\parallel} Q) \setminus M \cup H_Q \\
&= ((P \underset{M}{\parallel} Q) \setminus M) \setminus H_P \cup H_Q
\end{aligned}
$$

The result also holds when P and Q allow signal events in H_P and H_Q. In this case, H_P is partitioned into HI_P and HO_P, and H_Q is partitioned into HI_Q and HO_Q. In the proof we will use the characterisation for may-NI for Q, that $(U_1 \underset{M \cup HI_Q}{\parallel} Q) \setminus M \cup H_Q = (U_2 \underset{M \cup HI_Q}{\parallel} Q) \setminus M \cup H_Q$ for any processes U_1 and U_2.

$$
\begin{aligned}
&(((P \underset{M}{\parallel} Q) \setminus M) \underset{HI_P \cup HI_Q}{\parallel} Stop) \setminus HO_P \cup HO_Q \\
&= (((P \underset{HI_P}{\parallel} Stop) \underset{M}{\parallel} (Q \underset{HI_Q}{\parallel} Stop)) \setminus M) \setminus HO_P \cup HO_Q \\
&= ((P \underset{HI_P}{\parallel} Stop) \underset{M \cup HI_Q}{\parallel} Q) \setminus M \cup HI_Q \cup HO_Q \cup HO_P \\
&= ((P \setminus HI_P \,|||\, Run_{HI_Q}) \underset{M \cup HI_Q}{\parallel} Q) \setminus M \cup HI_Q \cup HO_Q \cup HO_P
\end{aligned}
$$

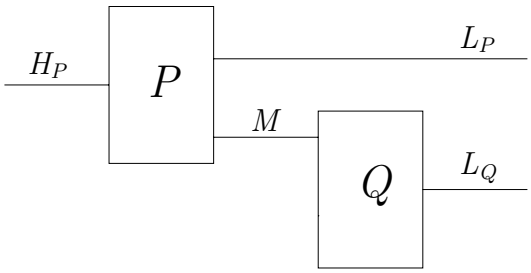

Fig. 11. Q connected to P's low level interface

$$= ((P \setminus HI_P) \underset{M}{\parallel} Q) \setminus M \cup HI_Q \cup HO_Q \cup HO_P$$
$$= ((P \underset{M}{\parallel} Q) \setminus M) \setminus HI_P \cup HI_Q \cup HO_P \cup HO_Q$$

This completes the proof for this case. □

5.8 Limited high-low connection 2

If all of the output from P is provided as input to Q, as pictured in Figure 11, then may-NI of the combination follows purely from that of P. In fact, Q can be any process and need not itself provide non-interference. This setup is much easier to verify than the previous one.

Theorem 5.10 *If P is may-NI wrt $(H_P, M \cup L_P)$, and Q has alphabet $M \cup L_Q$, then $(P \underset{M}{\parallel} Q) \setminus M$ is may-NI wrt $(H_P, L_P \cup L_Q)$.*

Proof. We are concerned with $(P \underset{M}{\parallel} Q)$.

$$\begin{aligned}(P \underset{M}{\parallel} Q) \setminus H_P &= ((P \setminus H_P) \underset{M}{\parallel} Q) \\ &= (P \underset{H_P}{\parallel} Stop) \underset{M}{\parallel} Q \\ &= (P \underset{M}{\parallel} Q) \underset{H_P}{\parallel} Stop\end{aligned}$$

In the case where there are high level signals, with P being may-NI on $((HI_P, HO_P), L_P)$, we have

$$\begin{aligned}(P \underset{M}{\parallel} Q) \setminus HI_P \cup HO_P &= ((P \setminus HI_P \cup HO_P) \underset{M}{\parallel} Q) \\ &= ((P \underset{HI_P}{\parallel} Stop) \setminus HO_P) \underset{M}{\parallel} Q \\ &= ((P \underset{M}{\parallel} Q) \underset{HI_P}{\parallel} Stop) \setminus HO_P\end{aligned}$$

□

Thus may-NI is retained by the combination in this case too.

