Published in Formal Aspects of Computing. The final publication is available at www.springerlink.com

On the Limits of Refinement-Testing for Model-Checking CSP

Toby Murray[1,2,3]

[1]Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom
[2]NICTA, Australia
[3]School of Computer Science and Engineering, UNSW, Sydney, Australia
`toby.murray@nicta.com.au`

Abstract. Refinement-checking, as embodied in tools like FDR, PAT and ProB, is a popular approach for model-checking refinement-closed predicates of CSP processes. We consider the limits of this approach to model-checking these kinds of predicates.

By adopting Clarkson and Schneider's hyperproperties framework, we show that every refinement-closed denotational predicate of finitely-nondeterministic, divergence-free CSP processes can be written as the conjunction of a safety predicate and the refinement-closure of a liveness predicate. We prove that every safety predicate is refinement-closed and that the safety predicates correspond precisely to the CSP refinement checks in finite linear observations models whose left-hand sides (*i.e.* specification processes) are independent of the systems to which they are applied.

We then show that there exist important liveness predicates whose refinement-closures cannot be expressed as refinement checks in any finite linear observations model $\mathcal{M}$, divergence-strict model $\mathcal{M}^{\Downarrow}$ or non-divergence-strict divergence-recording model $\mathcal{M}^{\#}$, *i.e.* in any standard CSP model suitable for reasoning about the kinds of processes that FDR can handle, namely finitely-branching ones. These liveness predicates include liveness properties under intuitive fairness assumptions, branching-time liveness predicates and non-causation predicates for reasoning about authority. We conclude that alternative verification approaches, besides refinement-checking, currently under development for CSP should be further pursued.

Keywords: Refinement-testing, expressiveness, CSP, model-checking, hyperproperties.

1. Introduction

Since the introduction [Ros94, RGG+95] of the FDR model-checking tool [GGH+05] in the early-to-mid 1990s, the process algebra CSP [Hoa85, Ros97] has remained a popular formalism in which complex systems can be modelled, and their adherence to certain correctness conditions automatically proved with FDR (and other, more recent, tools such as PAT [SLD09] and ProB [LF08]). The CSP/FDR approach differs from other

Correspondence and offprint requests to: Toby Murray model checking approaches because, rather than testing whether a system satisfies a predicate expressed in a logic separate from the language used to model the system, FDR tests CSP *refinement assertions*. These are assertions that involve the system being analysed and assert that the system, or some CSP process that involves it, *refines* some specification, which is also written in CSP. These CSP refinement assertions express predicates about the system being checked, such that a refinement assertion holds if and only if the system satisfies the predicate that the assertion expresses.

A CSP refinement assertion is a statement of the form

$$P \sqsubseteq_{\mathcal{M}} Q,$$

where P and Q are CSP processes and $\mathcal{M}$ is a denotational semantic model of CSP. P is referred to as the *left-hand-side*, or *specification* process, of the refinement check, while Q is referred to as the *right-hand-side*, or *system*, of the refinement check. In this paper, we restrict our attention to the *standard* denotational semantic models $\mathcal{M}$ of CSP [Ros08, Ros09], which represent a process by various sets of linear behaviours that it can exhibit. The model $\mathcal{M}$ might record multiple kinds of behaviours of processes, such as recording traces as well as stable-failures. This statement asserts that for each kind of behaviour recorded by $\mathcal{M}$, all behaviours of this kind that $\mathcal{M}$ records as being able to be exhibited by Q, $\mathcal{M}$ also records as being able to be exhibited by P. In general, if $\mathcal{M}$ records k different kinds of behaviours, we may denote the representation of a process R in $\mathcal{M}$ by the sets $\mathcal{M}_1[\![R]\!], \ldots, \mathcal{M}_k[\![R]\!]$, where $\mathcal{M}_i[\![R]\!]$ is the set of R's behaviours of the ith kind that $\mathcal{M}$ records. We then have that for any processes P and Q

$$P \sqsubseteq_{\mathcal{M}} Q \Leftrightarrow \forall\, 1 \leq i \leq k \;.\; \mathcal{M}_i[\![Q]\!] \subseteq \mathcal{M}_i[\![P]\!]. \tag{1}$$

When $\mathcal{M}$ records just a single kind of behaviour, we write simply $\mathcal{M}[\![P]\!]$ to denote P's representation in $\mathcal{M}$.

As a simple example, to assert that some system *System* never performs any of the events in some set A, one simply tests the refinement

$$CHAOS_{\Sigma - A} \sqsubseteq_{\mathcal{T}} System. \tag{2}$$

Σ is the set of all visible events and $\mathcal{T}$ denotes CSP's *traces model* [Hoa80]. For some set of events $B \subseteq \Sigma$, $CHAOS_B$ is the most general process that performs events only from the set B. The traces model $\mathcal{T}$ records just a single kind of behaviour, namely all finite sequences of visible events that a process can perform. Such sequences are called *traces*. Hence, by (1), this refinement holds iff $\mathcal{T}[\![System]\!] \subseteq \mathcal{T}[\![CHAOS_{\Sigma-A}]\!]$, *i.e.* iff every sequence of visible events performed by *System* can be performed by $CHAOS_{\Sigma-A}$. Since the latter performs all sequences of events that don't contain any A-events, this refinement holds iff the former never performs any A events, as required.

This basic approach, in which predicates of systems are expressed as refinement checks to be carried out automatically by FDR, has proved remarkably successful. In particular, CSP refinement-checking has been used to express predicates as diverse as safety properties [Low96], deadlock-freedom [Ros97, Chapter 13], determinism [Laz99], refinement-closed formulations of noninterference [Low07] (and variations thereof [ML09]), responsiveness [RSR04], certain forms of non-causation [ML07], and certain kinds of temporal properties [LMC01, Low08] including certain liveness properties like the LTL property $\Diamond\, e$, which asserts that the event e must eventually occur.

Most, if not all, of the examples from the previous paragraph involve predicates that can be expressed as refinement checks in which the left-hand side of the test is independent of the system to which the predicate is being applied. Such refinement checks can be written in the form

$$Spec \sqsubseteq_{\mathcal{M}} G(System) \tag{3}$$

where *Spec* is a *specification* process for the predicate being tested (*e.g.* $CHAOS_{\Sigma-A}$ in (2) above), *System* is the system to which the predicate is being applied and $G(\_)$ is a CSP *context*, *i.e.* a CSP process expression parameterised by *System* (*e.g.* the identity expression as used in (2), or perhaps an expression that runs multiple copies of *System* in parallel as in [Low09]). Such refinement checks are generally much easier for FDR (and similar tools such as PAT [SLD09] and ProB [LF08]) to carry out than the remaining checks for which the left-hand side depends on *System*. This is because the left-hand side of any refinement check must usually be *normalised* (although approaches exist that partially avoid this *e.g.* [SLD09, ACH$^+$10]) and the complexity of normalisation is (in the worst-case) exponential in the state space of the process being normalised. When the left-hand side of a refinement check depends on the system being analysed, the complexity of model-checking by refinement-testing can become exponential in the size of the system being checked. (In contrast, the complexity of a refinement check is linear in the state space of its right-hand side.)

It is easily shown that all predicates that can be expressed as refinement checks of the form (3) are *refinement-closed* in the model $\mathcal{M}$, meaning that whenever they hold for a CSP process P, they must hold for all processes Q for which $P \sqsubseteq_{\mathcal{M}} Q$. This is formally defined as follows.

Definition 1 (Refinement-Closed in $\mathcal{M}$). A predicate *Pred* is *refinement-closed in the model* $\mathcal{M}$ iff for all CSP processes P

$$\mathit{Pred}(P) \Rightarrow \forall\, Q \;.\; P \sqsubseteq_{\mathcal{M}} Q \Rightarrow \mathit{Pred}(Q).$$

To see that any predicate *Pred* expressed as (3) must be refinement-closed in $\mathcal{M}$, consider some process P such that $\mathit{Pred}(P)$. Then the refinement $\mathit{Spec} \sqsubseteq_{\mathcal{M}} G(P)$ must hold. Consider some process Q that refines P, *i.e.* where $P \sqsubseteq_{\mathcal{M}} Q$. It is the case that for any CSP context $G(\_)$ and model $\mathcal{M}$, if $P \sqsubseteq_{\mathcal{M}} Q$ then $G(P) \sqsubseteq_{\mathcal{M}} G(Q)$ [Ros09, Ros08]. Hence, we have $\mathit{Spec} \sqsubseteq_{\mathcal{M}} G(P) \sqsubseteq_{\mathcal{M}} G(Q)$. Because refinement is transitive, we must have that $\mathit{Spec} \sqsubseteq_{\mathcal{M}} G(Q)$ and so $\mathit{Pred}(Q)$ must hold. Hence, *Pred* must be refinement-closed in $\mathcal{M}$.

From the above two paragraphs, one can conclude that refinement-testing is best suited for model-checking predicates that are refinement-closed in some CSP model $\mathcal{M}$. In this paper, we consider the limits of using refinement-testing to model-check refinement-closed predicates of CSP processes. In particular, we consider the entire set of all refinement-closed denotational predicates that can be defined for a *finitely nondeterministic*, *divergence-free* (FNDF) CSP process *System* and the question as to which of these predicates can, and cannot, be expressed as refinement checks. We restrict our attention to predicates of systems that are finitely nondeterministic, since FDR can handle only finitely nondeterministic processes, and divergence-free, since divergence is almost always an incorrect behaviour for a system to exhibit and so a system that is divergent is normally *ipso facto* incorrect. We also restrict our attention to a fragment of CSP that excludes termination and sequential composition, for simplicity, and for which the set Σ of visible events is finite, as required by FDR.

Note however that we do not necessarily restrict our attention to just *finitary* refinement checks [Ros04], which are those that are finite state when the system being analysed is finite state; although any refinement check that is not finitary cannot be directly checked by FDR. This is because, like *e.g.* Roscoe [Ros05a] before us, we are primarily interested in the limits of which predicates can be expressed as refinement checks in the CSP language. Any predicate that is to be expressed as a practical refinement check will naturally require a finite encoding, but exploring which predicates can and cannot be expressed finitely is beyond the scope of this paper.

We begin in Section 2 by presenting an overview of the fragment of CSP used in this paper, and the hierarchies [Ros09, Ros08] of standard denotational semantic models for CSP. With the aid of the topmost element of the simplest of these hierarchies, namely the denotational model $\mathcal{FL}$ [Ros08, Ros09] of the hierarchy of *finite linear observations* models, we define the set **Pred** that contains all of the predicates (refinement-closed and otherwise) that can be defined over the FNDF CSP processes in any standard denotational model for CSP.

Then in Section 3, we show how the problem of expressing the refinement-closed members of **Pred** can be decomposed into the two smaller problems of expressing those members of **Pred** that involve safety and those that are the *refinement-closures* of predicates that involve liveness, respectively. We do so by making use of Clarkson and Schneider's *hyperproperties* framework [CS08, CS10] to show that the entire set of predicates **Pred** can be understood by distinguishing those predicates that involve safety from those that involve liveness. In particular, we define the *completed linear observations (CLO) hyperproperties* and show that each predicate from **Pred** can be identified with a unique CLO hyperproperty. Clarkson and Schneider's distinction between *safety* and *liveness* hyperproperties is then adapted to distinguish the safety and liveness CLO hyperproperties, and hence the *safety* and *liveness* predicates from **Pred**.

We show that, when $\mathcal{M}$ is a finite linear observations model, all refinement checks of form (3) express safety predicates. We adapt Clarkson and Schneider's proof, that every hyperproperty can be expressed as the intersection of a safety and a liveness hyperproperty, to show that each predicate $\mathit{Pred} \in \mathbf{Pred}$ can be expressed as the conjunction of a safety predicate Pred_S and a liveness predicate Pred_L respectively. We show that when *Pred* is refinement-closed in some model $\mathcal{M}$, Pred_L may be replaced by its $\mathcal{M}$*-refinement-closure*, $\mathit{RC}_{\mathcal{M}}(\mathit{Pred}_L)$, *i.e.*, in this case, $\mathit{Pred} = \mathit{Pred}_S \cap \mathit{RC}_{\mathcal{M}}(\mathit{Pred}_L)$. Unlike Pred_L, $\mathit{RC}_{\mathcal{M}}(\mathit{Pred}_L)$ is guaranteed to be refinement-closed in $\mathcal{M}$, and so is more likely to be amenable to being expressed as a feasible refinement check, *i.e.* one whose left-hand-side is independent of the system being analysed, for FDR to carry out. Hence, the problem of how to express *Pred* as a refinement check can be decomposed into the two smaller problems of how to express $Pred_S$, a safety predicate, and $RC_{\mathcal{M}}(Pred_L)$, the refinement-closure of a liveness predicate, respectively as refinement checks.

Then, in Section 4, we show that every safety predicate $Pred_S \in \mathbf{Pred}$ can be expressed as a refinement check of form (3) in the model $\mathcal{FL}$ (*i.e.* where $\mathcal{M} = \mathcal{FL}$). This result implies (a) that all safety predicates are naturally refinement-closed in $\mathcal{FL}$, and (b) that the safety predicates and the predicates that can be expressed as refinement checks of form (3) with $\mathcal{M}$ a finite linear observations model, are one and the same. We conclude that refinement-checking is well suited to expressing safety predicates.

However, we then consider predicates $RC_{\mathcal{M}}(Pred_L)$ that are the refinement-closures of liveness predicates $Pred_L$ in Section 5. These predicates include those liveness predicates $Pred_L$ that are (already naturally) refinement-closed. We show how refinement checking in models that record divergence can be used to verify certain refinement-closed liveness predicates that are violated by the presence of a finite number of infinite traces and finite behaviours. However, we then show that there exist useful $RC_{\mathcal{M}}(Pred_L)$ that do not fall into this category and cannot be expressed as refinement checks for FDR to carry out in any standard CSP model suitable for reasoning about finitely-branching processes, *i.e.* only those processes that FDR is able to support.[1] We extend a previous result of Roscoe from [Ros05a], to provide a general characterisation of certain predicates that cannot be expressed as CSP refinement checks in any such model of CSP. We show that these predicates include simple refinement-closed LTL properties that express liveness under intuitive notions of fairness, refinement-closed branching-time liveness predicates, as well as simple refinement-closed non-causation conditions for reasoning about *authority* [Mil06] in systems of interacting agents; however, these are surely only a tiny fraction of the set of all such predicates. This leads us to conclude that refinement-checking is not as well suited to expressing the refinement-closures of liveness predicates, and thus has real limitations for expressing arbitrary refinement-closed predicates.

In light of this result, in Section 6, we consider future and related work and sketch some other prospects for mechanically proving these kinds of refinement-closed predicates that cannot be tested via refinement-checking, including approaches based on automatically identifying certain *strongly connected subgraphs* within the labelled transition system that represents a system's operational semantics (see *e.g.* [Ros01, SLDW08, Liu09]), as well as those based on interactive theorem proving (see *e.g.* [IR05, IR08]). Finally, we conclude in Section 7.

2. CSP

In this section we give a brief overview of CSP as relevant to this paper (further details can be found elsewhere, *e.g.* [Ros97, Ros09, Ros08]), and define the set that contains all denotational predicates that can be defined for FNDF processes.

CSP is a process algebra for describing and reasoning about concurrent systems. CSP's syntax describes systems of concurrently executing *processes* that perform atomic *events* drawn from the set Σ; these processes communicate with each other by synchronising on the performance of common events. In this paper, we restrict our attention to a fragment of CSP in which the set Σ of events is finite, which is necessary in order to allow systems to be checked with FDR. Our fragment of CSP also excludes termination and sequential composition, for simplicity.

2.1. Syntax

CSP has a rich syntax for describing processes. The primitive process $STOP$ can perform no activity and represents deadlock. For an event $a \in \Sigma$, the process $a \to P$ can perform the event a and then behave like the process P. For a set of events $A \subseteq \Sigma$, the process $?a : A \to P_a$ is initially willing to perform all of the events from the set A. The environment has the choice of which event $x \in A$ will be performed. Once a particular event, $x \in A$, has been performed, $?a : A \to P_a$ behaves like the process P_a with the identifier a bound to the value x that was chosen.

Note that the "$\to$" operator binds tighter than all others.

CSP allows multi-part events to be defined, where a dot is used to separate each part of an event. The

[1] As explained in Section 2.4, these are those models that record finite behaviours and possibly divergences (whether *divergence-strict* or not [Ros05b]) but don't record other infinite behaviours.

"?" and "!" operators are then used to offer specific sets of events by pattern-matching on the structure of multi-part events. Suppose we define the set of events $\{\mathsf{plot}.x.y \mid x, y \in \mathbb{N}\}$. Then the process $\mathsf{plot}?x?y \to STOP$ offers its environment all of the events from the set $\{\mathsf{plot}.x.y \mid x, y \in \mathbb{N}\}$, whilst the process $\mathsf{plot}?x : \{1, \ldots, 5\}!3 \to STOP$ offers all events from $\{\mathsf{plot}.x.3 \mid x \in \{1, \ldots, 5\}\}$.

The process $P \,\Box\, Q$ can behave like either the process P or the process Q and offers its environment the initial events of both processes, giving the environment the choice as to which process it behaves like. If the environment chooses an initial event of P, $P \,\Box\, Q$ goes on to behave like P, and similarly for Q. The process $P \sqcap Q$ can also behave like either P or Q but doesn't allow the environment to choose which; instead, it makes this choice internally. It therefore nondeterministically offers its environment the initial events of either P or Q (but not both). The "$\sqcap$" operator is often used, therefore, to model nondeterminism.

The "$\sqcap$" operator extends naturally to sets of processes. If S is a set of processes, $\bigsqcap S$ behaves like the process that nondeterministically chooses one of the processes $P \in S$ to behave like, and then behaves like whichever process it chose. When the set Σ is finite, a process is *finitely nondeterministic* when it never uses an $\bigsqcap$ (or an equivalent operator from outside the fragment of CSP considered in this paper) over an infinite set of processes. Processes that are not finitely nondeterministic cannot, of course, be readily model-checked. For this reason, recall that we restrict our attention in this paper to predicates of finitely nondeterministic processes.

The process $P \rhd Q$ may behave like either P or Q. It can refuse to behave like P but cannot refuse to behave like Q. $P \rhd Q$ might be initially willing to behave like P, however this offer is inherently *unstable*: at any time, $P \rhd Q$ can perform some internal activity and transition to a state in which only the initial events of Q are offered.

$P \setminus A$ denotes the process obtained when P is run but all occurrences of the events in A are internalised, and occur as internal activity, and so are hidden from its environment. The environment cannot observe the occurrence of these events and is, therefore, prevented from synchronising on them.

The process $P \not\leftarrow b \not\rightarrow Q$ behaves like P if the boolean condition b is true and like Q otherwise.

The process $P[\![{}^{y_1, \ldots, y_n}/_{x_1, \ldots, x_n}]\!]$ behaves like the process P except that, for all $i \in \{1, \ldots, n\}$, it performs the event y_i whenever P performs the event x_i.

The process $P \underset{A}{\|} Q$ runs the processes P and Q in parallel forcing them to synchronise on all events from the set A. If S is a finite set of processes, then the process $\underset{A}{\|} S$ runs each process from S in parallel, forcing them all to synchronise on all events from the set A.

The process $P \,|||\, Q$ runs P and Q in parallel with no synchronisation (and, hence, no communication) between the two. We say that P and Q are *interleaved* here. For a finite set S of processes, $\big|\big|\big| S$ interleaves each process from S.

A process *diverges* when it performs an infinite amount of internal activity without performing a visible event from Σ. If we take the process $P = a \to P$ that continually performs the event a and then we internalise the occurrence of a using the hiding operator, arriving at the process $P \setminus \{a\}$, we see that $P \setminus \{a\}$ diverges immediately because each occurrence of a results in $P \setminus \{a\}$ performing some internal activity and P can perform an infinite number of as. The primitive process **div** diverges immediately and, for our purposes, is equivalent to the process $P \setminus \{a\}$ above. A process that never diverges is said to be *divergence-free.* Recall that, because divergence is almost always considered *ipso facto* to be an incorrect behaviour, in this paper we restrict our attention to predicates of divergence-free processes.

The process $CHAOS_A$ is the most nondeterministic, divergence-free process that performs events from the set A, for which $A \subseteq \Sigma$. This process may be defined, for our purposes, as follows.

$$CHAOS_A = \bigsqcap_{A' \subseteq A} ?a : A' \to CHAOS_A.$$

Note that since $\{\} \subseteq A$, $CHAOS_A$ can always deadlock immediately.

2.2. Notation

We use the following notation. Sequences are written between angle-brackets, so the sequence that contains the first 3 natural numbers is written $\langle 0, 1, 2 \rangle$. Let s and t be sequences. Then $s \,\hat{}\, t$ denotes the sequence obtained by concatenating s and t. We write $s \leq t$ to mean that s is *prefix* of t, *i.e.* $s \leq t \Leftrightarrow \exists\, u \,.\, s \,\hat{}\, u = t$.

We write $s < t$ when s is a strict prefix of t, *i.e.* when $\exists\, u \,.\, s\hat{\ }u = t \wedge u \neq \langle\rangle$. We write $s \upharpoonright A$ to denote the sequence obtained by removing from s all elements that are not members of the set A.

When using multi-part events, the notation $\{|c_1.c_2.\ldots.c_k|\}$ denotes the set of events whose first k components are respectively $c_1, c_2 \ldots, c_k$. So $\{|\mathsf{plot}|\} = \{\mathsf{plot}.x.y \mid x, y \in \mathbb{N}\}$ and $\{|\mathsf{plot}.1|\} = \{\mathsf{plot}.1.y \mid y \in \mathbb{N}\}$ using the example from the previous subsection.

If X is a set, then $\mathcal{P}(X)$ denotes the powerset of X. We use "$-$" to denote set difference, so that if X and Y are sets, then $X - Y$ denotes the set obtained by removing all Y-elements from X.

For this paper, we write *CSP* to denote the set that contains all finitely nondeterministic, divergence-free (FNDF) CSP processes.

2.3. The Finite Linear Observations Models and the model $\mathcal{FL}$

A number of denotational semantic models have been defined for CSP. Each of these represents a process by some set of *linear* behaviours that it can exhibit. By linear, we mean that each behaviour contains only information that could be obtained from observing a single execution of the process (and so, for instance, contains no branching). The simplest models are the *finite linear observations* models. A finite linear observations model is one that records only finite linear behaviours of a process. The traces model $\mathcal{T}$ [Hoa80], *stable-failures* model $\mathcal{F}$ [Ros97, Section 8.4] and the *refusal-traces* model $\mathcal{RT}$ [Muk93] (also known as the *refusal-testing* model) are examples of finite linear observations models, as is the model $\mathcal{FL}$ [Ros09], which is explained shortly. The *failures-divergences* model $\mathcal{F}^{\Downarrow}$ [BR85] (but traditionally denoted $\mathcal{N}$, see *e.g.* [Ros97]), however, is not a finite linear observations model, since it also records divergences, which are not finite behaviours.

The finite linear observations models form a natural hierarchy, according to the following ordering. One model $\mathcal{M}'$ is greater than or equal to another $\mathcal{M}$, written $\mathcal{M} \preceq \mathcal{M}'$, iff for every two processes P and Q

$$(\forall\, 1 \leq i \leq j \,.\, \mathcal{M}'_i[\![P]\!] = \mathcal{M}'_i[\![Q]\!]) \Rightarrow (\forall\, 1 \leq i \leq k \,.\, \mathcal{M}_i[\![P]\!] = \mathcal{M}_i[\![Q]\!])$$

where j and k are the numbers of different kinds of behaviours recorded by $\mathcal{M}'$ and $\mathcal{M}$ respectively, and $\mathcal{M}_i[\![P]\!]$ denotes the set of behaviours of the ith kind recorded by model $\mathcal{M}$ (and similarly for $\mathcal{M}'$). In other words, $\mathcal{M} \preceq \mathcal{M}'$ iff $\mathcal{M}'$ distinguishes every pair of processes distinguished by $\mathcal{M}$. It is the case that whenever $P \sqsubseteq_{\mathcal{M}'} Q$ and $\mathcal{M} \preceq \mathcal{M}'$, then $P \sqsubseteq_{\mathcal{M}} Q$. The models in this hierarchy already mentioned are ordered as follows:

$$\mathcal{T} \preceq \mathcal{F} \preceq \mathcal{RT} \preceq \mathcal{FL}.$$

This hierarchy also contains other models besides those mentioned here. However, Roscoe has shown that, due to the constraint placed on any standard model of CSP that it be a congruence [Ros09], if $\mathcal{M}$ is any model of this hierarchy other than $\mathcal{T}$, the traces model, then $\mathcal{F} \preceq \mathcal{M}$, *i.e.* $\mathcal{M}$ must be at least as powerful as the stable-failures model $\mathcal{F}$ [Ros09, Lemmas 7.3 and 7.4].