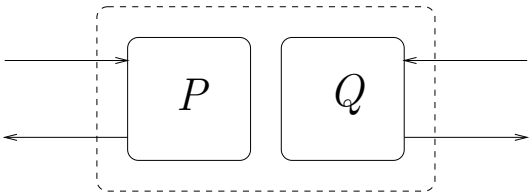

Fig. 12. Cross product of P and Q

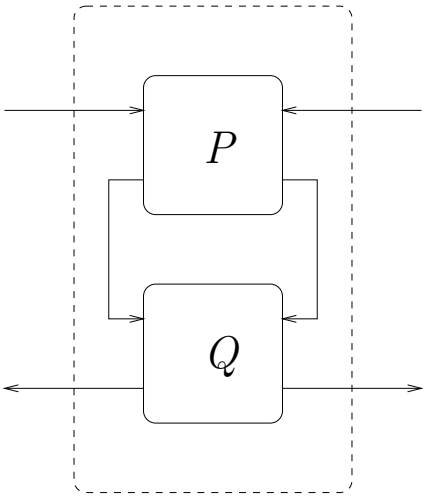

Fig. 13. Cascade of P and Q

5.9 External composition

The three composition constructs most frequently considered in the literature with regard to composability of non-interference properties are cross-product, cascade, and feedback. The descriptions of [12] will be considered here. In that context, high and low events are classified into inputs and outputs. These can be considered in terms of the compositions that have been discussed already.

5.9.1 Cross-product

The cross product of two systems simply considers one as high level and the other as low level, with no communication between them. It can be pictured as in Figure 12.

In this situation, P has alphabet H and Q has alphabet L. They are therefore both may-NI wrt (H, L). Their combination may be considered as an interleaving $P \mid\mid\mid Q$, yielding may-NI. Alternatively, they may be considered as a chain $(P \underset{M}{\parallel} Q) \setminus M$, (where neither of them ever performs events on the common channel M, or where M is empty). This also yields may-NI.

5.9.2 Cascade

A cascade allows both high and low level communication between P and Q, as pictured in Figure 13.

In CSP terms, there will be some intersection $H_I = H_P \cap H_Q$ between the high level interfaces H_P and H_Q of P and Q, and also $L_I = L_P \cap L_Q$ between their low level interfaces L_P and L_Q. A cascade will then be a parallel combination of P and Q, with their common interfaces treated as internal channels:

$$(P \underset{H_I \cup L_I}{\parallel} Q) \setminus (H_I \cup L_I)$$

Since parallel combination (synchronisation) preserves may-NI , and so does hiding, it follows that cascade also preserves may-NI.

5.9.3 Feedback

Feedback is the most complicated of the three composition operators, and in general its definition varies depending on the framework being used for analysis, and can be quite complicated. In [12], some of the output of P is supplied as input to Q, and similarly the output from Q is provided as input to P. P also accepts input from and provides output to its environment. Messages may be transformed when passing between P and Q. We will use processes R_H and R_L to describe such transformations. Events are categorised (as high or low) the same way by both processes, so there is no connection from one process' high events to another's low events.

In the CSP framework one way of describing this form of feedback is as follows:

$$(P \;|||\; Q) \underset{H}{\parallel} R_H) \underset{L}{\parallel} R_L) \setminus I$$

for suitable R_H and R_L which describe the transformations on the messages passing between P and Q. This is pictured in Figure 14. Since this is composed entirely using operations which preserve may-NI (provided there are no signals), it follows that the resulting system is may-NI. Indeed, even if the processes R_H and R_L were absent, and the processes P and Q connected up directly, this is still true.

In this description a feedback composition in CSP is not much more complicated than a cascade: it is simply a parallel combination of two systems with some additional process behaviour at the high and the low level, with some channels hidden. It follows that it preserves may-NI.

Synchronous I/O semantic frameworks describe a form of pure feedback which connects an output of a process directly to one of its own inputs. This form of feedback, in which an output *synchronises* with an input of the same process, cannot be modelled directly in CSP, since each event in a process execution is independent. However, a buffered feedback loop is easy to describe. Figure 15 pictures a high level output ho being fed into a high level input hi. This is an example of restricted synchronisation discussed above: the result will be may-NI wrt (H, L) whenever P does wrt $(H \cup \{hi, ho\}, L)$.