The topmost element of the hierarchy of finite linear observations models, *i.e.* the most powerful such model, is the model $\mathcal{FL}$ [Ros09, Ros08]. $\mathcal{FL}$ records finite behavioural sequences of form (4). That is, for any CSP process P, $\mathcal{FL}[\![P]\!]$ is a set of behaviours of the form

$$\langle A_0, a_0, A_1, a_1, \ldots, A_{n-1}, a_{n-1}, A_n\rangle. \tag{4}$$

This sequence represents that the process in question can perform the trace $\langle a_0, a_1, \ldots, a_{n-1}\rangle$ of visible events from Σ; each A_i indicates the events that the process was observed to be able to *stably* accept after performing the trace $\langle a_0, \ldots, a_{i-1}\rangle$. A process stably accepts the set of events $A \subseteq \Sigma$ when it is in a *stable* state, from which no internal activity can occur, in which each event $a \in A$ is available and no other events are available in this state. Each A_i in (4) is a *generalised acceptance*, being either: a set of visible events, indicating that the process was observed to stably accept this set of events at this point; or the special symbol $\bullet$, which is used to indicate that no stability was observed at this point even if the process being observed actually stabilised (*i.e.* reached a stable state) here.[2] When $A_i \neq \bullet$ we of course have that $a_i \in A_i$, for all $0 \leq i \leq n-1$.

[2] We refer the reader to Roscoe [Ros09, Section 3.1] for a justification of why standard denotational models of CSP purposefully avoid positively observing instability.

Consider the process P where

$$P = a \to STOP \mathbin{\Box} b \to STOP.$$

Initially, P can accept a and b, *i.e.* P can accept the set of events $\{a, b\}$. Hence, $\langle\{a, b\}\rangle \in \mathcal{FL}[\![P]\!]$. From its initial state, P can perform a and then deadlock (*i.e.* accept no events, and so accept $\{\}$). Hence, $\langle\{a, b\}, a, \{\}\rangle \in \mathcal{FL}[\![P]\!]$. P can also perform b from its initial state and then deadlock, so $\langle\{a, b\}, b, \{\}\rangle \in \mathcal{FL}[\![P]\!]$ too. $\mathcal{FL}[\![P]\!]$ also contains all behaviours in which some of the acceptances in the above behaviours are replaced by $\bullet$, since any observer who observes P performing the above behaviours is not required to observe P's stability at any time. $\mathcal{FL}[\![P]\!]$ contains no other behaviours.

On the other hand, if we define Q so that

$$Q = a \to STOP \sqcap b \to STOP,$$

then Q can initially accept just a, or it can initially accept just b, but it cannot initially accept both events at once. Hence, $\mathcal{FL}[\![Q]\!]$ contains the sequences $\langle\{a\}\rangle$, $\langle\{b\}\rangle$, $\langle\{a\}, a, \{\}\rangle$, $\langle\{b\}, b, \{\}\rangle$ and all sequences in which some acceptances in one of these sequences is replaced by $\bullet$.

Finally, consider R defined as

$$R = a \to STOP \rhd b \to STOP.$$

From its initial state, R can perform the event a. However, recall that a is only unstably available here because R can also perform some internal activity from this state to reach a (stable) state in which only b is available. Hence, a cannot be observed to occur from a stable state and so the only behaviours of $\mathcal{FL}[\![R]\!]$ in which a occurs are $\langle\bullet, a, \{\}\rangle$ and $\langle\bullet, a, \bullet\rangle$. $\mathcal{FL}[\![R]\!]$ also contains the behaviour $\langle\{b\}\rangle$ since R can be observed to initially enter a stable state from which just b is available. Besides those behaviours already listed, $\mathcal{FL}[\![R]\!]$ contains just $\langle\bullet\rangle$ and those behaviours obtained by replacing zero or more acceptances in $\langle\{b\}, b, \{\}\rangle$ by $\bullet$.

Letting F abbreviate $\mathcal{FL}[\![P]\!]$ for some process P, we have that F satisfies the following healthiness conditions [Ros08], customarily presented as axioms. Here, s and t denote sequences of the form of (4), including both those that end in a generalised acceptance A_i as well as those that end in an event a_i as necessary; a denotes an event from Σ and A denotes a generalised acceptance.

FL0. F is non-empty: specifically $\langle\bullet\rangle \in F$.
FL1. F is prefix-closed: if $s\hat{\ }t \in F$ and s ends in a generalised acceptance, then $s \in F$.
FL2. F is closed under observing less stability: if $s\hat{\ }\langle A\rangle\hat{\ }t \in F$, then $s\hat{\ }\langle\bullet\rangle\hat{\ }t \in F$ too.
FL3. All proper acceptances can be realised: if $s\hat{\ }\langle A\rangle \in F$ and $A \neq \bullet$, then, for all $a \in A$, $s\hat{\ }\langle A, a, \bullet\rangle \in F$.

For a divergence-free process P, we have that P can always stabilise, so

$$\forall\, s \in \mathcal{FL}[\![P]\!] \,.\, \forall\, t \,.\, s = t\hat{\ }\langle\bullet\rangle \Rightarrow \exists\, A \subseteq \Sigma \,.\, t\hat{\ }\langle A\rangle \in \mathcal{FL}[\![P]\!]. \tag{5}$$

Let $\mathcal{M}$ be a finite linear observation model, where necessarily $\mathcal{M} \preceq \mathcal{FL}$, and let k denote the number of different kinds of behaviour recorded by $\mathcal{M}$, so that each process P is represented in $\mathcal{M}$ by the k sets $\mathcal{M}_1[\![P]\!], \ldots, \mathcal{M}_k[\![P]\!]$. Then $\mathcal{M}$ can be defined in terms of k relations, $r_1, \ldots, r_k$, each of whose domain is the set of all behaviours that $\mathcal{FL}$ might record of any CSP process, such that for any CSP process P, $\mathcal{M}_i[\![P]\!]$ is the relational image of $\mathcal{FL}[\![P]\!]$ under the relation r_i [Ros09], *i.e.*

$$\mathcal{M}_i[\![P]\!] = \{t \mid s \in \mathcal{FL}[\![P]\!] \wedge s \; r_i \; t\}.$$

Consider the stable-failures model [Ros97, Section 8.4]. This model records two kinds of behaviours, namely traces and *stable-failures*. A process P is therefore represented in this model by two sets, which are usually denoted $\mathit{traces}(P)$ and $\mathit{failures}(P)$ respectively. $\mathit{traces}(P)$ is just P's representation in the traces model $\mathcal{T}$ and is simply the set of all sequences of visible events from Σ that P can perform. $\mathit{failures}(P)$ contains P's stable-failures, each of which is a pair (s, X) that represents that P can perform the trace of events s and reach a stable state in which none of the events in the set $X \subseteq \Sigma$ can be performed. We may define $\mathit{traces}(P)$ as the relational image of $\mathcal{FL}[\![P]\!]$ under the function tr, which gives the sequence of events performed in any finite linear observation (which is necessarily of the form (4)) so that

$$\mathit{tr}(\langle A_0, a_0, A_1, a_1, \ldots, A_{n-1}, a_{n-1}, A_n\rangle) = \langle a_0, a_1, \ldots, a_{n-1}\rangle. \tag{6}$$

$\mathit{traces}(P)$ is then the relational image of $\mathcal{FL}[\![P]\!]$ under tr, *i.e.*

$$\mathit{traces}(P) = \{\mathit{tr}(s) \mid s \in \mathcal{FL}[\![P]\!]\}. \tag{7}$$

We may do similarly to obtain $\mathit{failures}(P)$ from each member of $\mathcal{FL}[\![P]\!]$ by defining the relation f that, for each behaviour $s \in \mathcal{FL}[\![P]\!]$, relates it to every stable-failure of P that can be observed when P is observed to perform s. For ease of notation, we define the relation f in the form of a function such that, for each element s of its domain, $f(s)$ is the set of elements in f's codomain that are related to s under f.

$$\begin{aligned} f(s \hat{\ } \langle \bullet \rangle) &= \{\}, \\ f(s \hat{\ } \langle A_n \rangle) &= \{(tr(s), X) \mid X \subseteq \Sigma - A_n\}. \end{aligned} \tag{8}$$

Note that all behaviours s that end in a $\bullet$-acceptance are related to no stable-failures. This is because, when P is observed to exhibit such an s, P is not observed to stabilise after performing the sequence of events $tr(s)$, and so no stable-failure can be observed here. With f suitably defined, $\mathit{failures}(P)$ is simply the relational image of $\mathcal{FL}[\![P]\!]$ under f, *i.e.*

$$\mathit{failures}(P) = \bigcup \{f(s) \mid s \in \mathcal{FL}[\![P]\!]\}. \tag{9}$$

Let $G(\_)$ be a CSP context. Then it is the case that for any process P, for each behaviour $s \in \mathcal{FL}[\![G(P)]\!]$, there exists a finite set Φ of behaviours from $\mathcal{FL}[\![P]\!]$ that give rise to s, such that for all processes Q, $\Phi \subseteq \mathcal{FL}[\![Q]\!] \Rightarrow s \in \mathcal{FL}[\![G(Q)]\!]$ [Ros09, Ros08]. Let $\mathcal{M}$ be a finite linear observations model, so $\mathcal{M} \preceq \mathcal{FL}$, that records k different kinds of behaviours and let t be an arbitrary behaviour in some $\mathcal{M}_i[\![G(P)]\!]$ for $1 \leq i \leq k$. Because $\mathcal{M}_i[\![G(P)]\!]$ can be defined as the relational image of $\mathcal{FL}[\![G(P)]\!]$ under some relation r_i, the presence of t in $\mathcal{M}_i[\![G(P)]\!]$ can be inferred from the presence of a single corresponding behaviour $s \in \mathcal{FL}[\![G(P)]\!]$, where $s\ r_i\ t$, such that for all processes Q, $t \in \mathcal{M}_i[\![G(Q)]\!]$ whenever $s \in \mathcal{FL}[\![G(Q)]\!]$. Combining this with the previous result, we have that for all CSP processes P, CSP contexts $G(\_)$, finite linear observations models $\mathcal{M}$ and all $i \in \{1, \ldots, k\}$, where k denotes the number of different kinds of behaviours recorded by $\mathcal{M}$,

$$\forall t \in \mathcal{M}_i[\![G(P)]\!] \mathbin{.} \exists \Phi \subseteq \mathcal{FL}[\![P]\!] \mathbin{.} \forall Q \mathbin{.} \Phi \subseteq \mathcal{FL}[\![Q]\!] \Rightarrow t \in \mathcal{M}_i[\![G(Q)]\!]. \tag{10}$$

This result becomes important later on.

Observe that our characterisation of the stable-failures model $\mathcal{F}$ in (7) and (9) above, in terms of the model $\mathcal{FL}$, is necessarily dependent on the value of Σ, the set of all visible events. In particular, because the function f above depends on Σ, the value of $\mathit{failures}(P)$ depends on Σ since different values for Σ yield different values for $\mathit{failures}(P)$. One might say, then, that each model $\mathcal{M}$, for which $\mathcal{M} \preceq \mathcal{FL}$, has not just a single interpretation in terms of $\mathcal{FL}$ but actually has an interpretation in terms of $\mathcal{FL}$ for every Σ.

For any denotational CSP model $\mathcal{M}$, if one extends the set Σ by adding new events, processes that were equivalent in $\mathcal{M}$ before Σ was extended remain equivalent afterwards [Ros09, Ros08]. This fact will become important later on, when we will adopt a common technique for proving that certain predicates can be expressed in terms of refinement checks that involves extending the set Σ of visible events.

Finally, for two processes P and Q, when $\mathit{traces}(P) = \mathit{traces}(Q)$, we say that P and Q are *traces-equivalent* (*i.e.* their representations are equivalent in the traces model $\mathcal{T}$) and write $P \equiv_{\mathcal{T}} Q$.

2.4. More Elaborate Kinds of Model and Standard Models that FDR might Reasonably Support

At least three other hierarchies of models exist, besides the hierarchy of finite linear observations models, that each mirror the structure of the hierarchy of finite linear observations models. The failures-divergences model [BR85] belongs to one of these other hierarchies. Each of the models in one of these other hierarchies extends a corresponding model from the hierarchy of finite linear observations models by recording not only finite behaviours but also some kind of infinite behaviours.

The failures-divergences model $\mathcal{F}^{\Downarrow}$ (traditionally denoted $\mathcal{N}$) is part of the hierarchy of *divergence-strict* models. Given a finite linear observations model $\mathcal{M}$, one can obtain its divergence-strict counterpart $\mathcal{M}^{\Downarrow}$ by augmenting $\mathcal{M}$ so that it also records those finite incomplete behaviours after which a process may diverge (called *divergences*), under the assumption that once a process can diverge it can do anything at all [Ros08]. The failures-divergences model $\mathcal{F}^{\Downarrow}$ is the divergence-strict counterpart of the stable-failures model $\mathcal{F}$, and, besides recording failures, records those traces after which a process may diverge. $\mathcal{FL}^{\Downarrow}$ is the divergence-strict counterpart of $\mathcal{FL}$. As well as recording the finite linear observations of a process of form (4), it also records a second set of finite linear observations, which are those directly after which the process can diverge. This second set of divergences contains finite linear observations of form (4), each of whose final acceptance A_n is necessarily $\bullet$, since a process does not, and so cannot be observed to, stabilise when it diverges.

Besides having a divergence-strict counterpart $\mathcal{M}^{\Downarrow}$, each finite linear observations model $\mathcal{M}$ also has a *non-divergence-strict divergence-recording* counterpart $\mathcal{M}^{\#}$. $\mathcal{M}^{\#}$ is similar to $\mathcal{M}^{\Downarrow}$ except that it dispenses with the divergence-strict assumption, *i.e.* it does not assume that once a process can diverge, it can do anything at all.

Divergence-strict models $\mathcal{M}^{\Downarrow}$, and non-divergence-strict divergence-recording models $\mathcal{M}^{\#}$ are appropriate for reasoning only about processes whose operational semantics are finitely branching [Ros08], which includes all finitely nondeterministic processes. While other hierarchies of models exist that are capable of reasoning about processes that are not finitely branching, they offer no extra power for reasoning about processes that are finitely branching. Because FDR cannot handle processes whose operational semantics are not finitely branching, we say that the only standard denotational models for CSP that FDR might *reasonably* support are the finite linear observations models $\mathcal{M}$, divergence-strict models $\mathcal{M}^{\Downarrow}$ and non-divergence-strict divergence-recording models $\mathcal{M}^{\#}$.

For FNDF processes $P \in CSP$ and $Q \in CSP$, if $\mathcal{FL}[\![P]\!] = \mathcal{FL}[\![Q]\!]$ then P and Q will also be identified in every other standard denotational model for CSP, even those that FDR cannot reasonably support. This does not imply, however, that we can limit our attention in this paper to refinement checks expressed only in finite linear observations models. As we will see in Section 5, certain refinement-closed predicates that can be expressed as refinement checks whose left-hand side is independent of the system being analysed, require the refinement check to be performed in a model that records divergences, for instance.

When considering whether a predicate can be expressed as a refinement check, we therefore will consider all standard models of CSP that FDR might reasonably support as defined above.

2.5. The Denotational Predicates of FNDF CSP Processes

Recall that, when considering what sorts of predicates about processes can be expressed as CSP refinement checks, we restrict our attention to predicates of FNDF processes (*i.e.* those from the set *CSP*).

We may identify a predicate with the set of processes that satisfy it [Ros05a]. A process may in turn be identified by its representation in a semantic model. FNDF processes $P \in CSP$ may of course be identified, without loss of fidelity, by their representation $\mathcal{FL}[\![P]\!]$ in the model $\mathcal{FL}$. This leads to the following definition of a denotational predicate of FNDF processes.

Definition 2 (FNDF Predicate). A predicate *Pred* of FNDF processes $P \in CSP$ is a set of sets $\mathcal{FL}[\![P]\!]$.

For every (FNDF) process $P \in CSP$, $\mathcal{FL}[\![P]\!] \in Pred$ iff P satisfies the predicate that *Pred* represents. We call a denotational predicate of FNDF CSP processes, an *FNDF predicate* for short. We will often abbreviate $\mathcal{FL}[\![P]\!] \in Pred$ by $Pred(P)$.

The set **Pred** contains all FNDF predicates:

$$\mathbf{Pred} = \mathcal{P}(\{\mathcal{FL}[\![P]\!] \mid P \in CSP\}).$$

The *complement* $\overline{Pred} \in \mathbf{Pred}$ of an FNDF predicate $Pred \in \mathbf{Pred}$ is defined as

$$\overline{Pred} = \{\mathcal{FL}[\![P]\!] \mid P \in CSP \wedge \mathcal{FL}[\![P]\!] \notin Pred\}.$$

For any predicate $Pred \in \mathbf{Pred}$, we of course have that

$$\forall\, P \in CSP \,.\, \overline{Pred}(P) \Leftrightarrow \neg Pred(P).$$

Any predicate of finitely nondeterministic, divergence-free processes that is expressible in any standard denotational model for CSP is, of course, captured by some $Pred \in \mathbf{Pred}$.

Note that **Pred** contains predicates that are not refinement-closed in any CSP model $\mathcal{M}$. An example is the predicate $Pred_{\mathbf{EF}\,e}$ that asserts that the system can perform the event $e \in \Sigma$ at some point, but need not perform e during every execution[3]. $Pred_{\mathbf{EF}\,e}$ is defined as

$$Pred_{\mathbf{EF}\,e} = \{\mathcal{FL}[\![P]\!] \mid \exists\, s \in \mathcal{FL}[\![P]\!] \,.\, tr(s) \upharpoonright \{e\} \neq \langle\rangle\}. \tag{11}$$

[3] This subscript is designed to suggest the corresponding CTL [CES86] formula.

To see that $\mathit{Pred}_{\mathbf{EF}\,e}$ is not refinement-closed in any CSP model $\mathcal{M}$, it is enough to observe that $\mathit{Pred}_{\mathbf{EF}\,e}(e \rightarrow \mathit{STOP} \sqcap \mathit{STOP})$ but $\neg\mathit{Pred}_{\mathbf{EF}\,e}(\mathit{STOP})$, noting that $e \rightarrow \mathit{STOP} \sqcap \mathit{STOP} \sqsubseteq_{\mathcal{M}} \mathit{STOP}$ in every CSP model $\mathcal{M}$.

As stated earlier, in this paper we concentrate on predicates that are refinement-closed in some CSP model $\mathcal{M}$, since predicates that are otherwise cannot usually be expressed as refinement checks that are feasible for FDR to easily carry out. Lemma 3 (below) shows that any predicate that is refinement-closed in $\mathcal{M}$ is also refinement-closed in every CSP model $\mathcal{M}'$ that is finer than $\mathcal{M}$, *i.e.* for which $\mathcal{M} \preceq \mathcal{M}'$. When considering refinement-closed predicates of divergence-free processes, we may restrict our attention to those predicates that are refinement-closed in finite linear observations models $\mathcal{M}$. By Lemma 3, all such predicates are refinement-closed in $\mathcal{FL}$.

Lemma 3. Let $\mathcal{M}$ and $\mathcal{M}'$ be denotational models of CSP such that $\mathcal{M} \preceq \mathcal{M}'$ and let *Pred* be a predicate that is refinement-closed in $\mathcal{M}$. Then *Pred* is refinement-closed in $\mathcal{M}'$.

Proof. Let $\mathcal{M}$, $\mathcal{M}'$ and *Pred* be as stated and consider an arbitrary process P such that $\mathit{Pred}(P)$. Consider a process Q such that $P \sqsubseteq_{\mathcal{M}'} Q$. Then, since $\mathcal{M} \preceq \mathcal{M}'$, it follows that $P \sqsubseteq_{\mathcal{M}} Q$. Since *Pred* is refinement-closed in $\mathcal{M}$, $\mathit{Pred}(Q)$ must hold. Thus, by Definition 1, *Pred* is refinement-closed in $\mathcal{M}'$. $\square$

3. Hyperproperties: Understanding the FNDF Predicates

In this section, we show how the set **Pred** of FNDF predicates may be understood in terms of Clarkson and Schneider's *hyperproperties* [CS08, CS10] framework. Doing so sheds light on the subset of those predicates that are refinement-closed and allows us to decompose the problem of how to express such predicates as refinement checks into two smaller problems.

We first define the set of *completed linear observations* (CLO) predicates, and show that each FNDF predicate from **Pred** is equivalent to one from this set, and vice versa.

We then show that each CLO predicate may be identified with a *hyperproperty* from the class of *CLO hyperproperties*, which is a specific instantiation of Clarkson and Schneider's hyperproperties notion (explained shortly). Clarkson and Schneider distinguish two kinds of hyperproperties, namely the *safety* and *liveness* hyperproperties, and show that every hyperproperty can be expressed as the conjunction of a safety hyperproperty and a liveness hyperproperty [CS10]. We show how to adapt their definitions and results to distinguish safety CLO hyperproperties from liveness CLO hyperproperties, which allows us to define the *safety* and *liveness* FNDF predicates respectively, and show that every FNDF predicate $\mathit{Pred} \in \mathbf{Pred}$ can be expressed as the conjunction of a safety FNDF predicate Pred_S and a liveness FNDF predicate Pred_L, so that $\mathit{Pred} = \mathit{Pred}_S \cap \mathit{Pred}_L$.

When *Pred* is refinement-closed in some finite linear observations model $\mathcal{M}$, Pred_L may be replaced in this equality by its *$\mathcal{M}$-refinement-closure*, $\mathit{RC}_{\mathcal{M}}(\mathit{Pred}_L)$ (defined later), meaning that $\mathit{Pred} = \mathit{Pred}_S \cap \mathit{RC}_{\mathcal{M}}(\mathit{Pred}_L)$. Unlike Pred_L, $\mathit{RC}_{\mathcal{M}}(\mathit{Pred}_L)$ is guaranteed to be refinement-closed in $\mathcal{M}$, and is therefore likely to be easier to express as an efficient refinement check for FDR[4]. This allows us to decompose the problem of how to express refinement-closed FNDF predicates as refinement checks, into the two smaller problems of (a) how to express the safety FNDF predicates as refinement checks, and (b) how to express the FNDF predicates that are the $\mathcal{M}$-refinement-closure of liveness predicates as refinement checks. We consider these problems in turn in Sections 4 and 5 respectively.

3.1. The Completed Linear Observations Predicates

The connection between hyperproperties and the FNDF predicates is more easily seen by first converting the FNDF predicates to an intermediate representation, which we call the *completed linear observations* (CLO) predicates.

A *completed* observation of a divergence-free process is either an observation that ends in deadlock (which is necessarily finite), or one that is infinite. Any other observation is necessarily *incomplete* because, following such an observation, the process being observed can always perform further visible activity (because it is divergence-free).

[4] No such transformation is required for Pred_S since, as we will prove in Section 4, all safety predicates are refinement-closed.

The observations of form (4) recorded by $\mathcal{FL}$ that end in deadlock are precisely those in which the final acceptance is the empty set, indicating that the process has reached a stable state from which no events can occur and so has deadlocked. These are all behaviours of the form

$$\langle A_0, a_0, A_1, a_1, \ldots, A_{n-1}, a_{n-1}, \{\}\rangle. \tag{12}$$

Let *DLO* denote the set of all such *deadlocked linear observations.*

As hinted at earlier, while $\mathcal{FL}$ doesn't explicitly record infinite behaviours, for finitely nondeterministic processes P, the finite behaviours in $\mathcal{FL}[\![P]\!]$ contain enough information to allow one to accurately infer P's infinite visible behaviours [Ros08, Section 3]. These infinite visible behaviours are naturally of the form

$$\langle A_0, a_0, A_1, a_1, \ldots, A_n, a_n \ldots\rangle \tag{13}$$

Let *ILO* denote the set of all such *infinite linear observations.* Then we can define the set, denoted $\mathcal{I}[\![P]\!]$, containing P's infinite linear observations of form (13) as the *closure* of $\mathcal{FL}[\![P]\!]$ [Ros08, Section 3]:

$$\mathcal{I}[\![P]\!] = \{s \in \mathit{ILO} \mid \forall\, t < s \,.\, t \in \mathcal{FL}[\![P]\!]\}. \tag{14}$$

Let $\mathit{CLO} = \mathit{DLO} \cup \mathit{ILO}$ be the set of *completed linear observations.* Then we may define the completed linear observations $\mathcal{C}[\![P]\!]$ that an FNDF process $P \in \mathit{CSP}$ may exhibit as follows.

$$\mathcal{C}[\![P]\!] = (\mathcal{FL}[\![P]\!] \cup \mathcal{I}[\![P]\!]) \cap \mathit{CLO}.$$

It turns out that an FNDF process $P \in \mathit{CSP}$ may be characterised equally well by $\mathcal{C}[\![P]\!]$ as it is by $\mathcal{FL}[\![P]\!]$. The following lemma proves this.

Lemma 4. For any FNDF processes P and Q,

$$(\mathcal{FL}[\![P]\!] = \mathcal{FL}[\![Q]\!]) \Leftrightarrow (\mathcal{C}[\![P]\!] = \mathcal{C}[\![Q]\!]).$$

Proof. Suppose we have two FNDF processes P and Q. We prove $(\mathcal{FL}[\![P]\!] \neq \mathcal{FL}[\![Q]\!]) \Leftrightarrow (\mathcal{C}[\![P]\!] \neq \mathcal{C}[\![Q]\!])$.

We begin with the reverse implication. Suppose $\mathcal{C}[\![P]\!] \neq \mathcal{C}[\![Q]\!]$ and, without loss of generality, that $\mathcal{C}[\![P]\!]$ contains a behaviour s that is absent from $\mathcal{C}[\![Q]\!]$. If s is a finite behaviour, then s ends in deadlock and so is present in $\mathcal{FL}[\![P]\!]$ but not $\mathcal{FL}[\![Q]\!]$. In this case, $\mathcal{FL}[\![P]\!] \neq \mathcal{FL}[\![Q]\!]$. On the other hand, if s is an infinite behaviour, then s being absent from $\mathcal{FL}[\![Q]\!]$ but not from $\mathcal{FL}[\![P]\!]$ implies that some finite prefix t of s must be present in $\mathcal{FL}[\![P]\!]$ but not $\mathcal{FL}[\![Q]\!]$. Hence, again, $\mathcal{FL}[\![P]\!] \neq \mathcal{FL}[\![Q]\!]$.

We now show the forwards implication. Suppose $\mathcal{FL}[\![P]\!] \neq \mathcal{FL}[\![Q]\!]$ and, without loss of generality, suppose that $\mathcal{FL}[\![P]\!]$ contains a behaviour s that $\mathcal{FL}[\![Q]\!]$ does not. Then if $s \in \mathit{CLO}$ (*i.e.* s ends in deadlock) the claim follows trivially. On the other hand, if $s \notin \mathit{CLO}$ and, so $\mathcal{FL}[\![P]\!] \cap \mathit{CLO} = \mathcal{FL}[\![Q]\!] \cap \mathit{CLO}$, then $s \in \mathcal{FL}[\![P]\!] - \mathit{CLO}$. So let $s = \langle A_0, a_0, A_1, a_1, \ldots, A_{n-1}, a_{n-1}, A_n\rangle$ for some $n \geq 0$. We have that $A_n \neq \{\}$.

We claim that s must have an infinite extension $t \in \mathcal{I}[\![P]\!]$ for which $s < t$. Suppose otherwise for a contradiction. Then because P is divergence-free, by Axiom **FL3** and (5), it must be the case that P inevitably deadlocks when (or following when) s is observed. This implies that P has some deadlock observation $u \in \mathcal{FL}[\![P]\!]$ for which: $s < u$ or, if $\mathit{last}(s) = \bullet$, u is identical to s except that $\mathit{last}(u) = \{\}$. However, this implies that $u \in \mathcal{FL}[\![Q]\!]$ since $\mathcal{FL}[\![P]\!] \cap \mathit{CLO} = \mathcal{FL}[\![Q]\!] \cap \mathit{CLO}$. By Axiom **FL1** or **FL2**, this implies $s \in \mathcal{FL}[\![Q]\!]$. This is a clear contradiction. Hence, s must have an infinite extension $t \in \mathcal{C}[\![P]\!]$.

So let t be an infinite extension of s. Because $s \notin \mathcal{FL}[\![Q]\!]$, by Axiom **FL1**, $\forall\, v \,.\, s < v \Rightarrow v \notin \mathcal{FL}[\![Q]\!]$. Hence, because $s < t$, $t \notin \mathcal{I}[\![Q]\!]$. So $t \notin \mathcal{C}[\![Q]\!]$ but $t \in \mathcal{C}[\![P]\!]$. Hence, $\mathcal{C}[\![P]\!] \neq \mathcal{C}[\![Q]\!]$ as required. □

It is natural, therefore, to consider the set of *CLO predicates*, defined as follows.

Definition 5 (CLO Predicate). A *CLO predicate* is a set Pred' of sets $\mathcal{C}[\![P]\!]$, where $P \in \mathit{CSP}$.

As for FNDF predicates, for every (FNDF) process $P \in \mathit{CSP}$, $\mathcal{C}[\![P]\!] \in \mathit{Pred}'$ iff P satisfies the predicate that Pred' represents.

It follows from Lemma 4 that every FNDF predicate *Pred* is *equivalent* to some CLO predicate Pred' and vice-versa, in the sense that

$$\forall\, P \in \mathit{CSP} \,.\, \mathcal{FL}[\![P]\!] \in \mathit{Pred} \Leftrightarrow \mathcal{C}[\![P]\!] \in \mathit{Pred}'. \tag{15}$$

Theorem 6. The relation defined by the notion of equivalence from (15) is a bijection.

3.2. Hyperproperties

By Theorem 6, we may replace any FNDF predicate *Pred* by the equivalent CLO predicate Pred' and vice-versa. We now show that every CLO predicate and, hence every FLO predicate, is equivalent to a unique member of the class of *CLO hyperproperties*, and vice-versa.

Clarkson and Schneider recently introduced the notion of a *hyperproperty* [CS08, CS10] to capture a broad class of security predicates, across a range of formalisms, that includes traditional safety and liveness properties, as well as information flow predicates and various other kinds of predicate. The concept of a hyperproperty is, therefore, necessarily very general; its general definition is made meaningful by instantiating it within a particular semantic framework. Here, we present the general definitions and then show how they may be instantiated in the context of completed linear observations. When instantiated this way, each hyperproperty corresponds to a unique CLO predicate and vice-versa.

In the general definition of a hyperproperty, a system is represented by a non-empty set of infinite sequences of *states* σ. We call each of these sequences an *execution* of the system. Any completed finite execution of a system that ends in some state σ is represented by the infinite sequence obtained from the finite one by infinitely stuttering the final state σ. The set of all such infinite-length executions is denoted Ψ_{inf}. The set of all partial (incomplete) executions, which are finite sequences of states σ, is denoted Ψ_{fin}.

Certain constraints may be imposed on the representation of a valid system. Hence, the set *Rep* denotes the set containing all valid system representations. Each member of *Rep* is therefore a non-empty set of infinite executions that represents a valid system.

A *hyperproperty HProp* for system representation *Rep* is then a set of systems from *Rep*. Each system from *Rep* is present in *HProp* iff it satisfies the predicate that *HProp* represents. The set of all hyperproperties for system representation *Rep* is then $\mathcal{P}(\mathit{Rep})$. The hyperproperty *true* (which every system satisfies) is of course *Rep* and the hyperproperty *false* (which no system satisfies) is $\{\}$.

Each CLO predicate *Pred* is trivially mapped onto a corresponding hyperproperty by mapping each CLO $s \in \mathcal{C}[\![P]\!] \in \mathit{Pred}$ onto a corresponding infinite execution. Each state σ of an infinite execution is either:

- a pair (A, a) where $a \in \Sigma$ and $(A \subseteq \Sigma \land a \in A) \lor A = \bullet$, or
- the symbol dl (for "deadlock").

Then each CLO is mapped onto an infinite sequences of states σ as follows.

- A deadlocked observation $\langle A_0, a_0, A_1, a_1, \ldots, A_{n-1}, a_{n-1}, \{\}\rangle$ corresponds to the infinite execution $\langle (A_0, a_0), (A_1, A_1), \ldots, (A_{n-1}, a_{n-1}), \mathsf{dl}, \mathsf{dl}, \ldots\rangle$, in which the final dl state is stuttered infinitely, and vice-versa.
- An infinite observation $\langle A_0, a_0, A_1, a_1, \ldots, A_n, a_n \ldots\rangle$ corresponds to the infinite execution $\langle (A_0, a_0), (A_1, a_1), \ldots, (A_n, a_n), \ldots\rangle$, and vice-versa.

For each FNDF process $P \in \mathit{CSP}$, let $\mathcal{E}[\![P]\!]$ denote the unique set of infinite executions σ that corresponds to $\mathcal{C}[\![P]\!]$ in this way. The set *Rep* of valid system representations is then defined as

$$\mathit{Rep} = \{\mathcal{E}[\![P]\!] \mid P \in \mathit{CSP}\}.$$

Hence, every CLO predicate *Pred* corresponds to a hyperproperty for this system representation *Rep* and vice-versa. We call such a hyperproperty a *CLO-hyperproperty.* By Theorem 6, every FNDF predicate corresponds to a unique CLO hyperproperty and vice-versa.

3.3. Safety Hyperproperties: Capturing the Safety Predicates

Clarkson and Schneider distinguish two kinds of hyperproperty, namely the *safety* and *liveness* ones. The definition of each kind mirrors the standard definitions for safety and liveness properties [AS85]. Intuitively, a safety property asserts that something (bad) never occurs, while a liveness property asserts that something (good) must occur [Lam77, AS85]. Liveness properties have the characteristic that any incomplete observation of a system can always be extended so as to satisfy any liveness property [VVK05, AFK88, AL91]. Each of these ideas are captured in the context of hyperproperties as follows.

Clarkson and Schneider define the set *Obs* of all *observations* that could be made of any system in a finite amount of time, while allowing the observer to restart the system at any point while it is being observed to observe multiple finite executions of the system[5]. Each observation is therefore a finite set of finite executions from Ψ_{fin}.

$$Obs = \mathcal{P}^{fin}(\Psi_{fin}),$$

where $\mathcal{P}^{fin}(X)$ denotes the set of all finite subsets of X.

Given an observation $M \in Obs$ and a set T of finite or infinite executions, we say that M is a prefix of T, written $M \leq T$, when the observation M can be made of T, *i.e.*

$$M \leq T \Leftrightarrow (\forall\, s \in M \,.\, \exists\, t \in T \,.\, s \leq t).$$

Under this definition, T can of course contain new executions not in M as one would expect.

The set $Obs(Rep)$ contains all observations that could be made of any valid system. Hence,

$$Obs(Rep) = \{M \mid M \in Obs \wedge \exists\, Sys \in Rep \,.\, M \leq Sys\}.$$

Then a *safety hyperproperty* is one that asserts that something bad can never happen. This bad thing is necessarily an observation $M \in Obs(Rep)$. Once this bad thing has occurred, the hyperproperty is violated forever; no further action by the system can undo the violation. This leads naturally to the following definition from [CS10], which parallels the standard definition for safety properties [AS85].

Definition 7 (Safety Hyperproperty for system representation Rep**).** A hyperproperty $HProp$ is a *safety hyperproperty for system representation Rep* iff

$$\begin{array}{l}\forall\, Sys \in Rep \,.\, Sys \notin HProp \Rightarrow \\ \quad \big(\, \exists\, M \in Obs(Rep) \,.\, M \leq Sys \wedge (\forall\, Sys' \in Rep \,.\, M \leq Sys' \Rightarrow Sys' \notin HProp) \,\big)\,.\end{array}$$

Observe that each finite (*i.e.* partial or deadlocked) observation of form (4) can be trivially mapped onto a finite sequence of states σ in which dl is always the last element if it is present. Each member of the set $Obs(Rep)$ of observations of valid systems then simply corresponds to a unique finite set $M \subseteq \mathcal{FL}[\![P]\!]$ of partial and deadlocked observations that can be exhibited by some FNDF process $P \in CSP$. Then some observation $M \in Obs(Rep)$ is a prefix of a system $Sys \in Rep$ iff the finite set M' of finite linear observations that corresponds to M can be exhibited by the system $Sys' \in CSP$ for which $\mathcal{C}[\![Sys']\!]$ corresponds to Sys, *i.e.*

$$M \leq Sys \Leftrightarrow M' \subseteq \mathcal{FL}[\![Sys']\!].$$

This allows us to define when some CLO predicate $Pred$ corresponds to a safety CLO hyperproperty. This occurs when

$$\begin{array}{l}\forall\, Sys \in CSP \,.\, \mathcal{C}[\![Sys]\!] \notin Pred \Rightarrow \\ \quad \big(\, \exists\, M \,.\, |M| \in \mathbb{N} \wedge M \subseteq \mathcal{FL}[\![Sys]\!] \wedge (\forall\, Sys' \in CSP \,.\, M \subseteq \mathcal{FL}[\![Sys']\!] \Rightarrow \mathcal{C}[\![Sys']\!] \notin Pred) \,\big)\,.\end{array}$$

Hence, we call such a CLO predicate a *safety* CLO predicate.

Applying Theorem 6, we may define the *safety FNDF predicates*, namely those FNDF predicates that are each equivalent, under (15), to a unique safety CLO predicate, and vice-versa. This is done as follows, recalling that we write $Pred(Sys)$ to mean $\mathcal{FL}[\![Sys]\!] \in Pred$.

Definition 8 (Safety FNDF Predicate). An FNDF predicate $Pred \in \mathbf{Pred}$ is a *safety* FNDF predicate iff

$$\begin{array}{l}\forall\, Sys \in CSP \,.\, \neg Pred(Sys) \Rightarrow \\ \quad \big(\, \exists\, M \,.\, |M| \in \mathbb{N} \wedge M \subseteq \mathcal{FL}[\![Sys]\!] \wedge (\forall\, Sys' \in CSP \,.\, M \subseteq \mathcal{FL}[\![Sys']\!] \Rightarrow \neg Pred(Sys')) \,\big)\,.\end{array}$$

Clarkson and Schneider show that every safety hyperproperty is *subset-closed*, which intuitively implies that it should be refinement-closed. Later, in Section 4, we will prove that every safety predicate is, indeed, refinement-closed.

The predicate $true = \{\mathcal{FL}[\![P]\!] \mid P \in CSP\}$ (that every process in CSP satisfies) is trivially a safety

[5] Equivalently, allowing the observer to run a finite number of copies of the system in parallel.

predicate[6]. As is the predicate *false* $= \{\}$ (that no process in *CSP* satisfies). Each may be expressed as refinement checks, as follows. We have that for all processes $P \in CSP$

$$\mathcal{FL}[\![P]\!] \in true \Leftrightarrow CHAOS_\Sigma \sqsubseteq_{\mathcal{FL}} P.$$

Similarly for *false*, recalling that $\mathbf{div} \notin CSP$, we have that for all processes $P \in CSP$,

$$\mathcal{FL}[\![P]\!] \in false \Leftrightarrow \mathbf{div} \sqsubseteq_{\mathcal{FL}} P,$$

since $\mathcal{FL}[\![\mathbf{div}]\!] = \{\langle\bullet\rangle\}$ and, since P is divergence-free, P must be able to initially reach a stable state, so for some $A \subseteq \Sigma$, $\langle A\rangle \in \mathcal{FL}[\![P]\!]$, but $\langle A\rangle \notin \mathcal{FL}[\![\mathbf{div}]\!]$. Hence, for all processes $P \in CSP$, this refinement check can never be satisfied and so accurately captures *false* as required.

Other predicates that one intuitively considers to express notions of safety are naturally safety predicates. Perhaps the simplest example is the predicate expressed by the refinement check in (2), which is a safety predicate because any process $P \in CSP$ violates it iff P has some behaviour $s \in \mathcal{FL}[\![P]\!]$ in which $tr(s)$ contains an A-event. Similarly, the condition of deadlock-freedom is also a safety predicate[7], because any process $P \in CSP$ that violates this condition has some behaviour $s \in \mathcal{FL}[\![P]\!]$ where $s \in DLO$, *i.e.* s is a deadlocked observation of form (12). Similar arguments can be made to show that all of Lowe's n-ary failures predicates [Low09], including all known refinement-closed noninterference [GM82] predicates (such as Lowe's RCFNDC [Low07] and Roscoe's Lazy Independence [Ros97, Section 12.4]), certain formulations of responsiveness [RSR04, Low09], determinism [Ros97, Laz99] and certain non-causation predicates [ML07], are also safety predicates.

All of the predicates mentioned in the previous two paragraphs can be expressed as CSP refinement checks of form (3), where *System* is the process to which the predicate is being applied and $\mathcal{M}$ is a finite linear observations model, *i.e.* $M \preceq \mathcal{FL}$. The following theorem proves, in fact, that all predicates that may be expressed as this sort of refinement check are safety predicates.

Theorem 9. Let *Pred* be a FNDF predicate. If there exists a specification process *Spec*, CSP context $G(\_)$ and finite linear observations model $\mathcal{M}$ such that, for all processes $P \in CSP$,

$$Pred(P) \Leftrightarrow Spec \sqsubseteq_{\mathcal{M}} G(P)$$

then *Pred* is a safety predicate.

Proof. Let *Pred* be an FNDF predicate and *Spec*, $G(\_)$ and $\mathcal{M}$ be as stated in the theorem. We show that *Pred* is a safety predicate.

There are two cases to consider. Either *Pred* is *true* (*i.e.* is satisfied by every process $P \in CSP$) or not. If *Pred* = *true* then the claim follows trivially from the arguments above. Otherwise, there exists some process $P \in CSP$ for which $\mathcal{FL}[\![P]\!] \notin Pred$. Let P be any FNDF process for which $\mathcal{FL}[\![P]\!] \notin Pred$. Then we must have that, $Spec \not\sqsubseteq_{\mathcal{M}} G(P)$. Let k be the number of different kinds of behaviour recorded by $\mathcal{M}$ and, for all processes Q, let Q's representation in $\mathcal{M}$ be given by the k sets $\mathcal{M}_1[\![Q]\!], \ldots, \mathcal{M}_k[\![Q]\!]$. Then we must have that there exists some i, where $1 \le i \le k$, and some behaviour t such that $t \in \mathcal{M}_i[\![G(P)]\!]$ but $t \notin \mathcal{M}_i[\![Spec]\!]$. From (10), we can conclude that there exists a finite set Φ, where $\Phi \subseteq \mathcal{FL}[\![P]\!]$, such that for all processes $Q \in CSP$, $\Phi \subseteq \mathcal{FL}[\![Q]\!] \Rightarrow t \in \mathcal{M}_i[\![G(Q)]\!]$, *i.e.*

$$\Phi \subseteq \mathcal{FL}[\![Q]\!] \Rightarrow Spec \not\sqsubseteq_M G(Q).$$

Let $M = \Phi$. Then $|M| \in \mathbb{N}$, $M \subseteq \mathcal{FL}[\![P]\!]$ and

$$\forall\, Q \in CSP\ .\ M \subseteq \mathcal{FL}[\![Q]\!] \Rightarrow \mathcal{FL}[\![Q]\!] \notin Pred.$$

Hence, since P is any process for which $\neg Pred(P)$, by Definition 8, *Pred* is a safety predicate. □

This result implies that every property expressible in Lowe's *bounded positive* fragment [Low08] of LTL, which excludes the *eventually* and *until* operators and restricts the use of negation, is also a safety predicate, since Lowe has shown that all such properties can be expressed as refinement checks of the form $Spec \sqsubseteq_{\mathcal{RT}} P$, where P is the system being analysed and $\mathcal{RT}$ recall is CSP's refusal-traces model [Muk93], which is a finite linear observations model. This resonates with earlier results regarding which LTL formulae represent safety properties [Sis94, Lat03].

[6] As we will see later it is also a liveness predicate.
[7] Like *true*, deadlock-freedom also turns out to be liveness predicate as well.

We conjecture that, in practice, the majority of predicates that have been tested using CSP refinement assertions in FDR are safety predicates. This is because one very often performs CSP refinement assertions in finite linear observations models like $\mathcal{T}$, $\mathcal{F}$ or $\mathcal{RT}$. The use of more elaborate models, such as those that record divergence, is very often restricted to checking that some system is divergence-free before going on to test more interesting predicates via refinement checks in finite linear observations models. There are, of course, examples where this has not been the case, notably refinement checks that have been used to test certain liveness properties, *e.g.* [Ros05a, Section 5] and [Low08, Section 6.2].

3.4. Liveness Hyperproperties: Capturing the Liveness Predicates

A *liveness hyperproperty* is one such that any incomplete observation can always be extended so as to satisfy it. This is captured by the following definition [CS10], which parallels the standard definition for liveness [AS85].

Definition 10 (Liveness Hyperproperty for system representation *Rep*). A hyperproperty *HProp* is a *liveness hyperproperty for system representation Rep* iff

$$\forall M \in Obs(Rep) \,.\, \exists\, Sys' \in Rep \,.\, M \leq Sys' \wedge Sys' \in HProp.$$

Note that this definition allows M to contain observations that end in deadlock. However, observe that any such $M \in Obs(Rep)$ cannot be extended so as to satisfy the liveness property that asserts that in every behaviour, some event e must occur infinitely often (*i.e.* which might be written in LTL [Pnu77] as $\Box \Diamond e$). Hence, in order to ensure that $\Box \Diamond e$ is a liveness CLO hyperproperty, we need to restrict our attention to those M in the above definition that contain only executions that correspond to those *partial* finite observations, in which deadlock has not been observed.

Let PLO denote the set of *partial linear observations*, which contains all behaviours of form (4) that are not present in DLO, the set of all linear observations of form (12) ending in deadlock. Then any CLO predicate corresponds to a liveness hyperproperty under the restriction introduced in the previous paragraph iff

$$\begin{aligned}&\forall M \,.\, \forall Sys \in CSP \,.\, |M| \in \mathbb{N} \wedge M \subseteq \mathcal{FL}[\![Sys]\!] \cap PLO \Rightarrow \\ &\quad \exists\, Sys' \in CSP \,.\, M \subseteq \mathcal{FL}[\![Sys']\!] \wedge \mathcal{C}[\![Sys']\!] \in Pred.\end{aligned} \tag{16}$$

Naturally, we call such CLO predicate, a *liveness* CLO predicate.

This leads straightforwardly to the following definition of a *liveness* FNDF predicate.

Definition 11 (Liveness FNDF Predicate). An FNDF predicate $Pred \in \mathbf{Pred}$ is a *liveness* FNDF predicate iff

$$\begin{aligned}&\forall M \,.\, \forall Sys \in CSP \,.\, |M| \in \mathbb{N} \wedge M \subseteq \mathcal{FL}[\![Sys]\!] \cap PLO \Rightarrow \\ &\quad \exists\, Sys' \in CSP \,.\, M \subseteq \mathcal{FL}[\![Sys']\!] \wedge Pred(Sys').\end{aligned}$$

Under this definition, the LTL property $\Box \Diamond e$ is a liveness predicate, as is the weaker property $\Diamond e$ that asserts that e must eventually occur in every execution, as well as the other LTL properties involving the $\Diamond$ and the *until* operators that are normally considered to express so-called "unbounded" liveness conditions. The property of deadlock-freedom (which, recall, is a safety predicate) is also a liveness FNDF predicate, since any set of partial observations can always be extended so that deadlock need never occur.

$\Diamond e$ and $\Box \Diamond e$ may each be expressed as refinement checks in the failures-divergences model $\mathcal{F}^{\Downarrow}$. For instance, we have that [Low08] for all processes $P \in CSP$

$$Pred_{\Diamond e}(P) \Leftrightarrow e \rightarrow \mathbf{div} \sqsubseteq_{\mathcal{F}^{\Downarrow}} P \setminus (\Sigma - \{e\}), \tag{17}$$

where, for any LTL formula ϕ, $Pred_{\phi}$ denotes the predicate that expresses ϕ.[8] This refinement check fails iff P can either: (1) perform an infinite sequence of non-e events, in which case $P \setminus (\Sigma - \{e\})$ will be able to diverge initially which the specification $e \rightarrow \mathbf{div}$ cannot; or (2) deadlock before performing e, in which case $P \setminus (\Sigma - \{e\})$ will be able to stably refuse e initially which the specification cannot. Similarly, letting

[8] This idea is formalised later on in Section 5.3.