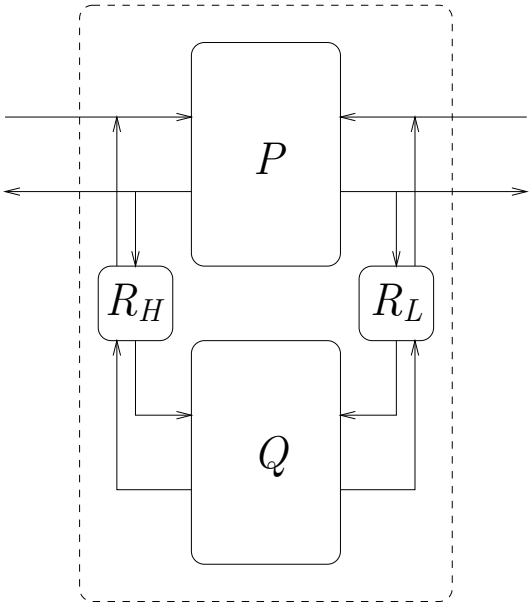

Fig. 14. CSP view of Feedback between P and Q

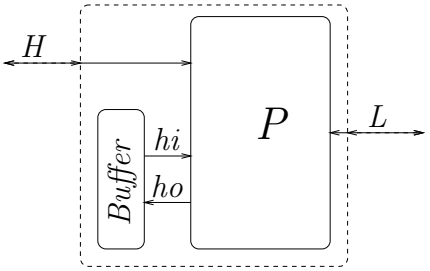

Fig. 15. CSP view of Buffered Feedback on P

6 Discussion

This paper has formulated a non-interference property, *may non-interference* and has shown that it is compositional in the sense that it is preserved by a wide range of CSP composition operators, including interleaving, synchronising parallel (under certain conditions), restriction, hiding, chaining, and forms of high-low connection. As a result it is also preserved by the external operators typically considered in studies of non-interference: cross product, cascade, and forms of feedback. These results hold both when all events are considered as synchronisations, and when high level outputs are considered as signals that cannot be blocked.

The use of a process algebraic approach was originally motivated by the fact that concurrency theory provides a mature framework for defining and analysing properties, and for reasoning about combinations of components. This has resulted in a wide range of results, and in simple proofs.

6.1 Semantic frameworks

There have been a large number of approaches to non-interference. To some extent compositionality results must depend on the semantic framework used to model systems, and its composition operators, as well as the characterisation of the non-interference properties within that framework. For example, Wittbold and Johnson [18] and McLean [9] consider processes as sets of traces in which all system inputs and outputs occur in each step. Systems to be composed thus proceed in synchronous lockstep. On the other hand, Goguen and Meseguer [4], McCullough [8] and O'Halloran [11] consider sets of traces in which high and low level inputs and outputs occur independently of each other, as we do in this paper. Zakinthinos and Lee [19] also consider such traces, but require that processes are open on all of their inputs. Noninterference properties in these frameworks generally take the form that if the system has an execution with a particular low level projection, then there must be other executions, with different high level activity and with the same low level projection. For example, noninference [11] requires that if an execution has a particular low level projection, then there must be some other execution with that same projection but no high level activity at all. Forward correctability, described in [18], requires that it must be possible to alter any high level input value, and obtain another trace exactly the same except possibly for the values of subsequent high level outputs. Separability [9] requires that for any low level projection of any execution, and any high level projection of any execution, there must be some execution which combines exactly those two views. These definitions (and others) are motivated by the philosophy that the low level view of an execution should not give away certain information about what has occurred at the high level. Composition results for these properties are obtained in the state machine or event system frameworks in which they are defined and analysed, but the definitions of system composition in these different semantic frameworks will be different to each other, and it is not clear how to compare them, or how to apply results from one approach to another.