$\mathit{Repeats}_e = e \to \mathit{Repeats}_e$ be the process that continually performs the event e, for all processes $P \in \mathit{CSP}$ we have that [Low08]

$$\mathit{Pred}_{\Box\Diamond e}(P) \Leftrightarrow \mathit{Repeats}_e \sqsubseteq_{\mathcal{F}^{\Downarrow}} P \setminus (\Sigma - \{e\}). \tag{18}$$

Equations (17) and (18) of course imply that these LTL properties are refinement-closed in $\mathcal{F}^{\Downarrow}$ and, when applied to divergence-free processes, are also refinement-closed in $\mathcal{F}$ and, hence, $\mathcal{FL}$ as well.

While we stated earlier that all safety predicates are refinement-closed[9], the same is not true of liveness predicates. In particular, the predicate $\mathit{Pred}_{\mathbf{EF}\,e}$ defined in (11), which asserts that the event e occurs during some execution of the system, is not refinement-closed, as noted earlier; however, it is a liveness predicate.

3.5. Decomposing FNDF Predicates

Clarkson and Schneider show that every hyperproperty for system representation *Rep* can be expressed as the intersection of a safety and liveness hyperproperty for *Rep* respectively. This parallels the well-known analogue of this result for safety and liveness properties [AS85]. Theorem 12 is a straightforward adaptation of their result, and states that every FNDF predicate *Pred* can be expressed as the intersection of a safety FNDF predicate Pred_S and a liveness FNDF predicate Pred_L.

Theorem 12. Let *Pred* be an FNDF predicate from **Pred**. Then there exists a safety FNDF predicate $\mathit{Pred}_S \in \mathbf{Pred}$ and a liveness FNDF predicate $\mathit{Pred}_L \in \mathbf{Pred}$ such that

$$\mathit{Pred} = \mathit{Pred}_S \cap \mathit{Pred}_L$$

The proof of this result is a direct adaptation of that for Theorem 5 from [CS10, Appendix D]. Because of its similarity to the proof from which it was adapted, it has been omitted from the main body of this paper and appears in Appendix A.

We would like to apply this result to allow us to decompose the problem of expressing an arbitrary FNDF predicate *Pred* as a refinement-check, into two smaller problems: one involving expressing Pred_S (or some adaptation of it) as a refinement-check and the other involving expressing Pred_L (or some adaptation of it) as a refinement check, such that the conjunction of the two checks can be used to test *Pred*. The left-hand-side of each of these hypothetical refinement checks must be independent of the process to which it is being applied, in order for it to be practical. Hence, the predicate that each refinement check tests for must be refinement-closed in some CSP model that FDR might support, and so (when applied only to FNDF processes) must be refinement-closed in some finite linear observations model $\mathcal{M}$.

As we will prove later in Section 4, all safety predicates Pred_S are naturally refinement-closed. However, the same is not true for all liveness predicates Pred_L, as shown earlier. It turns out, though, that when *Pred* is refinement-closed in some finite linear observations model $\mathcal{M}$, we may replace Pred_L in Theorem 12 by a variation of Pred_L that is refinement-closed in $\mathcal{M}$. This variation is known as Pred_L's *$\mathcal{M}$-refinement-closure* and is simply the predicate that holds for a process P when Pred_L holds for all of P's refinements, under $\sqsubseteq_{\mathcal{M}}$ [Low07]. This idea is defined formally as follows.

Definition 13 ($\mathcal{M}$-Refinement-Closure of a Predicate). Let $\mathit{Pred} \in \mathbf{Pred}$ be an FNDF predicate. Then the *$\mathcal{M}$-refinement-closure* of *Pred* is the FNDF predicate $\mathit{RC}_{\mathcal{M}}(\mathit{Pred}) \in \mathbf{Pred}$ such that

$$\forall\, P \in \mathit{CSP} \,.\, \mathit{RC}_{\mathcal{M}}(\mathit{Pred})(P) \Leftrightarrow \forall\, Q \in \mathit{CSP} \,.\, P \sqsubseteq_{\mathcal{M}} Q \Rightarrow \mathit{Pred}(Q).$$

To illustrate this with an example, consider the predicate $\mathit{Pred}_{\mathbf{EF}\,e}$ from (11), which we saw earlier is not refinement-closed in any standard CSP model $\mathcal{M}$. Let $P = d \to \mathit{STOP} \sqcap e \to \mathit{STOP}$. $\mathit{Pred}_{\mathbf{EF}\,e}(P)$ holds since P clearly has an execution in which e can occur. Let $\mathcal{M}$ be a standard CSP model and consider $\mathit{RC}_{\mathcal{M}}(\mathit{Pred}_{\mathbf{EF}\,e})$, which holds for P iff $\mathit{Pred}_{\mathbf{EF}\,e}$ holds for all of P's $\mathcal{M}$-refinements. In every such model $\mathcal{M}$, $P \sqsubseteq_{\mathcal{M}} d \to \mathit{STOP}$, which clearly fails $\mathit{Pred}_{\mathbf{EF}\,e}$. Hence, it must be that $\neg \mathit{RC}_{\mathcal{M}}(\mathit{Pred}_{\mathbf{EF}\,e})(P)$.

For any predicate *Pred*, $\mathit{RC}_{\mathcal{M}}(\mathit{Pred})$ is refinement-closed in $\mathcal{M}$. Also, for any $\mathit{Pred} \in \mathbf{Pred}$, $\mathit{RC}_{\mathcal{M}}(\mathit{Pred}) \subseteq \mathit{Pred}$. When *Pred* is itself refinement-closed in $\mathcal{M}$, it is straightforward to show that $\mathit{Pred} = \mathit{RC}_{\mathcal{M}}(\mathit{Pred})$.

Theorem 14 follows from Theorem 12 and shows how we may decompose refinement-closed FNDF predicates into two predicates that are each refinement-closed.

[9] This result will be proved later in Section 4.

Theorem 14. Let *Pred* be an FNDF predicate from **Pred** that is refinement-closed in some finite linear observations model $\mathcal{M}$. Then there exists a safety FNDF predicate $Pred_S$ and a liveness FNDF predicate $Pred_L$ such that

$$Pred = Pred_S \cap RC_{\mathcal{M}}(Pred_L).$$

Proof. Let *Pred* and $\mathcal{M}$ be as stated. Applying Theorem 12, let $Pred_S$ and $Pred_L$ be safety and liveness FNDF predicates respectively, such that $Pred = Pred_S \cap Pred_L$. Then it suffices to show that $Pred_S \cap RC_{\mathcal{M}}(Pred_L) = Pred_S \cap Pred_L$. It is the case that $RC_{\mathcal{M}}(Pred_L) \subseteq Pred_L$, hence we have that $Pred_S \cap RC_{\mathcal{M}}(Pred_L) \subseteq Pred_S \cap Pred_L$. We prove the reverse containment, *i.e.* that $Pred_S \cap Pred_L \subseteq Pred_S \cap RC_{\mathcal{M}}(Pred_L)$, by contradiction.

Suppose this containment doesn't hold, *i.e.* that there exists some FNDF process P such that $\mathcal{FL}[\![P]\!] \in Pred_S \cap Pred_L$ but $\mathcal{FL}[\![P]\!] \notin Pred_S \cap RC_{\mathcal{M}}(Pred_L)$. Then it follows that $\mathcal{FL}[\![P]\!] \in Pred_S$ and $\mathcal{FL}[\![P]\!] \in Pred_L$ but $\mathcal{FL}[\![P]\!] \notin RC_{\mathcal{M}}(Pred_L)$. Then $\neg RC_{\mathcal{M}}(Pred_L)(P)$ and so P must have some $\mathcal{M}$-refinement, Q, such that $\neg Pred_L(Q)$. Hence, it must be that $\neg Pred(Q)$. Note that because $\mathcal{FL}[\![P]\!] \in Pred_S$ and $\mathcal{FL}[\![P]\!] \in Pred_L$, $Pred(P)$ holds. Because *Pred* is refinement-closed in $\mathcal{M}$ this creates a clear contradiction, as required. □

From Theorem 14, we may decompose the problem of how to express refinement-closed predicates as refinement checks, into the two separate problems of (a) how to express safety predicates as refinement checks, and (b) how to express the refinement-closures of liveness predicates as refinement checks, respectively. We consider these problems in turn in the following two sections.

4. Expressing Safety Predicates as Refinement Tests

In this section, we consider to what degree safety predicates can be expressed as CSP refinement checks. As we will see, it turns out that all safety predicates are, in fact, refinement-closed, and can be expressed in the form of CSP refinement assertions in $\mathcal{FL}$ whose left-hand-side is independent of the system being analysed. We present a constructive proof of this result.

A direct corollary of this result is that all safety predicates are refinement-closed in $\mathcal{FL}$. This result also allows us to conclude that the safety predicates, and the predicates that can be expressed as refinement checks of the form $Spec \sqsubseteq_{\mathcal{M}} G(P)$ in some finite linear observations model $\mathcal{M}$, are one and the same.

We have already shown how the safety predicates *true* and *false* can be expressed as $\mathcal{FL}$ refinement assertions. We now show that for every other safety predicate *Pred* there exists a specification process *Spec* and CSP context $G(\_)$ such that

$$\forall\, P \in CSP \;.\; Pred(P) \Leftrightarrow Spec \sqsubseteq_{\mathcal{FL}} G(P).$$

Let *Pred* be a safety predicate. We show how to construct *Spec* and $G(\_)$. In order to do so, we first need to capture *Pred* via some mathematical encoding around which we can frame the resulting refinement test. Adapting Roscoe [Ros05a, Definition 2.4 and proof of Theorem 2.1], we may capture *Pred* via a set of *refutations*. Let P be some process that fails *Pred*. Then, by Definition 8, there exists some finite set M such that $M \subseteq \mathcal{FL}[\![P]\!]$ and for all $Q \in CSP$, whenever $M \subseteq \mathcal{FL}[\![Q]\!]$, $\neg Pred(Q)$. We call M a *refutation* of *Pred* for P. Any safety predicate *Pred* may, therefore, be characterised by a set S of such finite sets M, such that for every process $P \in CSP$ for which $\neg Pred(P)$, S contains a refutation M for P, and S contains nothing that isn't a refutation for some such P.

Given such a set S, we then have

$$\forall\, P \in CSP \;.\; (Pred(P) \Leftrightarrow \forall\, M \in S \;.\; M \not\subseteq \mathcal{FL}[\![P]\!]).$$

Our strategy is therefore, given such a set of refutations S that characterises *Pred*, to build a refinement check $Spec \sqsubseteq_{\mathcal{FL}} G(P)$ that asserts the statement on the right-hand side of this bi-implication.

We begin by considering a single refutation $M \in S$ and the problem of building a refinement check to assert that $M \not\subseteq \mathcal{FL}[\![P]\!]$. We build a testing context $Test_M(\_)$ such that $Test_M(P)$ exhibits a behaviour from the set U (whose value we fix later on) iff P exhibits all behaviours in M. We then build a specification *Spec* that performs all behaviours except those in U, so that

$$Spec \sqsubseteq_{\mathcal{FL}} Test_M(P) \Leftrightarrow M \not\subseteq \mathcal{FL}[\![P]\!].$$

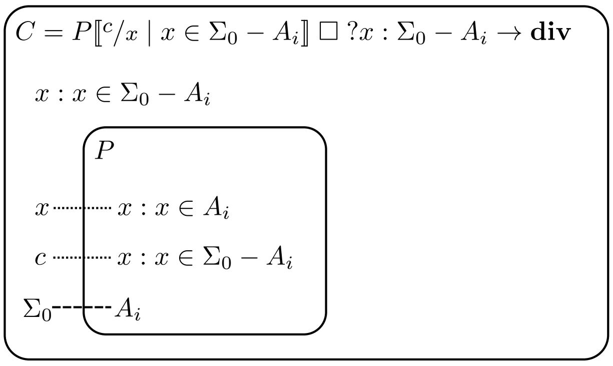
$C = P[\![{}^{c}/x \mid x \in \Sigma_0 - A_i]\!] \,\square\, ?x : \Sigma_0 - A_i \to \mathbf{div}$
$x : x \in \Sigma_0 - A_i$
P
x
$x : x \in A_i$
c
$x : x \in \Sigma_0 - A_i$
Σ_0
A_i

Fig. 1. Testing for when P stably accepts A_i: C offers all $x \in \Sigma_0 - A_i$; when P performs $x \in \Sigma_0 - A_i$, C performs c while other events of P are left unchanged; this means that C accepts Σ_0 iff P accepts A_i. Each box represents the process labelled in its top-left corner, and contains its sub-processes. Dotted lines indicate renamings and dashed lines indicate corresponding acceptances.

To test that $M \not\subseteq \mathcal{FL}[\![P]\!]$ for all $M \in S$, we can assert that no member of U is present in $\bigcup_{M \in S} \mathcal{FL}[\![\mathit{Test}_M(P)]\!]$. This holds iff

$$\mathit{Spec} \sqsubseteq_{\mathcal{FL}} \bigsqcap_{M \in S} \mathit{Test}_M(P).$$

Hence, letting $G(\_) = \bigsqcap_{M \in S} \mathit{Test}_M(\_)$, we can then arrive at our result.

Note that since S will not generally be finite, for an arbitrary FNDF process P, the process $G(P)$ will in general be infinite state even when P is itself finite. Hence, the construction we present here is not finitary, and so cannot be directly applied without any further effort to produce a practical refinement check for FDR, or similar tools, for each safety predicate. Fortunately, this is not its purpose, which is instead to demonstrate that all safety predicates can be expressed as CSP refinement checks in the model $\mathcal{FL}$.

Example. We will use the safety predicate of determinism as a running example to help explain the construction of the process $\mathit{Test}_M(\_)$. A CSP process is deterministic when it can never both perform and refuse to perform some event at the same time. Hence, each refutation for this safety predicate is a set that contains two behaviours:

1. $s \hat{} \langle \bullet, e, \bullet \rangle$, in which some event e is performed after some incomplete behaviour s, and
2. $s \hat{} \langle A \rangle$, where $e \notin A$, in which e is refused at this same point.

For illustration, when considering the example of determinism, we will use the refutation $M = \{\langle \bullet, e, \bullet \rangle, \langle \{\} \rangle\}$.

Returning to the general case, to build $\mathit{Test}_M(\_)$, it is useful to first consider just a single element $s \in M$. s is necessarily of the form (4), *i.e.* $s = \langle A_0, a_0, A_1, a_1, \ldots, A_{n-1}, a_{n-1}, A_n \rangle$. Consider any non-$\bullet$ acceptance A_i from s and the process Q that represents P after P has performed the trace of events $\langle a_0, \ldots, a_{i-1} \rangle$ that precede A_i in s. We want to build a context $T_s(\_)$ such that $T_s(Q)$ exhibits some behaviour iff Q can stably accept A_i.

We will make use of the following result, which is due to Bill Roscoe[10]. We will need to extend the set Σ of visible events. Let Σ_0 be the initial set of visible events before Σ is extended. Extend Σ by adding a fresh event c, so $\Sigma = \Sigma_0 \cup \{c\}$. Then consider $P[\![{}^{c}/x \mid x \in \Sigma_0 - A_i]\!]$. This is the process that behaves like P except that each non A_i event x is renamed to c. Then the process $C = P[\![{}^{c}/x \mid x \in \Sigma_0 - A_i]\!] \,\square\, ?x : \Sigma_0 - A_i \to \mathbf{div}$ stably accepts Σ_0 iff P stably accepts A_i: when P accepts any set that contains some event from $\Sigma_0 - A_i$, then C must accept some $Y \cup \{c\}$ for some $Y \subseteq \Sigma_0$ and vice-versa; when P accepts any set that doesn't contain all of A_i, C won't accept the entirety of Σ_0 and vice-versa. Figure 1 depicts this arrangement.

We extend this argument to show how to build a context $T_s(\_)$ such that P exhibits s iff $T_s(P)$ exhibits the related behaviour $u(s) \in U$ (where we define the function u shortly), *i.e.*

$$u(s) \in \mathcal{FL}[\![T_s(P)]\!] \Leftrightarrow s \in \mathcal{FL}[\![P]\!].$$

[10] Private communication, 4th February 2010, regarding $\mathcal{FL}$ and full abstraction.

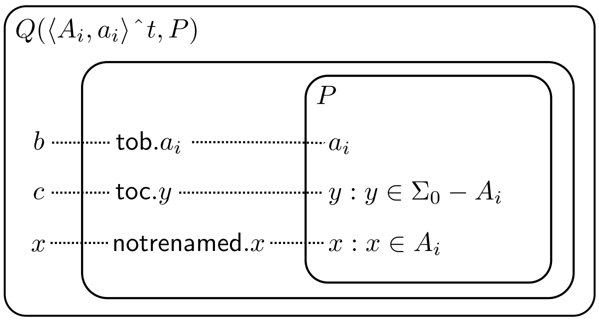

Fig. 2. Dynamic renaming: a_i is renamed to b; each event $y \in \Sigma_0 - A_i$ is renamed to c; each event $x \in A_i$ is left unchanged.

The context $Test_M(\_)$ is then defined in terms of $T_s(\_)$ as we will see.

For now, suppose all acceptances A_i in s are non-$\bullet$ ones. Then, we define $u(s)$ (for these s) as follows, where we extend Σ again with two new fresh events b and $\mathsf{endmark}$, so that $\Sigma = \Sigma_0 \cup \{b, c, \mathsf{endmark}\}$.

$$u(\langle A_i, a_i\rangle\,\hat{}\,t) = \langle \Sigma_0 \cup \{b\}, b\rangle\,\hat{}\,u(t),$$
$$u(\langle A_i\rangle) = \langle \Sigma_0 \cup \{b, \mathsf{endmark}\}, \mathsf{endmark}, \Sigma_0\rangle.$$

Notice that, when A_i is not the last acceptance in s, we have $T_s(P)$ stably accept $\Sigma_0 \cup \{b\}$ iff P stably accepts A_i (rather than having $T_s(P)$ stably accept just Σ_0 as before). We have $T_s(P)$ apply a renaming operation to P that renames a_i to itself and to the new event b. This allows the event that follows the acceptance $\Sigma_0 \cup \{b\}$ in $u(\langle A_i, a_i\rangle)$ to be b rather than a_i. This will become important later on. The event $\mathsf{endmark}$ is used to mark the point in $u(s)$ just before the final acceptance (which must be Σ_0) occurs; its purpose will become clear when we define the specification process *Spec* later on. Similarly, the purpose of offering $\Sigma_0 \cup \{b\}$ when $\mathsf{endmark}$ is offered will also become clear later on.

The context $T_s(\_)$ needs to perform the above renaming dynamically, since each A_i in s could be different from A_{i-1}. This can be achieved as follows. We further extend the set of events Σ by adding the events $\mathsf{notrenamed}.x$, $\mathsf{tob}.x$ and $\mathsf{toc}.x$ for every event $x \in \Sigma_0$. Then we construct the process $Q(s, P)$, defined as

$$Q(s,P) = \left(P[\![{}^{\mathsf{notrenamed}.x,\, \mathsf{tob}.x,\, \mathsf{toc}.x}/{}_{x,x,x} \mid x \in \Sigma_0]\!] \underset{X}{\|} R(s) \right) [\![{}^{b,\,c,\,x}/{}_{\mathsf{tob}.x,\, \mathsf{toc}.x,\, \mathsf{notrenamed}.x} \mid x \in \Sigma_0]\!],$$

where $X = \{|\mathsf{notrenamed}, \mathsf{tob}, \mathsf{toc}|\}$ and the process $R(s)$ is defined as

$$R(\langle A_i, a_i\rangle\,\hat{}\,t) = \mathsf{tob}!a_i \to R(t) \,\Box\, \mathsf{notrenamed}?x : A_i \to STOP \,\Box\, \mathsf{toc}?y : \Sigma_0 - A_i \to STOP,$$
$$R(\langle A_i\rangle) = \mathsf{notrenamed}?x : A_i \to STOP \,\Box\, \mathsf{toc}?y : \Sigma_0 - A_i \to STOP.$$

In $Q(s, P)$, a renaming is applied to P that renames every event $x \in \Sigma_0$ that P might perform, so that for each $x \in \Sigma_0$, whenever P could have performed x, $P[\![{}^{\mathsf{notrenamed}.x,\, \mathsf{tob}.x,\, \mathsf{toc}.x}/{}_{x,x,x} \mid x \in \Sigma_0]\!]$ can now perform $\mathsf{notrenamed}.x$, $\mathsf{tob}.x$ and $\mathsf{toc}.x$. The process $R(s)$ is used to control which of these events that this renamed P performs by having the renamed P synchronise with $R(s)$ on all such events. The outer renaming ensures that whenever the combination of the renamed P and $R(s)$ performs a $\mathsf{tob}.x$ or $\mathsf{toc}.x$ event, $Q(s, P)$ performs b or c respectively. When the combination performs a $\mathsf{notrenamed}.x$ event, $Q(s, P)$ performs x. In this way, when R permits a $\mathsf{tob}.x$ event to occur, P performs x but x is effectively renamed to b. The same is true for $\mathsf{toc}.x$ and c. When R permits a $\mathsf{notrenamed}.x$ event, P performs x and, effectively, no such renaming occurs (since $\mathsf{notrenamed}.x$ is renamed back to x).

At any time, R allows the renamed P to perform all $\mathsf{notrenamed}.x$ events where $x \in A_i$ and allows the renamed P to perform all $\mathsf{toc}.y$ events where $y \in \Sigma_0 - A_i$. This has the effect of allowing P to always perform all of its events from Σ_0 while renaming any from $\Sigma_0 - A_i$ to c. When A_i is followed by some event a_i, R allows the renamed P to also perform $\mathsf{tob}.a_i$. This effectively causes a_i to be renamed to itself and to b. Only if the renamed P performs $\mathsf{tob}.a_i$ (in which case P has performed a_i and this has been renamed to b), does R allow further activity. This is depicted in Figure 2.

Example. In the case of determinism, if we consider the element $\langle\{\}\rangle$ (the other will be considered shortly) from the refutation $M = \{\langle\bullet, e, \bullet\rangle, \langle\{\}\rangle\}$ chosen earlier for illustration, we see that $Q(\langle\{\}\rangle, P)$ is the process that behaves like P except that every event other than those from $\{\}$, *i.e.* every event that P might perform, is renamed to c.

Returning to the general case, to form $T_s(P)$, we then place $Q(s,P)$ in parallel with a process $O(s)$, whose job it is to offer $\Sigma_0 - A_i$ at each point other than when $\mathsf{endmark}$ is being offered, so that at these points $T_s(P)$ offers the entirety of $\Sigma_0 \cup \{b\}$ or Σ_0 (depending on whether A_i is not the final acceptance in s) iff P offers A_i. $O(s)$'s job is also to offer $\Sigma_0 \cup \{b, \mathsf{endmark}\}$ at the appropriate time and, for reasons that will become clear later on, to continue to offer $\Sigma_0 \cup \{b, \mathsf{endmark}\}$ whenever the b it is offering at one of these times gets performed. We further extend Σ by adding, for each event $x \in \Sigma_0 \cup \{b\}$, a new event $\mathsf{offer}.x$. We then define $T_s(P)$ as

$$T_s(P) = \left(Q(s,P) \underset{\Sigma_0 \cup \{b,c\}}{\parallel} O(s) \right) [\![{}^{x}/_{\mathsf{offer}.x} \mid x \in \Sigma_0 \cup \{b\}]\!],$$

where the process $O(s)$ is defined as

$$\begin{aligned} O(\langle A_i, a_i \rangle ^\frown t) &= b \to O(t) \,\square\, ?x : \Sigma_0 \cup \{c\} \to STOP \,\square\, \mathsf{offer}?y : \Sigma_0 - A_i \to STOP, \\ O(\langle A_i \rangle) &= \mathsf{endmark} \to (?x : \Sigma_0 \cup \{c\} \to STOP \,\square\, \mathsf{offer}?y : \Sigma_0 - A_i \to STOP) \,\square \\ &\quad\ \mathsf{offer}?x : \Sigma_0 \cup \{b\} \to (O(s) \mathrel{\triangleleft} x = b \mathrel{\triangleright} STOP). \end{aligned}$$

We have this process offer the events from $\Sigma_0 - A_i$ by having it perform each of them, $x \in \Sigma_0 - A_i$, as $\mathsf{offer}.x$. We do the same for the events $\Sigma_0 \cup \{b\}$. We then rename each $\mathsf{offer}.x$ to x. $O(s)$ must synchronise with $Q(s,P)$ so that it knows the point in s that P is up to. We therefore have it synchronise with $Q(s,P)$ on events from $\Sigma_0 \cup \{b,c\}$, always allowing $Q(s,P)$ to perform all such relevant events. When the final acceptance A_i in s is reached, $O(s)$ allows $T_s(P)$ to offer only $\Sigma_0 \cup \{b, \mathsf{endmark}\}$. If $\mathsf{endmark}$ is performed, it has $T_s(P)$ offer Σ_0 iff P offers A_i, in accordance with the definition of $u(s)$ above.

We have, then, that for all s that contain no $\bullet$-acceptances,

$$u(s) \in \mathcal{FL}[\![T_s(P)]\!] \Leftrightarrow s \in \mathcal{FL}[\![P]\!].$$

Example. In the case of our example element $\langle\{\}\rangle$ from the determinism refutation chosen earlier, $O(\langle\{\}\rangle)$, after performing $\mathsf{endmark}$ offers the event $\mathsf{offer}.y$, for all $y \in \Sigma_0$, as well as the other events it offers. This means that $T_{\langle\{\}\rangle}(P)$ is the process that initially offers $\langle \Sigma_0 \cup \{b, \mathsf{endmark}\}\rangle$. After it performs $\mathsf{endmark}$, it offers all events in Σ_0 but also offers the event c exactly when P offers any event from Σ_0. In other words, $T_{\langle\{\}\rangle}(P)$ exhibits the behaviour $\langle \Sigma_0 \cup \{b, \mathsf{endmark}\}, \mathsf{endmark}, \Sigma_0 \rangle = u(\langle\{\}\rangle)$ precisely when P exhibits $\langle\{\}\rangle$ as required.

Returning to the general case, we now extend this to handle $\bullet$-acceptances in the arbitrary behaviour s.

To do so, we need to decide on the value of $u(\langle \bullet, a_i \rangle)$, *i.e.* on what we want $T_s(P)$ to exhibit when P is observed to perform the event a_i but no stability is observed before a_i occurs. Note that it might not be possible for P to stabilise before a_i occurs, since a_i might be able to occur in P only from unstable states. In this case, $T_s(P)$ will not be able to stabilise before P performs a_i. We therefore complete the definition of $u(s)$ as follows, extending the set Σ further by adding the fresh event d, to obtain

$$\begin{aligned} u(\langle \bullet, a_i \rangle ^\frown t) &= \langle \bullet, d \rangle ^\frown u(t), \\ u(\langle \bullet \rangle) &= \langle \Sigma_0 \cup \{b, \mathsf{endmark}\}, \mathsf{endmark}, \Sigma_0 \rangle. \end{aligned}$$

When P exhibits $\langle \bullet, a_i \rangle$, we have $T_s(P)$ perform the event d (which will involve renaming a_i to d) but don't require $T_s(P)$ to stabilise first. We use a fresh event d here, rather than b as before, to distinguish this case from the earlier one, in which $T_s(P)$ is required to stabilise and accept the entirety of $\Sigma_0 \cup \{b\}$ before it performs b.

The final clause of u above (for $u(\langle \bullet \rangle)$) places no requirement on P, since any process can always exhibit $\langle \bullet \rangle$. P is guaranteed to stabilise, however, since it is divergence-free. We can therefore construct $T_s(P)$ here so that it stabilises, offers $\Sigma_0 \cup \{b, \mathsf{endmark}\}$, and then, after performing $\mathsf{endmark}$, offers the entirety of Σ_0 (with each of these offers being made by $O(s)$ if necessary). This is done simply to maintain uniformity with the definition of $u(\langle A_i \rangle)$ for simplicity.

We need to extend $Q(s,P)$ to allow for the extra renaming involving the new event d that is implied by the completed definition of $u(s)$ above. Therefore, we redefine $Q(s,P)$ as

$$Q(s,P) = \\ \left(P[\![{}^{\mathsf{notrenamed}.x,\, \mathsf{tob}.x,\, \mathsf{toc}.x,\, \mathsf{tod}.x}/{}_{x,\,x,\,x,\,x} \mid x \in \Sigma_0]\!] \underset{X}{\|} R(s)\right) [\![{}^{b,\,c,\,d,\,x}/{}_{\mathsf{tob}.x,\, \mathsf{toc}.x,\, \mathsf{tod}.x,\, \mathsf{notrenamed}.x} \mid x \in \Sigma_0]\!],$$

where $X = \{|\mathsf{notrenamed}, \mathsf{tob}, \mathsf{toc}, \mathsf{tod}|\}$.

$R(s)$ is then completed by defining

$$R(\langle \bullet, a_i\rangle \hat{}\, t) = \mathsf{tod}!a_i \to R(t),$$
$$R(\langle \bullet \rangle) = STOP.$$

Example. In the case of our running example for the determinism predicate, if we consider the other element $\langle \bullet, e, \bullet\rangle$ of the refutation $M = \{\langle \bullet, e, \bullet\rangle, \langle\{\}\rangle\}$ chosen earlier, we see that $Q(\langle \bullet, e, \bullet\rangle, P)$ is the process that can perform only the event d, and can do so only if P can perform the event e.

Returning to the general case, we redefine $T_s(P)$ so that $Q(s,P)$ now synchronises with $O(s)$ on d as well, obtaining

$$T_s(P) = \left(Q(s,P) \underset{\Sigma_0 \cup \{b,c,d\}}{\|} O(s)\right) [\![{}^{x}/{}_{\mathsf{offer}.x} \mid x \in \Sigma_0 \cup \{b\}]\!].$$

We then complete the definition of $O(s)$ by defining

$$O(\langle \bullet, a_i\rangle \hat{}\, t) = d \to O(t),$$
$$O(\langle \bullet \rangle) = \mathsf{endmark} \to \mathsf{offer}?x : \Sigma_0 \to STOP \,\Box\, \mathsf{offer}?x : \Sigma_0 \cup \{b\} \to (O(\langle\bullet\rangle) \not< x = b \not> STOP).$$

Notice that the second clause here has $O(s)$ offer $\{\mathsf{offer}.x \mid x \in \Sigma_0\}$ after performing $\mathsf{endmark}$, which will cause $T_s(P)$ to offer Σ_0 after performing $\mathsf{endmark}$ as required by the completed definition of $u(s)$ above.

Hence, we have that for all s,

$$u(s) \in \mathcal{FL}[\![T_s(P)]\!] \Leftrightarrow s \in \mathcal{FL}[\![P]\!].$$

Example. In the case of the example element $\langle \bullet, e, \bullet\rangle$ from the determinism refutation chosen earlier, $O(\langle \bullet, e, \bullet\rangle)$ is the process that performs d, then $\mathsf{endmark}$ and then offers the required events as defined above. This means that $T_{\langle \bullet, e, \bullet\rangle}(P)$ is the process that exhibits the behaviour $\langle \bullet, d, \Sigma_0 \cup \{b, \mathsf{endmark}\}, \mathsf{endmark}, \Sigma_0\rangle = u(\langle \bullet, e, \bullet\rangle)$ exactly when P exhibits $\langle \bullet, e, \bullet\rangle$, as required.

Returning to the general case, we can now define the entire set U that contains all behaviours $u(s)$.

$$U = \left\{ \langle A_0, a_0, \ldots, A_n, a_n\rangle \hat{}\, v \;\middle|\; \begin{array}{l} v = \langle \Sigma_0 \cup \{b, \mathsf{endmark}\}, \mathsf{endmark}, \Sigma_0\rangle \wedge n \geq -1 \wedge \\ (\forall\, 0 \leq i \leq n \,.\, (A_i = \Sigma_0 \cup \{b\} \wedge a_i = b) \vee (A_i = \bullet \wedge a_i = d)) \end{array} \right\}.$$

Note that $\langle \Sigma_0 \cup \{b, \mathsf{endmark}\}, \mathsf{endmark}, \Sigma_0\rangle \in U$, which is necessary for when s has length just 1, as it does for instance for the element $\langle\{\}\rangle$ of the determinism refutation discussed above.

We then build a specification *Spec* that roughly exhibits all behaviours other than those from U, as

$$\begin{aligned} Spec = {} & (?x : \Sigma_0 \cup \{b\} \to (Spec \not< x = b \not> CHAOS_\Sigma)) \sqcap \\ & \left(\textstyle\bigsqcap_{X \in \mathcal{P}(\Sigma - \{\mathsf{endmark}\}) - (\Sigma_0 \cup \{b\})} ?x : X \to (Spec \not< x = d \not> CHAOS_\Sigma)\right) \sqcap \\ & \left(\mathsf{endmark} \to \left(\textstyle\bigsqcap_{X \in \mathcal{P}(\Sigma) - \Sigma_0} ?x : X \to CHAOS_\Sigma\right) \Box\, ?x : \Sigma_0 \cup \{b\} \to CHAOS_\Sigma\right). \end{aligned}$$

Spec is designed so that $T_s(P)$ exhibits no behaviours from U iff $Spec \sqsubseteq_{\mathcal{FL}} T_s(P)$. *Spec* contains three clauses (one per line) separated by "$\sqcap$"s. Perhaps the most important is the last line, which allows *Spec* to, after offering $\Sigma_0 \cup \{b, \mathsf{endmark}\}$ and then performing $\mathsf{endmark}$, exhibit all stable acceptances except Σ_0. This is because the only acceptance that can follow $\mathsf{endmark}$ in any behaviour from U is Σ_0, and ensures that *Spec* never performs any behaviour from U. The previous two lines allow *Spec* to exhibit all behaviours that could be exhibited by $T_s(P)$. At any time, if $T_s(P)$ exhibits some behaviour that has no extensions in U, then *Spec* evolves to $CHAOS_\Sigma$ to allow $T_s(P)$ to behave arbitrarily from that point on.

Hence, we have that

$$s \notin \mathcal{FL}[\![P]\!] \Leftrightarrow \mathit{Spec} \sqsubseteq_{\mathcal{FL}} T_s(P).$$

Returning our attention now to the refutation M in which s is contained, we now extend this to test if P can perform all $s \in M$. Recall that $|M|$ is necessarily finite. Consider the process $\mathit{Test}_M(P)$, defined as

$$\mathit{Test}_M(P) = \underset{\Sigma_0 \cup \{b, \mathsf{endmark}\}}{\|}_{s \in M} T_s(P).$$

$\mathit{Test}_M(P)$ runs $|M|$ copies $T_s(P)$ in parallel, one for each $s \in M$. Each copy must synchronise with the others on the events in $\Sigma_0 \cup \{b, \mathsf{endmark}\}$ but may perform other events freely. This means, at any point, $\mathit{Test}_M(P)$ can stably accept $\Sigma_0 \cup \{b\}$ iff one $T_s(P)$ can stably accept $\Sigma_0 \cup \{b\}$ and the others can each stably accept $\Sigma_0 \cup \{b\}$ or $\Sigma_0 \cup \{b, \mathsf{endmark}\}$. $\mathit{Test}_M(P)$ can stably accept $\Sigma_0 \cup \{b, \mathsf{endmark}\}$ just when each $T_s(P)$ can. Since d can occur freely, $\mathit{Test}_M(P)$ can perform d whenever any $T_s(P)$ can.

This should explain why we defined $u(s)$ to have each $T_s(P)$ offer $\Sigma_0 \cup \{b\}$ whenever it can perform $\mathsf{endmark}$, and to continue to make this offer after performing b at this point. The reason is because, suppose some $T_s(P)$ has reached the point at which it can perform $\mathsf{endmark}$, some other $T_s(P)$ may not yet have reached the point at which it can perform $\mathsf{endmark}$; but all $T_s(P)$s must synchronise on all events from $\Sigma_0 \cup \{b\}$ (plus $\mathsf{endmark}$). Hence, those that reach their $\mathsf{endmark}$ before the others need to continue to offer $\Sigma_0 \cup \{b\}$ and must continue to offer these events when they perform b at this point (since, when one of them performs b here, the b has been synchronised on by the other $T_s(P)$s, some of whom might not yet have reached their $\mathsf{endmark}$s).

The effect of this parallel composition is that $\mathit{Test}_M(P)$ can perform some behaviour from U iff each $T_s(P)$ can perform a behaviour from U. Hence, we have that

$$M \nsubseteq \mathcal{FL}[\![P]\!] \Leftrightarrow \mathit{Spec} \sqsubseteq_{\mathcal{FL}} \mathit{Test}_M(P).$$

Example. In the case of the running example involving determinism, for which we chose the refutation $M = \{\langle \bullet, e, \bullet \rangle, \langle \{\} \rangle\}$, we see that $\mathit{Test}_M(P)$ is the parallel composition of two processes, $T_{\langle\{\}\rangle}(P)$ and $T_{\langle \bullet, e, \bullet \rangle}(P)$, synchronising on $\Sigma_0 \cup \{b, \mathsf{endmark}\}$. Recall that $T_{\langle\{\}\rangle}(P)$ exhibits the behaviour $\langle \Sigma_0 \cup \{b, \mathsf{endmark}\}, \mathsf{endmark}, \Sigma_0 \rangle$ precisely when P exhibits $\langle \{\} \rangle$, and that $T_{\langle \bullet, e, \bullet \rangle}(P)$ exhibits $\langle \bullet, d, \Sigma_0 \cup \{b, \mathsf{endmark}\}, \mathsf{endmark}, \Sigma_0 \rangle$ exactly when P exhibits $\langle \bullet, e, \bullet \rangle$. When P can exhibit each of these behaviours, $\mathit{Test}_M(P)$ will be able to initially perform $\langle \bullet, d \rangle$. Only at this point does $T_{\langle \bullet, e, \bullet \rangle}(P)$ offer $\Sigma_0 \cup \{b, \mathsf{endmark}\}$, while $T_{\langle\{\}\rangle}(P)$ is still in its initial state. The two can now synchronise and will proceed in lock-step, resulting in $\mathit{Test}_M(P)$ then performing $\langle \Sigma_0 \cup \{b, \mathsf{endmark}\}, \mathsf{endmark}, \Sigma_0 \rangle$. We see then that $\mathit{Test}_M(P)$ can perform the entire behaviour $\langle \bullet, d, \Sigma_0 \cup \{b, \mathsf{endmark}\}, \mathsf{endmark}, \Sigma_0 \rangle$, which is present in U, precisely when each of its sub-processes can perform a behaviour from U, as required.

Returning to the general case, as stated earlier we can assert that $M \nsubseteq \mathcal{FL}[\![P]\!]$ for all M in the set S of refutations that characterises the safety predicate Pred being expressed here by testing that

$$\mathit{Spec} \sqsubseteq_{\mathcal{FL}} \textstyle\bigsqcap_{M \in S} \mathit{Test}_M(P).$$

The context $G(\_)$ is then defined for all arguments P as

$$G(P) = \textstyle\bigsqcap_{M \in S} \mathit{Test}_M(P),$$

which gives us our result.

Theorem 15. Let $\mathit{Pred} \in \mathbf{Pred}$ be a safety predicate. Then there exists a specification process Spec and CSP context $G(\_)$ such that

$$\forall\, P \in \mathit{CSP} \,.\, \mathit{Pred}(P) \Leftrightarrow \mathit{Spec} \sqsubseteq_{\mathcal{FL}} G(P).$$

It follows that all such Pred are refinement-closed in $\mathcal{FL}$.

Corollary 16. Let $\mathit{Pred} \in \mathbf{Pred}$ be a safety predicate. Then Pred is refinement-closed in $\mathcal{FL}$.

Taken together with Theorem 9, Theorem 15, implies that the safety predicates, and the predicates that can be expressed as refinement checks of the form $\mathit{Spec} \sqsubseteq_{\mathcal{M}} G(P)$ in some finite linear observations model $\mathcal{M}$, are one and the same.

Theorem 17. Let *Pred* be a FNDF predicate from **Pred**. Then *Pred* is a safety predicate iff there exists a specification process *Spec* and CSP context $G(\_)$ and a finite linear observations model $\mathcal{M}$ such that

$$\forall\, P \in \mathit{CSP} \,.\, \mathit{Pred}(P) \Leftrightarrow \mathit{Spec} \sqsubseteq_{\mathcal{M}} G(P). \tag{19}$$

Proof. Let *Pred* be an FNDF predicate. Suppose we can find *Spec*, $G(\_)$ and $\mathcal{M}$ such that (19) is satisfied. Then by Theorem 9, *Pred* is a safety predicate. Alternatively, suppose *Pred* is a safety predicate. Then by Theorem 15, we can find *Spec* and $G(\_)$ and choose $\mathcal{M} = \mathcal{FL}$, such that (19) is satisfied. □

5. Expressing the Refinement-Closures of Liveness Predicates as Refinement Tests

We now consider to what degree the refinement-closures of liveness predicates can be expressed as refinement tests. Recall that the $\mathcal{M}$-refinement-closure of a liveness predicate Pred_L is denoted $RC_{\mathcal{M}}(\mathit{Pred}_L)$ and is simply the predicate that asserts that Pred_L must hold for all $\mathcal{M}$-refinements of a process. Recall also that when Pred_L is refinement-closed in $\mathcal{M}$, that $\mathit{Pred}_L = RC_{\mathcal{M}}(\mathit{Pred}_L)$. Hence, the set of predicates that are the refinement-closures of liveness predicates includes all liveness predicates that are refinement-closed in some finite linear observations model $\mathcal{M}$.

Theorem 17 implies that if any such predicate $RC_{\mathcal{M}}(\mathit{Pred}_L)$ (other than those, like *true* and deadlock-freedom, that are also safety predicates) is to be expressed as a refinement test whose left-hand side doesn't depend on the system being analysed, the refinement test will have to be performed in some model that is not a finite-linear observations model, *i.e.* in some $\mathcal{M}^{\Downarrow}$ or $\mathcal{M}^{\#}$ that records divergence. This explains why the refinement-closed predicates $\mathit{Pred}_{\Diamond\, e}$ and $\mathit{Pred}_{\Box\,\Diamond\, e}$, from Section 3.4, must be expressed as refinement checks in the failures-divergences model, $\mathcal{F}^{\Downarrow}$.

We first consider a class of liveness predicates that includes predicates such as $\mathit{Pred}_{\Diamond\, e}$ and $\mathit{Pred}_{\Box\,\Diamond\, e}$. We call this class of predicates the class of *liveness infinite traces predicates* and show that each such predicate can be captured by a refinement check in the model $\mathcal{FL}^{\Downarrow}$, the divergence-strict counterpart of $\mathcal{FL}$. This implies that all such predicates are refinement-closed in $\mathcal{FL}^{\Downarrow}$ and so, when applied to FNDF processes, are refinement-closed in $\mathcal{FL}$ (and so equivalent to their $\mathcal{FL}$-refinement-closures). To our knowledge, this class includes many (although not all, see *e.g.* [Ros05a, Section 5]) of the liveness predicates that have been captured as refinement checks in the literature to date, besides those like deadlock-freedom that are also safety predicates.

However, we then show that there exist important refinement-closed liveness predicates that do not fall into this category and cannot be expressed as refinement checks in any standard CSP model suitable for reasoning about the kinds of processes that FDR can support.

5.1. Expressing Liveness Infinite Traces Predicates as Refinement Checks

The tests for $\mathit{Pred}_{\Diamond\, e}$ and $\mathit{Pred}_{\Box\,\Diamond\, e}$ work by essentially building a CSP context $G(\_)$ that turns infinite traces that violate the predicate into divergences, which can then be detected by building a specification *Spec* that doesn't exhibit these divergences and testing for refinement in a divergence-recording model like $\mathcal{F}^{\Downarrow}$. This technique works only because whenever these predicates are violated by the presence of some infinite behaviour $s \in \mathit{ILO}$, where $s = \langle A_0, a_0, A_1, a_1, \ldots, A_n, a_n \ldots\rangle$, (such as those s in which all a_i are not e in the case of $\mathit{Pred}_{\Diamond\, e}$) they are also violated by the presence of each infinite behaviour $t \in \mathit{ILO}$ in which some acceptances A_i in s are replaced by $\bullet$. Such predicates are indifferent to the information contained in the acceptances A_i in infinite behaviours and, as such, the only infinite behaviours they care about are a system's *infinite traces*, of the form $\langle a_0, a_1, \ldots, a_n, \ldots\rangle$. As such, we call these predicates *infinite traces predicates.*

More formally, we define an infinite traces predicate to be one that is, like a safety predicate, violated by a process exhibiting a finite number of behaviours but, unlike a safety predicate, these behaviours can include infinite traces as well as finite linear observations. That is, any process that violates such a predicate does so because it exhibits two finite sets M and N of behaviours, where M is a finite set of finite linear observations, as in a safety predicate, and N is a finite set of infinite traces from Σ^{ω}, the set of all infinite sequences of events from Σ. We refer to infinite traces predicates that are liveness predicates as *liveness infinite traces* *predicates*. We formally define infinite traces predicates below, and show that all such predicates can be expressed as refinement checks in the model $\mathcal{FL}^{\Downarrow}$.

Recall that the infinite behaviours, $\mathcal{I}[\![P]\!]$, of an FNDF process P can be calculated by taking the closure of $\mathcal{FL}[\![P]\!]$ as given by (14). Lifting the function tr from (6), that given a linear observation $s \in \mathcal{FL}[\![P]\!]$ returns the sequence of events performed in that observation, from finite linear observations to infinite linear observations in the obvious way, we may define the set of P's infinite traces $itraces(P)$, as

$$itraces(P) = \bigcup_{s \in \mathcal{I}[\![P]\!]} \{tr(s)\}. \tag{20}$$

With this we can now formally define the infinite traces predicates.

Definition 18 (Infinite Traces Predicate). An FNDF predicate $Pred \in \mathbf{Pred}$ is an *infinite traces* predicate iff

$$\begin{array}{l} \forall\, Sys \in CSP \;.\; \neg Pred(Sys) \Rightarrow \\ \left(\begin{array}{l} \exists\, M, N \;.\; |M| \in \mathbb{N} \wedge |N| \in \mathbb{N} \wedge M \subseteq \mathcal{FL}[\![Sys]\!] \wedge N \subseteq itraces(Sys) \wedge \\ \qquad (\forall\, Sys' \in CSP \;.\; M \subseteq \mathcal{FL}[\![Sys]\!] \wedge N \subseteq itraces(Sys') \Rightarrow \neg Pred(Sys')) \end{array} \right). \end{array}$$

Observe that every safety predicate is trivially an infinite traces predicate. Observe also that $Pred_{\Diamond e}$ and $Pred_{\Box \Diamond e}$ are both liveness infinite traces predicates: each is a liveness predicate, and each is also an infinite traces predicate. We conjecture that, in practice, most liveness predicates (that are not safety predicates) that have been tested using refinement checks in divergence-recording models are liveness infinite traces predicates. However, we do note that there exist liveness predicates that have been expressed as refinement checks in the literature that are not liveness infinite traces predicates. A good example are predicates that are violated by the presence of an infinite number of finite traces of events, which Roscoe [Ros05a, Section 5] has shown how to express as refinement checks in $\mathcal{F}^{\Downarrow}$, the failures-divergences model.

We show that every infinite traces predicate for FNDF processes can be expressed as a CSP refinement check in the model $\mathcal{FL}^{\Downarrow}$, and hence that every liveness infinite traces predicate can be as well. Recall that $\mathcal{FL}^{\Downarrow}$ is the divergence-strict counterpart of $\mathcal{FL}$. As well as recording a process's finite linear observations, it also records those partial linear observations that necessarily end in a $\bullet$-acceptance after which the process can diverge, and assumes that once a process can diverge it can exhibit any behaviour at all.

As in Section 4, the refinement check that we construct is not in general finitary. Let $Pred$ be a liveness infinite traces predicate. As before, we encode $Pred$ by a (possibly infinite) set S of refutations. Each refutation is now a pair $(M, N) \in S$ of two finite sets, the first of which contains finite linear observations and the second of which contains infinite traces.