6.2 Process algebras

There are also a number of approaches which make use of process algebraic techniques, based on CSP [16,15,7], CCS [2], or the (asynchronous) π-calculus [6]. These are often concerned with issues such as nondeterminism, and generally use failures information (traces together with possible refusals) [16,15], bisimulation information (whether processes can match executions by passing through matching states) [2], or testing [6] to characterise non-interference properties. Thus if some high level activity can possibly lead to different low level offers being made, then such characterisations will identify this even if the low level trace does not provide that information.

For example, the process

$$P = (l \rightarrow h \rightarrow Stop) \,\Box\, (h \rightarrow (Stop \sqcap l \rightarrow Stop))$$

can perform l and h in either order, but if h occurs first then there is a nondeterministic possibility that l will not be offered; the refusal of l will provide information that h has occurred. However, each of its low level traces $\langle\rangle$ and $\langle l \rangle$ is compatible with each of its high level traces $\langle\rangle$ and $\langle h \rangle$. Some information other than traces is required to identify the possibility of information flow via the refusal.

Ryan proposed a generalisation of the Goguen and Meseguer unwinding characterisation of non-interference, [16] by stipulating that the set of refusal events and the set of possible next events for any state are the same for traces whose low level views match. This property is violated by P above, since initially (on the trace $\langle\rangle$) it cannot refuse l, but it can refuse l after occurrence of the first h (on the trace $\langle h \rangle$).

More recently, Roscoe, Woodcock, and Wulf [15] proposed that a process exhibits non-interference if its low level behaviour is deterministic whenever its high level behaviour is abstracted. This means that whatever high level behaviour occurs, the low level view will be unaffected. This property also incorporates refusal information, since this is required to identify nondeterminism. The process P above does not meet this property, since when h is abstracted the resulting low level process can both perform and refuse to perform l. This definition is very strong, and is preserved by refinement (unlike the majority of non-interference properties), but it rules out many processes which contain some nondeterminism, even if there is no information flow.

To allow nondeterminism, and motivated by the desire to find a property preserved by refinement, Lowe [7] recently proposed a definition which essentially requires that any resolution of nondeterminism in the process yields a process that meets Ryan's unwinding conditions. P fails to meet this property as well.

In the context of CCS, Focardi and Gorrieri [3] propose that a system Q does not allow information flow if Q prevented from performing H is 'equivalent' to Q with all occurrences of H hidden: a low level user should not be able to tell whether events in H have occurred or not. By varying the notion of 'equivalence' to cover a variety of failure and bisimulation equivalences, this gives rise to a variety of forms of non-interference. Our characterisation of may-NI is the same as their notion instantiated with trace equivalence or may testing equivalence. The process P above does not meet their definition when failures equivalence is used: the possibility of l's refusal after the occurrence of h is identified. By using bisimulation equivalence, yet more subtle properties can also be identified.

For example, the process

$$\begin{aligned} P' = {} & l_1 \rightarrow l_2 \rightarrow Stop \,\Box\, l_1 \rightarrow l_3 \rightarrow Stop \\ & \Box\, h \rightarrow l_1 \rightarrow (l_2 \rightarrow Stop \sqcap l_3 \rightarrow Stop) \end{aligned}$$

is able to perform l_1 at the low level followed nondeterministically by either

l_2 or l_3. This is consistent with h either present or absent at the high level. However, if h is performed, then the nondeterministic choice between l_2 and l_3 is not resolved until after l_1, whereas if h is not performed, then it is resolved on the occurrence of l_1. This is the kind of distinction that bisimulation equivalence picks up, and so by using (weak) bisimulation as the equivalence for non-interference, we find that P' allows interference since h can interfere with the point at which a choice is resolved. If this kind of information is available to a low level user, then there will have been some information flow.

Even in cases where this equivalence is stronger than required, bisimulation equivalence is often efficient to check, and in cases where it does hold it will imply weaker properties that are required.

6.3 Testing

Testing equivalences are also considered by Focardi and Gorrieri, though they use such equivalences directly in their definition, that P blocked on H should be testing equivalent to P with H hidden. This is the characterisation arrived at in this paper, but it was obtained from a simpler starting point in a definition of non-interference. This paper thus provides additional insight into their definition.