Note that the set M here is identical in nature to the refutations considered in Section 4, so we can basically reuse that existing construction to test when each of these behaviours in M can be exhibited. So we concentrate, for now, on building a testing context $Test2_N(\_)$ that diverges immediately iff the system to which it is applied can exhibit each of the infinite traces from N. We will then have that for every FNDF process $P \in CSP$

$$N \not\subseteq itraces(P) \Leftrightarrow CHAOS_\Sigma \sqsubseteq_{\mathcal{FL}^{\Downarrow}} Test2_N(P),$$

recalling that, for a set $A \subseteq \Sigma$, $CHAOS_A$ is the most general divergence-free process that performs events from A. This can then be combined with the previous construction to produce a refinement check for $Pred$.

Let P be an FNDF process and let $n \in \mathbb{N}$ be the size of the (necessarily finite) set N. Then let $sorted(N) = \langle s_0, \ldots, s_{n-1} \rangle$ be the sequence of infinite traces from N sorted under the lexicographic order. We will have $Test2_N(P)$ run n copies of P in parallel, one for each of the sequences s_i in $sorted(N)$, along with a *scheduler* process that regulates which copy of P can perform which events at which time. The scheduler will repeatedly have each copy of P perform an event in turn, always allowing the ith copy (numbered from 0) to perform only the first event in its sequence s_i. Once this event is performed, the event is removed from the front of s_i, and then the scheduler moves onto the next copy of P. $Test2_N(P)$ hides all occurrences of these events from the environment, however, so that they occur as internal activity. For each sequence s_i, let $s_i = \langle a_{i,0}, a_{i,1}, \ldots \rangle$. This means that $Test2_N(P)$ will be able to diverge iff the composition of the Ps and the scheduler can perform $\langle a_{0,0}, \ldots, a_{n-1,0}, a_{i,1}, \ldots, a_{n-1,1}, \ldots \rangle$, *i.e.* iff, for all i, the ith copy of P can perform its corresponding infinite trace s_i. Otherwise, $Test2_N(P)$ will simply deadlock.

In order to distinguish the events of each copy of P, $Test2_N(P)$ applies a renaming so that the ith copy of P performs each of its events a as $i.a$. Thus we need to extend the alphabet Σ of events as before. Recall that Σ_0 denotes the original set of events before it was extended. We extend Σ by adding to it the fresh event $i.x$ for every event $x \in \Sigma_0$, for all $i \in \{0, \ldots, n-1\}$. We then define $\mathit{Test2}_N(\_)$ for argument P as

$$\mathit{Test2}_N(P) = \left(\left(\mathop{|||}_{i \in \{0,\ldots,n-1\}} P[\![{}^{i.x}/_{x} \mid x \in \Sigma_0]\!] \right) \mathop{\|}_{\Sigma} \mathit{Sched}(0, n, \mathit{sorted}(N)) \right) \setminus \Sigma,$$

where the scheduler process Sched is defined as

$$\mathit{Sched}(i, n, \langle s_0, \ldots, s_i, \ldots, s_{n-1} \rangle) = i!\mathit{head}(s_i) \to \mathit{Sched}((i+1) \,\mathbf{mod}\, n, n, \langle s_0, \ldots, \mathit{tail}(s_i), \ldots, s_{n-1} \rangle),$$

where, for an arbitrary non-empty sequence s, $\mathit{head}(s)$ gives the first element of the sequence and $\mathit{tail}(s)$ gives the remainder of the sequence following its first element.

We now combine this with the previous construction for safety predicates from Section 4 to produce a refinement check that fails precisely when the process P can perform all finite behaviours in M and all infinite traces in N. The basic idea will be to run the safety predicate testing context $\mathit{Test}_M(P)$ until we reach the point at which all behaviours in M have been exhibited. We will then hand-over control to the context $\mathit{Test2}_N(P)$ to test for the occurrence of all behaviours in N via divergence.

The handover will be signalled by the occurrence of the event $\mathsf{endmark}$. We define a *monitor* process that runs alongside $\mathit{Test}_M(P)$ waiting for $\mathsf{endmark}$ to occur. Recall that $\mathit{Test}_M(P)$ is designed so that it stably accepts Σ_0 after performing $\mathsf{endmark}$ precisely when P has performed every behaviour in M. Hence, after synchronising with $\mathit{Test}_M(P)$ on $\mathsf{endmark}$, the monitor process then also offers the entirety of Σ_0, so that the combination accepts Σ_0 here iff $\mathit{Test}_M(P)$ does. Once some event from Σ_0 is performed, the monitor process hands control over to $\mathit{Test2}_N(P)$ to test for divergence.

The combination of the monitor process and the testing contexts already defined we denote $\mathit{Test3}_{(M,N)}(P)$ and define as

$$\mathit{Test3}_{(M,N)}(P) = \mathit{Test}_M(P) \mathop{\|}_{\Sigma_0 \cup \{\mathsf{endmark}\}} M_N(P),$$

where the monitor context $M_N(\_)$ is defined as

$$M_N(P) = ?x : \Sigma_0 \to M_N(P) \,\square\, \mathsf{endmark} \to ?x : \Sigma_0 \to \mathit{Test2}_N(P).$$

Notice that we need to have the monitor process continually offer the entirety of Σ_0 before $\mathsf{endmark}$ is performed, because it synchronises with $\mathit{Test}_M(P)$. Doing so ensures that it continually offers the entirety of $\Sigma_0 \cup \{\mathsf{endmark}\}$ until $\mathsf{endmark}$ is performed. Since this is precisely the set of events with which it synchronises with $\mathit{Test}_M(P)$, this ensures that it does not interfere with $\mathit{Test}_M(P)$.

To test for when P can exhibit every behaviour from M and N, we modify the previous specification Spec from Section 4 to allow it to stably accept Σ_0 after performing $\mathsf{endmark}$, but then to evolve to the process CHAOS_Σ, the most general divergence-free process. However, it will specifically allow divergence to occur, by evolving to $\mathbf{div}$, when Σ_0 is not stably accepted following $\mathsf{endmark}$. In this way, the refinement check will fail precisely when P exhibits every behaviour from M and then $\mathit{Test2}_N(P)$ diverges, in which case P has also exhibited every behaviour from N, and $\neg\mathit{Pred}(P)$ must be true.

We define the new specification process $\mathit{Spec2}$ as

$$\begin{aligned} \mathit{Spec2} = {} & (?x : \Sigma_0 \cup \{b\} \to (\mathit{Spec2} \lessdot x = b \gtrdot \mathit{CHAOS}_\Sigma)) \sqcap \\ & \left(\textstyle\bigsqcap_{X \in \mathcal{P}(\Sigma - \{\mathsf{endmark}\}) - (\Sigma_0 \cup \{b\})} ?x : X \to (\mathit{Spec2} \lessdot x = d \gtrdot \mathit{CHAOS}_\Sigma) \right) \sqcap \\ & \left(\mathsf{endmark} \to \left(\textstyle\bigsqcap_{X \in \mathcal{P}(\Sigma) - \Sigma_0} ?x : X \to \mathbf{div} \right) \square\, ?x : \Sigma_0 \cup \{b\} \to \mathbf{div} \,\square\, ?x : \Sigma_0 \to \mathit{CHAOS}_\Sigma \right). \end{aligned}$$

We then have that

$$\neg(M \subseteq \mathcal{FL}[\![P]\!] \wedge N \subseteq \mathit{itraces}(P)) \Leftrightarrow \mathit{Spec2} \sqsubseteq_{FL^{\Downarrow}} \mathit{Test3}_{(M,N)}(P).$$

As before, we can lift this to the entire set of refutations S, by testing whether $\mathit{Spec2} \sqsubseteq_{\mathcal{FL}^{\Downarrow}} \bigsqcap_{(M,N) \in S} \mathit{Test3}_{(M,N)}(P)$. Thus, we have that

$$\mathit{Pred}(P) \Leftrightarrow \mathit{Spec2} \sqsubseteq_{\mathcal{FL}^{\Downarrow}} \bigsqcap_{(M,N) \in S} \mathit{Test3}_{(M,N)}(P).$$

Letting $G(\_) = \bigsqcap_{(M,N)\in S} \mathit{Test3}_{(M,N)}(\_)$ and $\mathit{Spec} = \mathit{Spec2}$, we then arrive at our result.

Theorem 19. Let $\mathit{Pred} \in \mathbf{Pred}$ be an infinite traces predicate. Then there exists a specification process *Spec* and CSP context $G(\_)$ such that

$\forall\, P \in \mathit{CSP}\ .\ \mathit{Pred}(P) \Leftrightarrow \mathit{Spec} \sqsubseteq_{\mathcal{FL}^{\Downarrow}} G(P).$

This implies, of course, that every liveness infinite traces predicate can also be expressed in this way.

Corollary 20. Let $\mathit{Pred} \in \mathbf{Pred}$ be a liveness infinite traces predicate. Then there exists a specification process *Spec* and CSP context $G(\_)$ such that

$\forall\, P \in \mathit{CSP}\ .\ \mathit{Pred}(P) \Leftrightarrow \mathit{Spec} \sqsubseteq_{\mathcal{FL}^{\Downarrow}} G(P).$

It follows that all such *Pred* are refinement-closed in $\mathcal{FL}^{\Downarrow}$ and, when applied only to FNDF processes, are also refinement-closed in $\mathcal{FL}$.

Corollary 21. Let $\mathit{Pred} \in \mathbf{Pred}$ be an infinite traces FNDF predicate. Then *Pred* is refinement-closed in $\mathcal{FL}$ and $\mathcal{FL}^{\Downarrow}$.

Hence, any such predicate *Pred* is equivalent to $RC_{\mathcal{FL}}(\mathit{Pred})$, its $\mathcal{FL}$-refinement-closure.

As mentioned earlier, Roscoe [Ros05a, Section 5] has shown another class of liveness predicates that can be expressed as refinement checks in a standard divergence-recording model. These are the liveness predicates that are violated by the presence of an infinite number of finite traces, which Roscoe shows can be expressed in the failures-divergences model $\mathcal{F}^{\Downarrow}$, and so can naturally also be captured by checks in $\mathcal{FL}^{\Downarrow}$. We leave open the question as to what other kinds of liveness predicates can be expressed as refinement checks in standard divergence-recording models of CSP, like $\mathcal{FL}^{\Downarrow}$.

5.2. A Class of Predicates that Cannot be Expressed as Refinement Checks for FDR

Having showed that all liveness infinite traces predicates (which are equivalent to their refinement-closures) can be expressed as refinement checks, we now demonstrate that there exist other useful liveness predicates whose refinement-closures cannot be expressed as a refinement check at all, in any finite linear observations models $\mathcal{M}$, divergence-strict models $\mathcal{M}^{\Downarrow}$ and non-divergence-strict divergence-recording models $\mathcal{M}^{\#}$. Recall, from Section 2.4, that these encompass are all models suitable for reasoning about the kinds of processes FDR can support, and that for this reason we refer to these models as those standard CSP models that FDR might reasonably support. Theorem 22 below, which generalises an earlier result of Roscoe [Ros05a, p. 106], characterises one such class of predicates that cannot be expressed in the form of refinement checks in any such model.

Let *Pred* be an FNDF predicate (*e.g.* $\mathit{Pred} = RC_{\mathcal{M}}(\mathit{Pred}_L)$ for some liveness predicate Pred_L) from **Pred** that we wish to express as a refinement check. For most of this paper, we have concentrated on refinement checks whose left-hand side is independent of the system to which *Pred* is being applied. In order to ensure maximum generality, however, Theorem 22 considers all refinement checks that one might construct to express *Pred*, including those whose left-hand side is not independent of the system being analysed. Any such refinement check (including all of those considered so far) that expresses *Pred* can be defined in terms of two CSP contexts, $F(\_)$ and $G(\_)$, and a CSP model $\mathcal{M}$ such that

$\forall\, P \in \mathit{CSP}\ .\ \mathit{Pred}(P) \Leftrightarrow F(P) \sqsubseteq_{\mathcal{M}} G(P).$

For tests whose left-hand side is independent of the system being analysed, the context $F(\_)$ simply ignores its argument. If no two contexts $F(\_)$ and $G(\_)$ and no CSP model $\mathcal{M}$ can be found such that the above statement holds, then *Pred* cannot be expressed as a refinement check in any such model $\mathcal{M}$.

Theorem 22. Let $\mathcal{M}$ be a finite linear observations model, a divergence-strict model or a non-divergence-strict divergence-recording model and let $\mathit{Pred} \in \mathbf{Pred}$ be an arbitrary FNDF predicate. If there exists an infinite decreasing (under $\sqsubseteq_{\mathcal{M}}$) sequence $\langle B_n \mid n \in \mathbb{N}\rangle$ of FNDF, trace-equivalent processes, and a single FNDF process B^* that is the limit of this sequence (*i.e.* for all relevant i, $\mathcal{M}_i[\![B^*]\!] = \bigcup_{n\in\mathbb{N}} \mathcal{M}_i[\![B_n]\!]$), where $\forall\, n \in \mathbb{N}\ .\ B^* \equiv_{\mathcal{T}} B_n \wedge \mathit{Pred}(B_n)$ but $\neg\mathit{Pred}(B^*)$, then no refinement-test $F(P) \sqsubseteq_{\mathcal{M}} G(P)$ exists that can express $\mathit{Pred}(P)$ for arbitrary $P \in \mathit{CSP}$.

Proof. Suppose the conditions of the theorem. We use proof by contradiction. Suppose there is a refinement test of the form $F(P) \sqsubseteq_{\mathcal{M}} G(P)$ that expresses $\mathit{Pred}(P)$ for all $P \in \mathit{CSP}$. Let k be the number of different kinds of behaviour recorded by $\mathcal{M}$ and the representation of an arbitrary process P in $\mathcal{M}$ be denoted $\mathcal{M}_1[\![P]\!], \ldots, \mathcal{M}_k[\![P]\!]$. Then we have that $\forall\, 1 \leq i \leq k \;.\; \mathcal{M}_i[\![B^*]\!] = \bigcup_{n \in \mathbb{N}} \mathcal{M}_i[\![B_n]\!]$. We must have $\forall\, n \in \mathbb{N} \;.\; F(B_n) \sqsubseteq_{\mathcal{M}} G(B_n)$ but $F(B^*) \not\sqsubseteq_{\mathcal{M}} G(B^*)$. Then $G(B^*)$ must have some behaviour $b \in \mathcal{M}_i[\![G(B^*)]\!] - \mathcal{M}_i[\![F(B^*)]\!]$ for some $i \in \{1, \ldots, k\}$ that $F(B^*)$ does not. Hence, $\forall\, n \in \mathbb{N} \;.\; b \notin \mathcal{M}_i[\![F(B_n)]\!]$, and so

$$\forall\, n \in \mathbb{N} \;.\; b \notin \mathcal{M}_i[\![G(B_n)]\!]. \tag{21}$$

Observe that b cannot be a divergence since B^* is trace-equivalent to every B_n and, for any FNDF process P, the divergences of $G(P)$ depend only on P's traces and divergences. This means that the ith kind of behaviour recorded by $\mathcal{M}$ cannot be divergences and so must be some kind of finite observations. Then from (10), it follows that there exists some finite set Φ from $\mathcal{M}_i[\![B^*]\!]$ that gives rise to the behaviour $b \in \mathcal{M}_i[\![G(B^*)]\!]$. Because Φ is finite, for some sufficiently large choice of n we must have that $\Phi \subseteq \mathcal{M}_i[\![B_n]\!]$ and, by (10) therefore, $b \in \mathcal{M}_i[\![G(B_n)]\!]$. This contradicts (21) above. Hence, *Pred* is not expressible as a refinement check in $\mathcal{M}$. $\square$

When Theorem 22 holds for some *Pred* for all finite linear models $\mathcal{M}$, divergence-strict models $\mathcal{M}^{\Downarrow}$ and divergence-recording non-divergence-strict models $\mathcal{M}^{\#}$, then no refinement check can exist that can express *Pred* for any standard CSP model that FDR might reasonably support, as defined in Section 2.4. We now present some examples of useful liveness predicates whose refinement-closures fall into this category.

5.3. Liveness Properties under Strong and Weak Event Fairness

The first examples of liveness predicates whose refinement-closures cannot generally be expressed as refinement checks involve certain liveness properties under *fairness assumptions* [AFK88, LPS81, VVK05]. These liveness predicates are in-fact naturally refinement-closed, and so we need not explicitly consider their refinement-closures here (since they are equivalent to them). We show that these refinement-closed predicates cannot be expressed as refinement checks in standard CSP models that FDR might reasonably support, as defined above.

A fairness assumption implicitly restricts the infinite behaviours that a process can perform by forbidding all of those that would be judged to be *unfair*, *i.e.* all those that violate the fairness assumption. Fairness usually means that "if a choice is possible sufficiently often, then it is sufficiently often taken" [AFK88]. It is often useful to make fairness assumptions when testing liveness properties of systems, in order to rule out unfair behaviours that might unfairly violate the liveness property in question.

For instance, let P be the process $P = \mathit{Repeats}_a \mathbin{|||} e \to \mathit{STOP}$ where, recall, $\mathit{Repeats}_a = a \to \mathit{Repeats}_a$ is the process that repeatedly performs the event a. Consider the question as to whether P satisfies the liveness property $\Diamond\, e$ that asserts that, in every execution of P, the event e must eventually occur. P clearly does not satisfy this property because P can perform an infinite number of as without performing e. However, many intuitive notions of fairness would say that this infinite behaviour is unfair and hence that, under any of these notions of fairness, P should in fact satisfy $\Diamond\, e$.

Two such fairness notions that are sometimes used in the context of event-based formalisms like CSP are those of *strong event fairness* and *weak event fairness* [Lam00, PV01, SLDW08]. These concepts specialise the notions of *strong* and *weak fairness* [Lam77] respectively, applying them to the occurrence of events. Weak event fairness asserts that an event that is continually available occurs infinitely often. Strong event fairness asserts that an event that is available infinitely often occurs infinitely often; it naturally implies weak event fairness.

We will show that no refinement check for FDR exists that can express simple liveness properties like $\Diamond\, e$ under these fairness assumptions.

Like liveness properties, these fairness assumptions are often expressed in LTL. In [Low08], Lowe presents a fragment of LTL that is suitable here for expressing liveness and fairness properties. We consider a subset of this fragment that expresses formulae ϕ of the form

$$\phi ::= e \mid \mathsf{available}\; e \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \Rightarrow \phi \mid \Diamond\, \phi \mid \Box\, \phi.$$

Here, e is an event from Σ. The formula e asserts that the event e is (guaranteed to be) the first visible event the process performs. $\mathsf{available}$ e asserts that whenever the process stabilises before performing its first visible event, the event e is available. $\mathsf{available}\ e$ does not, and cannot, assert that the process must stabilise before performing its first visible event. This is because no standard CSP denotational semantic model distinguishes, for instance, the processes $d \to STOP \,\Box\, e \to STOP$ and $(d \to STOP \,\Box\, e \to STOP) \rhd (d \to STOP \,\Box\, e \to STOP)$ [Ros08]. The former must of course satisfy any sane definition of $\mathsf{available}\ e$ and, hence, so should the latter; however, the latter is not guaranteed to stabilise before performing its first visible event. The boolean connectives have their usual meanings, as do the temporal operators $\Diamond\,\phi$ and $\Box\,\phi$, which assert that the property ϕ is eventually and always true respectively.

Using this fragment of LTL, one may express strong and weak event fairness in the usual way (see *e.g.* [Lam00, PV01, SLDW08]).[11]

Definition 23 (Strong and Weak Event Fairness). *Strong event fairness*, denoted SEF, is defined as

$$SEF = \bigwedge_{e \in \Sigma} (\Box\,\Diamond\,\mathsf{available}\ e \Rightarrow \Box\,\Diamond\,e).$$

Weak event fairness, denoted WEF, is defined as

$$WEF = \bigwedge_{e \in \Sigma} (\Diamond\,\Box\,\mathsf{available}\ e \Rightarrow \Box\,\Diamond\,e).$$

A liveness property ϕ_L under a fairness assumption ϕ_F is naturally captured by the property $\phi_F \Rightarrow \phi_L$. For instance, the property that asserts that the event e eventually occurs under the assumption of strong event fairness is the property $SEF \Rightarrow \Diamond\,e$.

Lowe defines the semantics of this fragment of LTL in CSP's refusal-traces model $\mathcal{RT}$, which recall is a finite linear observations model. These semantics are defined in terms of a satisfaction relation $P \models \phi$ that defines when an FNDF process P satisfies an LTL formula ϕ. $\models$ is defined in terms of $\mathcal{RT}[\![P]\!]$, P's representation in $\mathcal{RT}$. These semantics are defined such that for all ϕ, if $P \models \phi$, then $Q \models \phi$ for every process Q such that $P \sqsubseteq_{\mathcal{RT}} Q$. Hence, each property ϕ is refinement-closed in $\mathcal{RT}$.

These semantics can be lifted straightforwardly to the model $\mathcal{FL}$ to define, for each formula ϕ, the FNDF predicate $Pred_\phi$ from **Pred** such that

$$\forall\, P \in CSP\ .\ P \models \phi \Leftrightarrow Pred_\phi(P).$$

By Lemma 3, $Pred_\phi$ will of course be refinement-closed in $\mathcal{RT}$ and $\mathcal{FL}$ since ϕ is refinement-closed in $\mathcal{RT}$ and $\mathcal{RT} \preceq \mathcal{FL}$.

We refer the reader to Lowe [Low08] for a formal presentation of the semantics of this fragment of LTL, as well as to [Mur10, Chapter 6], which contains a thorough treatment of the semantics of these fairness assumptions as well as a more traditional presentation of the Lowe's LTL semantics as judgements over individual linear executions, and instead illustrate the semantics of our fairness properties with some representative examples from which the results of this section are directly derived.

Consider the process P, where $P = a \to P \,\Box\, b \to P$. In P, both a and b are always stably available, hence under the assumption of either strong or weak event fairness, infinite executions in which only a occurs, or only b occurs, are unfair. Thus, both events should occur infinitely often in every fair execution under either fairness assumption. Hence, $Pred_{WEF \Rightarrow \Box\,\Diamond\,b}(P)$, $Pred_{SEF \Rightarrow \Box\,\Diamond\,b}(P)$ and similarly for a.

On the other hand, consider the process Q where $Q = a \to Q \sqcap b \to Q$. Observe that Q can reach a stable state, before performing its first visible event, from which b is not available. This means that Q has the infinite execution $\langle \{a\}, a, \{a\}, a, \ldots \rangle$ in which b is never stably available. This infinite execution is not unfair under either fairness assumption, then, because, while b never occurs, b is also never stably available. Hence $\neg Pred_{SEF \Rightarrow \Diamond\,b}(Q)$, $\neg Pred_{WEF \Rightarrow \Diamond\,b}(Q)$ and similarly for a. This makes sense if one recalls that these properties are all refinement-closed, since Q is refined by the process $R = a \to R$ in which b need never occur. It indicates that our notions of fairness do not forbid one branch of an internal choice being ignored forever, in line with our expectations about refinement.

Finally, consider the process T where $T = a \to T \rhd b \to T$. The operational semantics of the "$\rhd$"

[11] Many previous treatments (*e.g.* [PV01, Puh03, Puh05, SLDW08, Liu09]) of these fairness assumptions that are syntactically identical to ours have defined $\mathsf{available}\ e$ without requiring that e be *stably* available (meaning that, unlike our interpretation, it would be satisfied by the process $e \to STOP \rhd STOP$ in which e is available only from an unstable state). This causes problems that our treatment of $\mathsf{available}\ e$ avoids [Mur10] but means that these fairness properties have slightly different interpretations here than their predecessors.

operator imply that T can initially perform a; however a can be performed only from an unstable state and T can also initially perform some internal activity and transition to a stable state from where it can perform only b. Hence, a is available in T only from unstable states. It is not the case that, whenever T stabilises before performing its first visible event, a must occur. Hence, $\neg Pred_{\mathsf{available}\ a}(T)$. Thus T has the infinite behaviour $\langle \{b\}, b, \{b\}, b, \ldots \rangle$. As above, this behaviour is fair under either fairness assumption and so $\neg Pred_{WEF \Rightarrow \Box \Diamond a}(T)$ and $\neg Pred_{SEF \Rightarrow \Box \Diamond a}(T)$. However, $Pred_{\mathsf{available}\ b}(T)$ does hold. Hence, the (only) infinite behaviour $\langle \bullet, a, \bullet, a, \ldots \rangle$ in which b never occurs necessarily satisfies $\Box\, \mathsf{available}\ b$, and so is considered unfair under either fairness assumption[12]. Thus $Pred_{WEF \Rightarrow \Box \Diamond b}(T)$ and $Pred_{SEF \Rightarrow \Box \Diamond b}(T)$ both hold.