A formulation similar to the starting point taken in this paper is proposed by Hennessy and Riely [6] in the context of the more expressive asynchronous π-calculus. They formulate conditions (using types) on P and Q which guarantee a non-interference property: that if P and Q are equivalent under may testing at a particular security level, then $P \mid H$ and $Q \mid K$ are equivalent under may testing for arbitrary top level processes H and K. Our formulation uses a single process in place of P and Q, requiring that $P \mid H$ and $P \mid K$ are indistinguishable, but this is a minor difference. The greater expressiveness of the π-calculus over CSP allows dynamic process creation, and network reconfiguration, which may provide more opportunities for information to flow from high to low. Questions as to which systems exhibit non-interference are thus different to those for CSP. However, the formulation of non-interference in terms of the impact of the system, in conjunction with a high level user or process, on its low level environment, is strikingly similar to that presented in this paper.

6.4 Must testing

The process P described above allows information about occurrence of the high level event h to be deduced from a low level refusal of l. The may testing characterisation of non-interference presented in this paper is not fine enough to capture this, and in fact the process P is may-NI. However, it is easy to imagine a scenario in which the event l is blocked during an execution, and that this is observed at the low level. It would seem that a formulation of non-interference which is sensitive to refusal information would be of benefit.

The form of testing which corresponds to failures/divergences equivalence in CSP is *must testing*. This states that P must pass a test T if *all* (maximal) executions of $(P \underset{\Sigma}{\parallel} T) \setminus \Sigma$ reach a success state of T. Then P and Q are equivalent under must testing if they must pass exactly the same tests T.

This form of testing seems to offer some opportunity for making the kind of distinctions we wish to make. It would be natural to state that P is must-NI if $U_1 \underset{H}{\parallel} P$ is must equivalent to $U_2 \underset{H}{\parallel} P$ for any two high level processes U_1 and U_2. For example, the test $T = l \rightarrow \mathit{Success}$ is able to distinguish $\mathit{Stop} \underset{h}{\parallel} P$ from $h \rightarrow \mathit{Stop} \underset{h}{\parallel} P$ through must testing: the first process must pass T, but the second might not, since the l might be prevented from occurring. However, must-NI does not exhibit such pleasant properties as may-NI.

Firstly, it turns out that must-NI and strong must-NI (in which high level users can observe, though not prevent, low level events directly) are distinct. For example, the system

$$P = l_1 \rightarrow (\,(\mathit{Stop} \sqcap l_2 \rightarrow \mathit{Stop}) \,\Box\, h_1 \rightarrow l_2 \rightarrow \mathit{Stop})$$
$$\Box\; h_1 \rightarrow l_1 \rightarrow (\mathit{Stop} \sqcap l_2 \rightarrow \mathit{Stop})$$

is must-NI because at the low level it is always guaranteed to initially offer l_1, and then either refuse l_2 or offer it. If a high level process in the configuration of Figure 2 can initially refuse h_1 then this will not be affected by the occurrence of l_1, and so both alternatives must be possible. If it can initially perform h_1 then this might occur before l_1 and hence again lead to the two possibilities.

On the other hand, P is not strong-NI. The high level process

$$l_1 \rightarrow ((h_1 \rightarrow l_2 \rightarrow \mathit{Stop}) \,\Box\, (l_2 \rightarrow \mathit{Stop}))$$

in the configuration of Figure 3 is able to make h_1 available precisely after l_1 has occurred, so that l_2 cannot be refused after l_1. On the other hand, a high level process *Stop* is not able to do this. Thus the low level test $l_1 \rightarrow l_2 \rightarrow \mathit{Success}$ will distinguish between these two possibilities.

Conversely, if a process is strong must-NI, then it is must-NI.

With regard to compositionality, must-NI is not preserved by many of the CSP operators, not even by interleaving (and hence not by general parallel composition)! For example, we have seen that process P above is must-NI. So too is

$$Q = l_3 \rightarrow h_3 \rightarrow \mathit{Stop}$$

However, $P \;|||\; Q$ is not must-NI. A high level process $h_3 \rightarrow h_1 \rightarrow \mathit{Stop}$ can provide some information concerning the occurrence of h_1.