With these examples, we can construct a sequence of processes and invoke Theorem 22 to prove that no CSP refinement test exists that can express even simple liveness properties like $\Diamond e$ and $\Box \Diamond e$ under the assumptions of strong or weak event fairness, respectively.

Corollary 24. No refinement test in any standard CSP model that FDR might reasonably support (as defined in Section 2.4) can express the predicates $Pred_{SEF \Rightarrow \Diamond e}$, $Pred_{WEF \Rightarrow \Diamond e}$, $Pred_{SEF \Rightarrow \Box \Diamond e}$ and $Pred_{WEF \Rightarrow \Box \Diamond e}$.

Proof. Let

$$B^* = a \to B^* \sqcap b \to B^*,$$
$$B_0 = a \to B_0 \rhd b \to B_0,$$
$$B_n = a \to B_{n-1} \sqcap b \to B_{n-1}, \text{for } k > 0.$$

Then, in all finite-linear, divergence-strict and non-divergence-strict divergence-recording models B^* is indeed the limit of the decreasing sequence $\langle B_0, B_1, \ldots \rangle$, and B^* is trace-equivalent to each B_n. Observe that $Pred_{WEF \Rightarrow \Box \Diamond b}(B_0)$ and so $\forall n\ .\ Pred_{WEF \Rightarrow \Box \Diamond b}(B_n)$ and, hence, $\forall n\ .\ Pred_{WEF \Rightarrow \Diamond b}(B_n)$ too. However, $\neg Pred_{SEF \Rightarrow \Diamond b}(B^*)$ and, hence, $\neg Pred_{SEF \Rightarrow \Box \Diamond b}(B^*)$ too. Note finally that, for any FNDF process P and liveness property ϕ from our fragment of LTL, $Pred_{WEF \Rightarrow \phi}(P) \Rightarrow Pred_{SEF \Rightarrow \phi}(P)$. The result then follows by Theorem 22. $\Box$

5.4. Refinement-Closed Branching-Time Liveness Predicates

A second class of liveness predicates whose refinement-closures cannot be expressed as refinement checks are certain branching-time predicates. By *branching-time*[13], we mean those predicates like $Pred_{\mathbf{EF}\ e}$ from (11) that assert that some condition is satisfied on at least one execution of the system.

Recall that $Pred_{\mathbf{EF}\ e}$ is not refinement-closed in any CSP model $\mathcal{M}$. Hence, it is not equivalent to its $\mathcal{M}$-refinement-closure, $RC_{\mathcal{M}}(Pred_{\mathbf{EF}\ e})$, for any such $\mathcal{M}$. We will show that $RC_{\mathcal{M}}(Pred_{\mathbf{EF}\ e})$ cannot generally be expressed as a refinement check in standard CSP models that FDR might reasonably support, as defined in Section 2.4. For each choice of $\mathcal{M}$, the predicate $RC_{\mathcal{M}}(Pred_{\mathbf{EF}\ e})$ may be different, *i.e.* for two different CSP models $\mathcal{M}$ and $\mathcal{M}'$, $RC_{\mathcal{M}}(Pred_{\mathbf{EF}\ e})$ and $RC_{\mathcal{M}'}(Pred_{\mathbf{EF}\ e})$ might be different from each other[14]. It is therefore necessary to decide in which $\mathcal{M}$-refinement-closures of $Pred_{\mathbf{EF}\ e}$ we are interested (*i.e.* to determine what values $\mathcal{M}$ can sensibly take here).

For FNDF predicates $Pred$, we may restrict our attention to their refinement-closures in finite linear observations models. This is because, in Definition 13, the process Q is necessarily divergence-free. Hence, when $\mathcal{M}$ is a finite linear observations model, then for all FNDF predicates $Pred \in \mathbf{Pred}$, $RC_{\mathcal{M}}(Pred) = RC_{\mathcal{M}^{\Downarrow}}(Pred) = RC_{\mathcal{M}^{\#}}(Pred)$. Therefore, when considering $RC_{\mathcal{M}}(Pred)$ for some any FNDF predicate $Pred$, we can restrict our attention to those $\mathcal{M}$ that are finite linear observations models.

Furthermore, it turns out that, for any predicates $Pred$, if $\neg Pred(STOP)$ holds, then $RC_{\mathcal{T}}(Pred) = \{\}$, *i.e.* its refinement-closure in the traces model $\mathcal{T}$ is very often useless. This is because, when $\neg Pred(STOP)$ holds, $RC_{\mathcal{T}}(Pred) = \mathit{false}$, the predicate that is satisfied by no processes. To see why, observe that for any process P, $P \sqsubseteq_{\mathcal{T}} STOP$. When $\neg Pred(STOP)$, then for all processes P, $\neg RC_{\mathcal{T}}(Pred)(P)$ must hold. Hence,

[12] This can be confirmed by examining Lowe's semantics [Low08]. Further discussion about this matter can be found in [Mur10, Section 6.1], to which we refer the interested reader.
[13] This name is inspired by similar usage in *e.g.* [CS10].
[14] Lemma 29 in Appendix A shows that when $\mathcal{M} \preceq \mathcal{M}'$, it is the case that for any $Pred \in \mathbf{Pred}$, $RC_{\mathcal{M}}(Pred) \subseteq RC_{\mathcal{M}'}(Pred)$.

for predicates *Pred* that are not satisfied by $STOP$, when considering $RC_{\mathcal{M}}(Pred)$, we may further restrict our attention to those $\mathcal{M}$ that are finite linear observations models other than $\mathcal{T}$.

Note that $\neg Pred_{\mathbf{EF}\,e}(STOP)$, since $STOP$ never performs the event e. The following corollary proves that for all finite linear observations models $\mathcal{M}$ other than $\mathcal{T}$, $Pred_{\mathbf{EF}\,e}$'s $\mathcal{M}$-refinement-closure, $RC_{\mathcal{M}}(Pred_{\mathbf{EF}\,e})$, cannot be expressed as a refinement check in any standard CSP model that FDR might reasonably support, as defined in Section 2.4.

Corollary 25. Let $\mathcal{M}$ be a finite linear observations model other than $\mathcal{T}$. Then no refinement test in any standard CSP model that FDR might reasonably support (as defined in Section 2.4) can express the predicate $RC_{\mathcal{M}}(Pred_{\mathbf{EF}\,e})$.

Proof. Let B^* and $\langle B_n \mid n \in \mathbb{N}\rangle$ be as in Corollary 24 and consider the predicate $Pred_{\mathbf{EF}\,b}$. Then, since B^* is refined by the process $Repeats_a$ in every finite linear observations model $\mathcal{M}$, for all such models $\mathcal{M}$

$$\neg RC_{\mathcal{M}}(Pred_{\mathbf{EF}\,b})(B^*).$$

Also, for all finite linear observations models $\mathcal{M}$ other than $\mathcal{T}$, we have $\forall\, Q\ .\ B_0 \sqsubseteq_{\mathcal{M}} Q \Rightarrow Pred_{\mathbf{EF}\,b}(Q)$. Hence, for all such models $\mathcal{M}$, $RC_{\mathcal{M}}(Pred_{\mathbf{EF}\,b})(B_0)$. It follows that, for all such models $\mathcal{M}$,

$$\forall\, n \in \mathbb{N}\ .\ RC_{\mathcal{M}}(Pred_{\mathbf{EF}\,b})(B_n).$$

The result then follows by Theorem 22. □

5.5. Refinement-Closed Non-Causation

In earlier work [Mur10], we considered the problem of testing *non-causation* predicates for FNDF CSP processes, in order to reason about *authority* [Mil06] in systems of interacting agents. Imagine a process *System* that represents a system comprising a number of agents. Each agent has an *alphabet* of events $A \subseteq \Sigma$ in whose occurrence in *System* it is involved. Then the *authority* of each agent includes those effects that the agent can cause to occur in *System*, by performing events from its alphabet A

As a simple example, consider the effect that is "the event e can occur" and the question as to whether the agent whose alphabet is A causes this effect in *System*. Suppose e can occur in *System* (or else this question would be meaningless), *i.e.* that $Pred_{\mathbf{EF}\,e}(System)$ holds. Then one way to decide this question is to examine the system in the *counterfactual* [Lew73] case in which the agent with alphabet A is prevented from acting, and see whether e can occur here. This counterfactual case is captured by the process $System \underset{A}{\|} STOP$, in which we force all events in A to synchronise with the process $STOP$ to prevent any of them from occurring, which we abbreviate $System\,|_A$. If e cannot occur in $System\,|_A$, *i.e.* if $\neg Pred_{\mathbf{EF}\,e}(System\,|_A)$, then we say that the agent with alphabet A causes e in *System*, in accordance with Lewis' notion of causation by *counterfactual dependence* [Lew73].

Hence, we have that the effect "the event e can occur" is caused in *System* by the agent whose alphabet is A iff

$$Pred_{\mathbf{EF}\,e}(System) \wedge \neg Pred_{\mathbf{EF}\,e}(System\,|_A).$$

This allows us to define when this effect is not caused by the agent with alphabet A, *i.e.* the *non-causation* of this effect by this agent, as

$$Pred_{\mathbf{EF}\,e}(System) \Rightarrow Pred_{\mathbf{EF}\,e}(System\,|_A).$$

We may of course generalise this approach to arbitrary effects by identifying each effect with the predicate that holds for all systems in which the effect is present. Hence, each predicate $Pred \in \mathbf{Pred}$ captures an effect, and we can talk about effects and predicates interchangeably. This leads to the following general definition for non-causation.

Definition 26 (Non-Causation). In a system *System*, the agent whose alphabet is A, does not cause the effect captured by the predicate $Pred \in \mathbf{Pred}$ iff

$$Pred(System) \Rightarrow Pred(System\,|_A).$$

This is captured by the FNDF predicate denoted $NC(A, Pred)$ from **Pred** defined as

$$NC(A, Pred) = \{\mathcal{FL}[\![P]\!] \mid P \in CSP \wedge (Pred(P) \Rightarrow Pred(P\,|_A))\}.$$

We have shown [Mur10] that this general definition for non-causation is able to capture a range of kinds of authority, including delegable, non-delegable, revocable and single-use authority, as well as the notions of *defensive correctness* and *defensive consistency* [Mil06].

$NC(A, Pred)$ is not, in general, refinement-closed. This can be seen by considering the system $System = a \to b \to STOP \sqcap b \to STOP$ that comprises two agents with alphabets $A = \{a\}$ and $B = \{b\}$ respectively and asking the question as to whether the agent with alphabet A causes b to be able to occur (*i.e.* the effect captured by the predicate $Pred_{\mathbf{EF}\, b}$) in $System$. Since $System\,|_A = STOP \sqcap b \to STOP$, $Pred_{\mathbf{EF}\, b}(System\,|_A)$ holds and so non-causation clearly holds here. However, letting $System' = a \to b \to STOP$, we see that $System \sqsubseteq_{\mathcal{M}} System'$ in every CSP model $\mathcal{M}$. Non-causation doesn't hold for $System'$, since $System'\,|_A = STOP$ and $\neg Pred_{\mathbf{EF}\, b}(STOP)$. Hence, $NC(A, Pred_{\mathbf{EF}\, b})$ cannot be refinement-closed and so is not generally equivalent to its refinement-closure.

Note that for certain effects like $Pred_{\mathbf{EF}\, e}$, the traces refinement-closure of non-causation, *e.g.* $RC_{\mathcal{T}}(NC(A, Pred_{\mathbf{EF}\, e}))$, is not usually very sensible. Consider, for instance, the process $System = a \to b \to STOP \rhd b \to STOP$ and let $A = \{a\}$. For all finite linear observations models $\mathcal{M}$ other than $\mathcal{T}$, it is the case that for all $\mathcal{M}$-refinements $System'$ of $System$, $Pred_{\mathbf{EF}\, b}(System'\,|_A)$ holds and so $NC(A, Pred_{\mathbf{EF}\, b})(System')$ holds. Hence, for all such $\mathcal{M}$, $RC_{\mathcal{M}}(NC(A, Pred_{\mathbf{EF}\, b}))(System)$ holds. However, $System \sqsubseteq_{\mathcal{T}} a \to b \to STOP$ and clearly $\neg Pred_{\mathbf{EF}\, b}((a \to b \to STOP)\,|_A)$, since $(a \to b \to STOP)\,|_A = STOP$. Hence, $\neg NC(A, Pred_{\mathbf{EF}\, b})(a \to b \to STOP)$ and so $\neg RC_{\mathcal{T}}(NC(A, Pred_{\mathbf{EF}\, b}))(System)$. We conclude, therefore, that the traces refinement-closure of non-causation is not generally very useful.

For this reason, following the discussion in Section 5.4, when considering the refinement-closures $RC_{\mathcal{M}}(NC(A, Pred))$ of non-causation predicates $NC(A, Pred)$, we restrict our attention to those $\mathcal{M}$ that are finite linear observations models other than $\mathcal{T}$.

We show that even when $Pred$ is a simple effect like $Pred_{\mathbf{EF}\, e}$, $RC_{\mathcal{M}}(NC(A, Pred))$ cannot be expressed as a refinement check in any standard CSP model that FDR might reasonably support, as defined in Section 2.4, when $\mathcal{M}$ is a finite linear observations model other than $\mathcal{T}$.

Corollary 27. Let $\mathcal{M}$ be a finite linear observations model other than $\mathcal{T}$. Then no refinement test in any standard CSP model that FDR might reasonably support (as defined in Section 2.4) can express the predicate $RC_{\mathcal{M}}(NC(A, Pred_{\mathbf{EF}\, e}))$.

Proof. Let $P^* = c \to b \to STOP \;|||\; B^*$ and $P_n = c \to b \to STOP \;|||\; B_n$ for all $n \in \mathbb{N}$, where B^* and $\langle B_n \mid n \in \mathbb{N}\rangle$ are defined as in Corollary 24, and consider the predicate $RC_{\mathcal{M}}(NC(\{c\}, Pred_{\mathbf{EF}\, b}))$. Then, since B^* is refined by the process $Repeats_a$ in every finite linear observations model $\mathcal{M}$, P^* is refined by $c \to b \to STOP \;|||\; Repeats_a$ in every such model $\mathcal{M}$. The latter clearly doesn't satisfy $NC(\{c\}, Pred_{\mathbf{EF}\, b})$. So, for all finite linear observations models $\mathcal{M}$, we have that

$$\neg RC_{\mathcal{M}}(NC(\{c\}, Pred_{\mathbf{EF}\, b}))(P^*).$$

Also, recall from Corollary 25 that, for all finite linear observations models $\mathcal{M}$ other than $\mathcal{T}$, we have $\forall n \in \mathbb{N} \,.\, RC_{\mathcal{M}}(Pred_{\mathbf{EF}\, b}(B_n))$. Observe that for all $n \in \mathbb{N}$, $P_n\,|_{\{c\}} = B_n$. Hence, for all finite linear observations models $\mathcal{M}$ other than $\mathcal{T}$, we must have that $\forall n \in \mathbb{N} \,.\, RC_{\mathcal{M}}(Pred_{\mathbf{EF}\, b})(P_n\,|_{\{c\}})$. Let n be a natural number, $\mathcal{M}$ be some finite linear observations model other than $\mathcal{T}$ and Q be some FNDF process such that $P_n \sqsubseteq_{\mathcal{M}} Q$. Then $P_n\,|_{\{c\}} \sqsubseteq_{\mathcal{M}} Q\,|_{\{c\}}$ and so $Pred_{\mathbf{EF}\, b}(Q\,|_{\{c\}})$ must hold, and so $NC(\{c\}, Pred_{\mathbf{EF}\, b})(Q)$ holds. Hence,

$$\forall n \in \mathbb{N} \,.\, RC_{\mathcal{M}}(NC(\{c\}, Pred_{\mathbf{EF}\, b}))(P_n).$$

The result then follows by Theorem 22. $\square$

5.6. Refinement-Testing the Negation of a Non-Refinement-Testable Predicate

We have shown, by repeatedly applying Theorem 22, that a number of predicates $Pred$ (such as $Pred = RC_{\mathcal{M}}(Pred_{\mathbf{EF}\, e})$) cannot be expressed as refinement tests in any standard CSP model that FDR might reasonably support, as defined in Section 2.4. For lack of a better term, we call any such predicate a *non-refinement-testable* predicate. One way that one might work around the inability to express one of these predicates $Pred$ as a refinement test, might be to try to express $Pred$'s complement, $\overline{Pred}$, instead. One might then apply the resulting refinement test to decide $Pred$ by simply negating whatever result the refinement test yields.

However, we now show that this strategy is unlikely to work very well, at least for the *Pred*s considered in this section to which Theorem 22 can be applied, since the complement $\overline{\mathit{Pred}}$ of any such predicate *Pred* cannot be refinement-closed in any standard CSP model $\mathcal{M}$ that FDR might reasonably support. Hence, $\overline{\mathit{Pred}}$ cannot be expressed as a refinement test whose left-hand side is independent of the system to which $\overline{\mathit{Pred}}$ is being applied. This holds regardless of whether *Pred* is refinement-closed, thus further illustrating the limits of refinement checking for deciding these kinds of predicates.

Theorem 28. Let $\mathcal{M}$, *Pred*, B^* and $\langle B_n \mid n \in \mathbb{N}\rangle$ be as described in Theorem 22. Then $\overline{\mathit{Pred}}$ cannot be refinement-closed in $\mathcal{M}$.

Proof. Suppose the conditions of the theorem. Then $\neg \mathit{Pred}(B^*)$ but $\mathit{Pred}(B_n)$ for all $n \in \mathbb{N}$. Let n be any member of $\mathbb{N}$. Then

$$\overline{\mathit{Pred}}(B^*) \wedge \neg\overline{\mathit{Pred}}(B_n) \wedge B^* \sqsubseteq_{\mathcal{M}} B_n.$$

Hence, $\overline{\mathit{Pred}}$ cannot be refinement-closed in $\mathcal{M}$. □

6. Future and Related Work

From the previous section, we conclude that there exist useful refinement-closed predicates that are beyond the reach of refinement-testing to feasibly decide. We now consider future and related work, and sketch out some alternative approaches that might be used to mechanically decide these sorts of refinement-closed predicates.

6.1. Model-Checking Approaches Analysing Strongly Connected Subgraphs

We saw that refinement-closed liveness properties like $\mathit{SEF} \Rightarrow \Diamond\, e$ cannot be expressed as CSP refinement checks for FDR. We conjecture that existing work (such as [Ros01, SLDW08, Liu09]) on testing liveness properties by examining a system's operational semantics [Ros97, Chapter 7] could probably be adapted to allow one to automatically verify these kinds of liveness properties. We sketch one such possibility, based on the work in [Liu09].

The standard explicit-state, automata-based approach [VW86] to testing liveness properties expressed as LTL formulae ϕ against a system's operational semantics $\mathbf{A}$ involves first constructing [WVS83] a Büchi automaton [Büc62] $\mathbf{B}$ that corresponds to the LTL formula $\neg\phi$, *i.e.* a Büchi automaton that accepts those and only those infinite behaviours that violate the liveness property ϕ. One then constructs the *product* $\mathbf{A} \times \mathbf{B}$ of $\mathbf{A}$ and $\mathbf{B}$, which is a Büchi automaton that accepts all infinite behaviours that can be exhibited by $\mathbf{A}$ that are accepted by $\mathbf{B}$. Then the liveness property ϕ is satisfied by the system iff the language recognised by $\mathbf{A} \times \mathbf{B}$ is empty. This occurs precisely when $\mathbf{A} \times \mathbf{B}$ contains no reachable non-trivial strongly connected subgraph (SCS) that is *accepting*, *i.e.* an SCS that contains a node $(s_{\mathbf{A}}, s_{\mathbf{B}})$, where naturally $s_{\mathbf{A}}$ and $s_{\mathbf{B}}$ are states of $\mathbf{A}$ and $\mathbf{B}$ respectively, for which $s_{\mathbf{B}}$ is an accepting state of $\mathbf{B}$.

Recall that testing a liveness property ϕ_L (*e.g.* $\Diamond\, e$) under a fairness assumption ϕ_F (*e.g.* *SEF*) is equivalent to testing the property $\phi_F \Rightarrow \phi_L$ (*e.g.* $\mathit{SEF} \Rightarrow \Diamond\, e$). Hence, one way to test liveness under fairness using this standard approach involves building a Büchi automaton $\mathbf{B}$ that corresponds to the formula $\neg(\phi_F \Rightarrow \phi_L)$ and then computing $\mathbf{A} \times \mathbf{B}$ as usual. However, the construction of $\mathbf{B}$ usually scales poorly with the size of the LTL formula to which it corresponds (in the worst case, scaling exponentially). Recall that the fairness assumptions from Section 5, namely *SEF* and *WEF* from Definition 23, involve a conjunction over every event in Σ. Previous work by others [SLDW08, Liu09] examining the application of this approach using the SPIN model checker [Hol03] with similar fairness assumptions, indicate that it is unlikely to work well when Σ contains more than a few events, and is therefore infeasible here.

This problem is avoided in the PAT [Liu09] model-checking tool for CSP, by constructing $\mathbf{B}$ to correspond to just the formula $\neg\phi_L$. The accepting SCSs of $\mathbf{A} \times \mathbf{B}$ then correspond to all infinite behaviours of $\mathbf{A}$ that violate ϕ_L, whether fair or unfair. Unfair infinite behaviours that violate ϕ_F are then *pruned* away by algorithmically identifying all *fair* SCSs of the product that (correspond to infinite behaviours that) satisfy the fairness assumption ϕ_F. All infinite behaviours of the system then satisfy $\phi_F \Rightarrow \phi_L$ iff none of these fair SCSs are accepting.

Adapting this approach therefore requires one to be able to identify whether the infinite behaviours

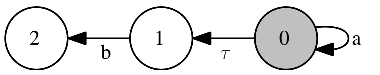

Fig. 3. The operational semantics $\mathbf{A}$ of $P = a \to P \rhd b \to STOP$.

captured by an SCS of some product $\mathbf{A} \times \mathbf{B}$ satisfy our fairness assumptions *SEF* and *WEF*. We briefly sketch how to do so.

Let S be a non-trivial SCS in the product $\mathbf{A} \times \mathbf{B}$ of a system's operational semantics $\mathbf{A}$ and a Büchi automaton $\mathbf{B}$. Let N_S and E_S be the sets of states and edges respectively of S. When $s_{\mathbf{A}}$ and $s'_{\mathbf{A}}$ are states of $\mathbf{A}$, we write $s_{\mathbf{A}} \xrightarrow{x} s'_{\mathbf{A}}$ to mean that from state $s_{\mathbf{A}}$ the system can transition to state $s'_{\mathbf{A}}$ by performing the event x from $\Sigma \cup \{\tau\}$, where τ is the special event used to represent internal activity [Ros97, Chapter 7]. We also write $s_{\mathbf{A}} \xrightarrow{x}$ to mean that there exists a state $s'_{\mathbf{A}}$ of $\mathbf{A}$ such that $s_{\mathbf{A}} \xrightarrow{x} s'_{\mathbf{A}}$. Then for each node $(s_{\mathbf{A}}, s_{\mathbf{B}}) \in N_S$, let

$$stableStates((s_{\mathbf{A}}, s_{\mathbf{B}})) = \{s'_{\mathbf{A}} \mid s_{\mathbf{A}}(\xrightarrow{\tau})^* s'_{\mathbf{A}} \wedge s'_{\mathbf{A}} \not\xrightarrow{\tau}\}$$

denote the set of states $s'_{\mathbf{A}}$ reachable from $s_{\mathbf{A}}$ in the system's operational semantics under zero or more τ-transitions such that each $s'_{\mathbf{A}}$ is stable.