Consider the low level behaviour $(\langle l_1, l_3 \rangle, \{l_2\})$ of $P \;|||\; Q$. This is possible if the high level process is *Stop*, but not if it is $h_3 \rightarrow h_1 \rightarrow \mathit{Stop}$, since the occurrence of l_3 after l_1 ensures that h_1 can only occur after l_1, and hence ensures that l_2 cannot be refused after $\langle l_1, l_3 \rangle$. Thus the test $l_1 \rightarrow l_3 \rightarrow l_2 \rightarrow$

Success allows those two high level processes to be distinguished under must testing.

The situation is fractionally better with regard to strong must-NI. This counterexample does not hold, because P is not itself strong must-NI. It is currently an open question as to whether or not strong must-NI is preserved by interleaving (let alone general parallel composition).

Although both forms of must-NI are initially plausible definitions, it is not clear what they correspond to in an intuitive sense. Furthermore, the rather bizarre examples given above seem to indicate that they are not straightforward. When a system is examined for information flow, we are generally concerned with what might leak during a single execution. We do not generally assume that the low level agent has the opportunity to interact with the system, from the same state, as many times as is necessary to exercise all of its possibilities. Yet this is the situation encapsulated by must testing. For the kind of property we are concerned with, it would be more appropriate to include some notion of *refusal test* [13,10] so that the refusal information obtained during a single execution can be accounted for in a semantic model that deals with this directly, perhaps in conjuction with an extended notion of may testing. This is the subject of current research.

6.5 Acknowledgements

This work has benefitted from discussions with Peter Ryan, and from the careful reading and comments of the anonymous referees. The work has also received financial support from DERA.

References

[1] J. W. Davies and S. A. Schneider. Recursion induction for real-time processes. *Formal Aspects of Computing*, 5(6), 1993.

[2] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1), 1994/5.

[3] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9), 1997.

[4] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, 1982.

[5] M. Hennessy. *Algebraic Theory of Processes*. MIT press, 1988.

[6] M. Hennessy and J. Riely. Information flow vs. resource access in the asynchronous pi-calculus. In *27th International Colloquium on Automata, languages and programming*, number 1853 in LNCS. Springer-Verlag, 2000.

[7] G. Lowe. Defining information flow. Technical Report 1999/3, University of Leicester, 1999.

[8] D. McCullough. Specifications for multi-level security and a hook-up property. In *IEEE Symposium on Security and Privacy*, 1987.

[9] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *IEEE Symposium on Research in Security and Privacy*, 1994.

[10] A. Mukkaram. *A Refusal testing Model for CSP.* D.Phil, Oxford University, 1993.

[11] C. O'Halloran. A calculus of information flow. In *Proceedings of European Symposium on Research in Information Security*, 1990.

[12] R. V. Peri, W. A. Wulf, and D. M. Kienzle. A logic of composition for information flow properties. In *9th IEEE Computer Security Foundations Workshop*, 1996.

[13] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50(3), 1987.

[14] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall, 1997.

[15] A. W. Roscoe, J. Woodcock, and L. Wulf. Non-interference through determinism. In *European Symposium on Research in Computer Security*, 1994.

[16] P. Y. A. Ryan. A CSP formulation of non-interference and unwinding. *Cipher*, 1991.

[17] S. A. Schneider. *Concurrent and Real Time Systems: the CSP approach.* John Wiley, 1999.

[18] J. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. In *Symposium on Research on Security and Privacy*, 1990.

[19] A. Zakinthinos and E. S. Lee. A general theory of security properties. In *IEEE Symposium on Security and Privacy*, 1997.

A The Traces Model

There is a universal set of events Σ. A trace is a finite sequence of events drawn from Σ. A set of traces S is in the traces model M_T if and only if:

(i) S is non-empty;

(ii) S is prefix closed: whenever $tr_1 \frown tr_2 \in S$, then $tr_1 \in S$

The function $traces : CSP \rightarrow M_T$ maps CSP process descriptions to sets of traces in M_T. It is defined compositionally for each of the operators provided by CSP. Here we give the definitions for the operators introduced in this paper.

$$
\begin{aligned}
traces(Stop) &= \{\langle\rangle\} \\
traces(a \rightarrow P) &= \{\langle\rangle\} \cup \{\langle a\rangle \frown tr \mid tr \in traces(P)\} \\
traces(c!v \rightarrow P) &= \{\langle\rangle\} \cup \{\langle c.v\rangle \frown tr \mid tr \in traces(P)\} \\
traces(c?x : T \rightarrow P(x)) &= \{\langle\rangle\} \cup \{\langle c.v\rangle \frown tr \mid v \in T \wedge tr \in traces(P(v))\} \\
traces(P \sqcap Q) &= traces(P) \cup traces(Q) \\
traces(P \;\Box\; Q) &= traces(P) \cup traces(Q) \\
traces(P \underset{A}{\parallel} Q) &= \{tr \mid \exists\, tr_P, tr_Q.\ tr_P \in traces(P) \wedge tr_Q \in traces(Q) \\
&\qquad \wedge\ tr \in tr_P \underset{A}{\parallel} tr_Q\} \\
traces(P \;|||\; Q) &= \{tr \mid \exists\, tr_P, tr_Q.\ tr_P \in traces(P) \wedge tr_Q \in traces(Q) \\
&\qquad \wedge\ tr \in tr_P \underset{\varnothing}{\parallel} tr_Q\} \\
traces(P \setminus A) &= \{tr \upharpoonright (\Sigma \setminus A) \mid tr \in traces(P)\} \\
traces(Run_A) &= \{tr \mid tr \in A^*\}
\end{aligned}
$$

where $tr \upharpoonright A$ is the projection of the trace tr onto the set $A \subseteq \Sigma$: the maximal subsequence of tr all of whose events appear in A.

The set of traces $tr_1 \underset{A}{\parallel} tr_2$ is defined inductively as follows:

$$
\begin{aligned}
& \langle\rangle \in \langle\rangle \underset{A}{\parallel} \langle\rangle \\
tr \in tr_1 \parallel tr_2 \wedge a \in A \quad &\Rightarrow \quad \langle a\rangle \frown tr \in (\langle a\rangle \frown tr_1) \underset{A}{\parallel} (\langle a\rangle \frown tr_2) \\
tr \in tr_1 \parallel tr_2 \wedge b \notin A \quad &\Rightarrow \quad \langle b\rangle \frown tr \in (\langle b\rangle \frown tr_1) \underset{A}{\parallel} tr_2 \\
tr \in tr_1 \parallel tr_2 \wedge b \notin A \quad &\Rightarrow \quad \langle b\rangle \frown tr \in tr_1 \underset{A}{\parallel} (\langle b\rangle \frown tr_2)
\end{aligned}
$$

It is the set of traces which agree with both tr_1 and tr_2 on events in the set A, and which interleave on the events from tr_1 and tr_2 not in the set A.

For a process P defined by a recursive equation $P = F(P)$, the traces of P are given by:

$$traces(P) = \bigcup_{n \in \mathbb{N}} traces(F^n(Stop))$$

For a family of processes $\langle P_i\rangle_{i \in I}$ defined by a family of recursive equations $P_j = F_j(\langle P_i\rangle_{i \in I})$ (where each F_j is a function on the entire vector of P_i processes), we define the function $\underline{F} : M_T^I \rightarrow M_T^I$ as follows:

$$\underline{F}(\langle X_i\rangle_{i \in I}) = \langle F_j(\langle X_i\rangle_{i \in I})\rangle_{j \in I}$$

Then the traces of all the P_i are given by:

$$traces(P_i) = \bigcup_{n \in \mathbb{N}} traces((\underline{F}^n(\langle Stop\rangle_{i \in I}))_k)$$

A fuller explanation of the semantics of mutual recursion in CSP can be found in [1].