Then for each $(s_{\mathbf{A}}, s_{\mathbf{B}}) \in N_S$, let

$$availableEvents((s_{\mathbf{A}}, s_{\mathbf{B}})) = \{e \mid e \in \Sigma \wedge \forall s'_{\mathbf{A}} \in stableStates((s_{\mathbf{A}}, s_{\mathbf{B}})) \,.\, s'_{\mathbf{A}} \xrightarrow{e}\}$$

denote the set that contains those events e that are available from every τ-reachable stable state $s'_{\mathbf{A}}$ in the system's operational semantics from $s_{\mathbf{A}}$. Then, let

$$\begin{aligned} sometimesAvailableEvents(S) &= \textstyle\bigcup_{(s_{\mathbf{A}}, s_{\mathbf{B}}) \in N_S} availableEvents((s_{\mathbf{A}}, s_{\mathbf{B}})), \\ alwaysAvailableEvents(S) &= \textstyle\bigcap_{(s_{\mathbf{A}}, s_{\mathbf{B}}) \in N_S} availableEvents((s_{\mathbf{A}}, s_{\mathbf{B}})), \end{aligned}$$

be the sets that contain those events that are sometimes and always respectively stably available at some point during S. Let $performedEvents(S)$ be the set of events performed in S, *i.e.*

$$performedEvents(S) = \{e \mid e \in \Sigma \wedge \exists (s_{\mathbf{A}}, s_{\mathbf{B}}), (s'_{\mathbf{A}}, s'_{\mathbf{B}}) \in N_S \,.\, ((s_{\mathbf{A}}, s_{\mathbf{B}}), e, (s'_{\mathbf{A}}, s'_{\mathbf{B}})) \in E_S\}.$$

Then we conjecture that S satisfies *SEF* iff

$$sometimesAvailableEvents(S) \subseteq performedEvents(S).$$

Similarly, we conjecture that S satisfies *WEF* iff

$$alwaysAvailableEvents(S) \subseteq performedEvents(S).$$

For example, consider the process $P = a \to P \rhd b \to STOP$. Under our LTL semantics from Section 5, $P \models WEF \Rightarrow \Diamond\, b$ and similarly for *SEF*. P's operational semantics $\mathbf{A}$ is depicted in Figure 3. The product $\mathbf{A} \times \mathbf{B}$ of $\mathbf{A}$ and the Büchi automaton $\mathbf{B}$ that corresponds to the formula $\neg \Diamond\, b$, has just a single state that has a single transition, namely a self-loop labelled with a. $\mathbf{A} \times \mathbf{B}$ has just one non-trivial SCS, which we denote S; S is, in fact, the entire automaton. S is accepting.

By the above definitions, we have that $sometimesAvailableEvents(S) = alwaysAvailableEvents(S) = \{b\}$, but that $performedEvents(S) = \{a\}$. We see that $\{b\} \not\subseteq \{a\}$. This indicates that, under the conjectures above, none of the behaviours captured by this SCS that violate the liveness property $\Diamond\, e$, satisfy either of these fairness assumptions. Hence, under these conjectures, none of the behaviours present in $\mathbf{A}$ that violate $\Diamond\, e$ satisfy *SEF* or *WEF*. This is, of course, consistent with P satisfying $WEF \Rightarrow \Diamond\, b$ and $SEF \Rightarrow \Diamond\, b$.

It can be observed that all of the predicates that we have proved in this paper cannot be tested by automatic refinement checking, involve detecting the presence of certain infinite behaviours that are not infinite traces. This can be seen by examining Theorem 22 and noting that what sets B^* there apart from each B_k must be some infinite behaviour that is not an infinite trace (since B^* is trace-equivalent to each B_k).

As implied by the discussion above, model-checking algorithms based on identifying certain SCSs within the operational semantics of a system are appropriate for testing predicates that involve detecting certain infinite behaviours (see *e.g.* [Ros01]). Hence, we conjecture that it might be possible to adapt pre-existing

SCS-based techniques more generally to detect infinite behaviours that cannot be observed using refinement-checking, *i.e.* those infinite behaviours that are not infinite traces and so cannot be detected by mapping them onto corresponding divergences. This might allow one to model-check predicates that cannot otherwise be tested using refinement-checking.

6.2. Mechanised Logical Proof

Besides automated model-checking approaches, one potential avenue would be to analyse these predicates by using mechanised logical proof directly over the $\mathcal{FL}$ model, with the aid of mechanical theorem proving technologies. The CSP-Prover [IR05, IR08] tool could be particularly useful here.

CSP-Prover is a collection of theory libraries and proof tactics for the Isabelle [Pau94] proof assistant. CSP-Prover encodes many of CSP's standard denotational semantic models as Isabelle theories, allowing one to write CSP processes and prove semantic refinement between them using Isabelle's interactive theorem proving interfaces. CSP-Prover has also been used to reason about CSP's denotational semantic models themselves [IR06, SRI09].

We conjecture that one could extend CSP-Prover to allow one to state and prove FNDF predicates of CSP processes. Doing so would first require the $\mathcal{FL}$ model to be formalised in CSP-Prover. Given [IR06, SRI09] that a number of other models, including the stable-failures and *stable-revivals* [Ros09] models, have been formalised in CSP-Prover, we expect that formalising the $\mathcal{FL}$ model should be a relatively straightforward task of adapting these existing encodings. Having formalised the $\mathcal{FL}$ model, one would then likely identify a number of key lemmas and results to be proved, regarding each kind of predicate one was interested in proving, that would assist generally in proving these predicates.

We conclude, therefore, that further work on these alternative approaches, besides refinement checking, would be very helpful in trying to allow one to verify a wider range of predicates of CSP processes than can currently be checked using FDR.

6.3. Other Related Work

This work is closely related to that of Roscoe [Ros05a], who considered what sorts of predicates could be expressed using refinement checks in the failures-divergences model $\mathcal{F}^{\Downarrow}$. Our work extends Roscoe's because we consider refinement checks in any standard CSP model that FDR might reasonably support, as defined in Section 2.4 (including, of course, the failures-divergences model). Unlike Roscoe, however, we've focused mainly on predicates that are refinement-closed in some CSP model $\mathcal{M}$, since predicates that are otherwise cannot usually be expressed as efficient refinement checks as explained earlier. Our work also differs from Roscoe's because we have been able to characterise the total set of denotational predicates of FNDF CSP processes that one might want to test in terms of safety and liveness (via Clarkson and Schneider's hyperproperties [CS10] framework). This allows one to gain an intuitive understanding of those predicates for which refinement checking is most likely to be most useful (for instance, the safety ones as shown earlier in Section 4).

In [Low09], Lowe considers the problem of how to express *n-ary failures* predicates as finitary refinement checks for FDR to carry out. These are predicates $Pred_f$ of the form

$$Pred_f(P) = \forall\, f_1, \ldots, f_n \in failures(P)\ .\ R(f_1, \ldots, f_n),$$

where R is an n-ary relation on stable-failures. Such a predicate $Pred_f$ is of course equivalent to the FNDF predicate $Pred$ where

$$\forall\, P \in CSP\ .\ Pred(P) \Leftrightarrow \forall\, b_1, \ldots, b_n \in \mathcal{FL}[\![P]\!]\ .\ \forall\, f_1, \ldots, f_n\ .\ (\forall\, 1 \leq i \leq n\ .\ f_i \in f(b_i)) \Rightarrow R(f_1, \ldots, f_n),$$

where f is as defined in (8). Hence, every n-ary failures predicate is violated by the presence of a finite set M (where $|M| = n$) of $\mathcal{FL}$ behaviours. This means that every n-ary failures predicate is a safety FNDF predicate. In fact, every n-ary failures predicate is equivalent to an *n-hypersafety predicate* [CS10], namely a safety predicate for which the size of the set M from Definition 8 is always bounded by n.

There do exist safety predicates, however, that are not n-hypersafety predicates. An example (due to Gavin Lowe) is *Pred* defined as follows, where $\#(t)$ denotes the length of the trace t,

$$\begin{array}{l}\forall P \in \mathit{CSP} \,.\, \mathit{Pred}(P) \Leftrightarrow \\ \quad \forall n \,.\, \forall t_1 \ldots t_n \in \mathit{traces}(P) \,.\, (\forall 1 \leq i \leq n \,.\, \#(t_i) \geq i) \Rightarrow \Sigma_{1 \leq i \leq n} \#(t_i \upharpoonright \{a\}) \geq n^2/6.\end{array}$$

Lowe [Low09] focused on producing *finitary* refinement checks for n-ary failures properties, namely those for which the left- and right-hand sides are always finite state whenever the system being analysed is finite-state. In contrast, our focus has been on general refinement checks, whether finitary or otherwise. (Indeed, the refinement check constructed in the proofs of Theorems 15 and 19 are certainly not finitary.) This is because we are interested in the limits of expressiveness. An obvious avenue for future work would consider how to adapt Lowe's results and techniques from [Low09] to produce finitary refinement checks for safety predicates, and to characterise those safety predicates and liveness predicates that can, and cannot, be expressed in the form of finitary refinement checks.[15]

As mentioned earlier, Lowe [Low08] has also shown how to construct finitary refinement checks to test properties expressed in the *bounded, positive* fragment of LTL. Lowe shows that such predicates can be expressed as refusal-traces refinement checks of the form $\mathit{Spec} \sqsubseteq_{\mathcal{RT}} P$ where P is the system to which the predicate is being applied. As discussed before, such predicates are, therefore, necessarily safety FNDF predicates.

Leuschel *et al.* have also considered the problem of expressing certain LTL properties as refinement checks [LMC01]. Their results are more general than Lowe's [Low08]. However, unlike Lowe's, their refinement checks involve placing the system being analysed on the left-hand side, and so are less practical to carry out (since the complexity of refinement checking can be exponential in the size of the left-hand side, as explained earlier). In this paper, we have focused on refinement checks whose left-hand side is independent of the system being analysed, in order to avoid this problem.

7. Conclusion

We have shown that the entire set of denotational predicates for finitely-nondeterministic, divergence-free CSP processes can be understood in terms of those that involve safety and those that involve liveness. We saw that refinement-closed predicates may be written as the conjunction of a safety predicate and the refinement-closure of a liveness predicate. We proved that refinement checking can express all of the safety predicates, and certain liveness predicates violated by infinite traces, but that there exist important liveness predicates whose refinement-closures cannot be expressed as refinement checks in standard CSP models that tools like FDR might support. Refinement-checking therefore has real limitations for model-checking arbitrary refinement-closed predicates of CSP processes. We conclude that further development on verification techniques for CSP, besides refinement checking, should be pursued in order to allow a wider range of predicates of CSP processes to be verified.

Acknowledgements

Thanks to Bill Roscoe and Gavin Lowe for useful discussions about the hierarchy of CSP models. Bill Roscoe also provided useful information on how to use CSP contexts to test for stable acceptances. This paper builds and extends upon work in my D.Phil. thesis [Mur10], which was supervised by, and benefited from the feedback of, Gavin Lowe, to whom I'm grateful. I would also like to thank Gavin Lowe and the anonymous reviewers, who provided very valuable feedback on earlier drafts of this paper.

References

[ACH+10] Parosh Abdulla, Yu-Fang Chen, Lukáš Holìk, Richard Mayr, and Tomáš Vojnar. When simulation meets antichains. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '10)*, volume 6015 of *Lecture Notes in Computer Science*, pages 158–174. Springer, 2010.

[15] It seems natural to suspect that perhaps only the n-hypersafety predicates [CS10] can be expressed as finitary refinement checks in finite-linear observations models, a conjecture that Gavin Lowe made to the author in private communication.

[AFK88] Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. Appraising fairness in languages for distributed programming. *Distributed Computing*, 2(4):226–241, 1988.

[AL91] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

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

[BR85] Stephen D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In *Proceedings of the 1984 Carnegie-Mellon University Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*. Springer, 1985.

[Büc62] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the 1st International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[CS08] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF '08)*, pages 51–65, 2008.

[CS10] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 2010. To appear. Preprint available at: `http://www.cs.cornell.edu/fbs/publications/Hyperproperties.JCS.pdf`.

[GGH$^+$05] Paul Gardiner, Michael Goldsmith, Jason Hulance, David Jackson, Bill Roscoe, Bryan Scattergood, and Philip Armstrong. *Failures-Divergences Refinement: FDR2 User Manual.* Formal Systems (Europe) Ltd, 2005.

[GM82] Joseph A. Goguen and José Meseguer. Security policies and security models. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy (SP '82)*, pages 11–20, 1982.

[Hoa80] C. A. R. Hoare. A model for communicating sequential processes. In R. M. McKeag and A. M. Macnaughten, editors, *On the Construction of Programs*, pages 229–254. Cambridge University Press, 1980.

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

[Hol03] Gerard J. Holzmann. *The SPIN model checker: Primer and reference manual.* Addison-Wesley, 2003.

[IR05] Y. Isobe and M. Roggenbach. A generic theorem prover of CSP refinement. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005)*, page 108. Springer Verlag, 2005.

[IR06] Yoshinao Isobe and Markus Roggenbach. A complete axiomatic semantics for the CSP stable-failures model. In *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR '06)*, volume 4137 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2006.

[IR08] Y. Isobe and M. Roggenbach. CSP-Prover: A proof tool for the verification of scalable concurrent systems. *Journal of Computer Software, Japan Society for Software Science and Technology (JSSST)*, 25(4):85–92, 2008.

[Lam77] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, March 1977.

[Lam00] Leslie Lamport. Fairness and hyperfairness. *Distributed Computing*, 13(4):239–245, 2000.

[Lat03] Timo Latvala. Efficient model checking of safety properties. In *Proceedings of the 10th International Conference on Model checking software (SPIN'03)*, pages 74–88. Springer-Verlag, 2003.

[Laz99] Ranko S. Lazić. *A Semantic Study of Data Independence with Applications to Model Checking.* D.Phil. thesis, Oxford University Computing Laboratory, 1999.

[Lew73] David Lewis. Causation. *Journal of Philosophy*, 70(17):556–567, 1973.

[LF08] Michael Leuschel and Marc Fontaine. Probing the depths of CSP-M: A new FDR-compliant validation tool. In *Formal Methods and Software Engineering, Proceedings of the 10th International Conference on Formal Engineering Methods (ICFEM '08)*, pages 278–297. Springer, 2008.

[Liu09] Yang Liu. *Model Checking Concurrent and Real-Time Systems: The PAT Approach.* PhD thesis, National University of Singapore, 2009. Draft available at: `http://www.comp.nus.edu.sg/~liuyang/thesis/thesis.pdf`.

[LMC01] Michael Leuschel, Thierry Massart, and Andrew Currie. How to make FDR spin: LTL model checking of CSP by refinement. In *Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity (FME '01)*, pages 99–118. Springer-Verlag, 2001.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[Low07] Gavin Lowe. On information flow and refinement-closure. In *Proceedings of the 7th International Workshop on Issues in the Theory of Security (WITS '07)*, 2007.

[Low08] Gavin Lowe. Specification of communicating processes: temporal logic versus refusals-based refinement. *Formal Aspects of Computing*, 20(3):277–294, 2008.

[Low09] Gavin Lowe. On CSP refinement tests that run multiple copies of a process. In *Proceedings of the Seventh International Workshop on Automated Verification of Critical Systems (AVoCS '07)*, volume 250 of *Electronic Notes in Theoretical Computer Science*, pages 153–170, 2009.

[LPS81] Daniel J. Lehmann, Amir Pnueli, and Jonathan Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In *Proceedings of the 8th Colloquium on Automata, Languages and Programming (ICALP 1981)*, volume 115 of *Lecture Notes in Computer Science*, pages 264–277. Springer-Verlag, 1981.

[Mil06] Mark Samuel Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control.* PhD thesis, Johns Hopkins University, 2006.

[ML07] Toby Murray and Gavin Lowe. Authority analysis for least privilege environments. In *Proceedings of the Joint*

Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA '07), pages 113–130, 2007.

[ML09] Toby Murray and Gavin Lowe. On refinement-closed security properties and nondeterministic compositions. In *Proceedings of the Eighth International Workshop on Automated Verification of Critical Systems (AVoCS '08)*, volume 250 of *Electronic Notes in Theoretical Computer Science*, pages 49–68, 2009.

[Muk93] Abida Mukarram. *A Refusal Testing Model for CSP*. D.Phil. thesis, University of Oxford, 1993.

[Mur10] Toby Murray. *Analysing the Security Properties of Object-Capability Patterns*. D.Phil. thesis, University of Oxford, 2010.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57, November 1977.

[Puh03] Antti Puhakka. Using fairness in process-algebraic verification. Technical Report 24, Institute of Software Systems, Tampere University of Technology, 2003.

[Puh05] Antti Puhakka. Using fairness constraints in process-algebraic verification. In *Proceedings of the Second International Colloquium on Theoretical Aspects of Computing (ICTAC 2005)*, volume 3722 of *Lecture Notes in Computer Science*, pages 546–561. Springer, 2005.

[PV01] Antti Puhakka and Antti Valmari. Liveness and fairness in process-algebraic verification. In *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR '01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 202–217. Springer, 2001.

[RGG$^+$95] A. W. Roscoe, Paul H. B. Gardiner, M. H. Goldsmith, J. R. Hulance, D. M. Jackson, and J. B. Scattergood. Hierarchical compression for model-checking CSP or how to check 10^{20} dining philosophers for deadlock. In *Proceedings of the First International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS '95)*, pages 133–152, London, UK, 1995. Springer-Verlag.

[Ros94] A. W. Roscoe. Model-checking CSP. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*, pages 353–378. Prentice-Hall, 1994.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, Upper Saddle River, NJ, USA, 1997. Available at: `http://www.comlab.ox.ac.uk/people/bill.roscoe/publications/68b.pdf`.

[Ros01] A. W. Roscoe. Compiling shared variable programs into CSP. In *Proceedings of the 2001 PROGRESS Workshop*, 2001. Available at: `http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/82.ps`.

[Ros04] A. W. Roscoe. Finitary refinement checks for infinitary specifications. In *Proceedings of Communicating Process Architectures (CPA 2004)*, 2004.

[Ros05a] A. W. Roscoe. On the expressive power of CSP refinement. *Formal Aspects of Computing*, 17(2):93–112, August 2005.

[Ros05b] A. W. Roscoe. Seeing beyond divergence. In *Proceedings of Communicating Sequential Processes: the first 25 years: Symposium on the Occasion of 25 Years of CSP, July 7-8, 2004*, volume 3525 of *Lecture Notes in Computer Science*, page 15. Springer-Verlag, 2005.

[Ros08] A. W. Roscoe. The three platonic models of divergence-strict CSP. In *Proceedings of the 5th International Colloquium on Theoretical Aspects of Computing (ICTAC 2008)*, volume 5160 of *Lecture Notes in Computer Science*, pages 23–49. Springer-Verlag, 2008.

[Ros09] A. W. Roscoe. Revivals, stuckness and the hierarchy of CSP models. *Journal of Logic and Algebraic Programming*, 78(3):163–190, 2009.

[RSR04] J. N. Reed, J. E. Sinclair, and A. W. Roscoe. Responsiveness of interoperating components. *Formal Aspects of Computing*, 16(4):394–411, 2004.

[Sis94] A. P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994.

[SLD09] Jun Sun, Yang Liu, and Jin Song Dong. Model checking CSP revisited: Introducing a Process Analysis Toolkit. In *Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science*, pages 307–322. Springer, 2009.

[SLDW08] Jun Sun, Yang Liu, Jin Song Dong, and Hai H. Wang. Specifying and verifying event-based fairness enhanced systems. In *Formal Methods and Software Engineering, Proceedings of the 10th International Conference on Formal Engineering Methods (ICFEM '08)*, pages 5–24. Springer-Verlag, 2008.

[SRI09] D. Gift Samuel, Markus Roggenbach, and Yoshinao Isobe. The stable revivals model in CSP-Prover. In *Proceedings of the Eighth International Workshop on Automated Verification of Critical Systems (AVoCS '08)*, volume 250 of *Electronic Notes in Theoretical Computer Science*, pages 119–134. Elsevier Science Publishers B. V., 2009.

[VVK05] Hagen Völzer, Daniele Varacca, and Ekkart Kindler. Defining fairness. In *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR '05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 458–472. Springer, 2005.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First IEEE Symposium on Logic in Computer Science (LICS '86)*, pages 322–331, 1986.

[WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (SFCS '83)*, pages 185–194. IEEE Computer Society, 1983.

A. Ancillary Proofs

Proof of Theorem 12. Given an FNDF predicate *Pred* from **Pred**, our strategy is to construct a safety predicate $Pred_S$ that contains *Pred* as a subset. We also construct a liveness predicate $Pred_L$ that contains *Pred* and then show that their intersection is *Pred*.

To construct $Pred_S$, we define the safety predicate $Safe(Pred)$ for which the (bad) thing that it asserts cannot happen is a set of executions that cannot be extended so as to satisfy *Pred*. So $Safe(Pred)$ contains systems all of whose observations (*i.e.* finite sets of finite linear observations) can be extended to satisfy *Pred*.

$$Safe(Pred) = \left\{ \mathcal{FL}[\![P]\!] \;\middle|\; P \in CSP \wedge \begin{array}{l} \forall\, M \,.\, |M| \in \mathbb{N} \wedge M \subseteq \mathcal{FL}[\![Sys]\!] \Rightarrow \\ \quad (\exists\, P' \in CSP \,.\, M \subseteq \mathcal{FL}[\![P']\!] \wedge \mathcal{FL}[\![P']\!] \in Pred) \end{array} \right\}$$

We show that $Safe(Pred)$ is a safety predicate. Consider any system $Sys \in CSP$ for which $\mathcal{FL}[\![Sys]\!] \notin Safe(Pred)$. Then there exists some finite set M where $M \subseteq \mathcal{FL}[\![Sys]\!]$ and

$$\forall\, Sys' \in CSP \,.\, M \subseteq \mathcal{FL}[\![Sys']\!] \Rightarrow \mathcal{FL}[\![Sys']\!] \notin Pred. \tag{22}$$

Because $M \subseteq \mathcal{FL}[\![Sys]\!]$, we have that $\mathcal{FL}[\![Sys]\!] \notin Pred$. Hence, $\mathcal{FL}[\![Sys]\!] \notin Safe(Pred) \Rightarrow \mathcal{FL}[\![Sys]\!] \notin Pred$ and so $(\exists\, Sys' \in CSP \,.\, M \subseteq \mathcal{FL}[\![Sys']\!] \wedge \mathcal{FL}[\![Sys']\!] \notin Safe(Pred)) \Rightarrow (\exists\, Sys' \in CSP \,.\, M \subseteq \mathcal{FL}[\![Sys']\!] \wedge \mathcal{FL}[\![Sys']\!] \notin Pred)$ since in both cases we can set $Sys' = Sys$. It follows that

$$\begin{array}{l}(\forall\, Sys' \in CSP \,.\, M \subseteq \mathcal{FL}[\![Sys']\!] \Rightarrow \mathcal{FL}[\![Sys']\!] \notin Pred) \Rightarrow \\ \quad (\forall\, Sys' \in CSP \,.\, M \subseteq \mathcal{FL}[\![Sys']\!] \Rightarrow \mathcal{FL}[\![Sys']\!] \notin Safe(Pred)).\end{array} \tag{23}$$

So, combining Equations 22 and 23,

$$\forall\, Sys' \in CSP \,.\, M \subseteq \mathcal{FL}[\![Sys']\!] \Rightarrow \mathcal{FL}[\![Sys']\!] \notin Safe(Pred).$$

Hence $Safe(Pred)$ satisfies Definition 8 and so is a safety predicate.

To construct $Pred_L$ we define the liveness predicate $Live(Pred)$ that asserts that it's always possible either to satisfy *Pred* or for satisfying *Pred* to become impossible due to $Safe(Pred)$ having been violated. Formally

$$Live(Pred) = Pred \cup \overline{Safe(Pred)}.$$

We show that $Live(Pred)$ is a liveness predicate. Consider any partial observation M where $|M| \in \mathbb{N}$ and $M \subseteq \mathcal{FL}[\![P]\!] \cap PLO$ for some process $P \in CSP$. Suppose there exists some process $P' \in CSP$ for which $M \subseteq \mathcal{FL}[\![P']\!]$ and $\mathcal{FL}[\![P']\!] \in Pred$. Then $\mathcal{FL}[\![P']\!] \in Live(Pred)$ as required. Otherwise, we must have that for all processes $P' \in CSP$, if $M \subseteq \mathcal{FL}[\![P']\!]$ then $\mathcal{FL}[\![P']\!] \notin Pred$. Let P' be an arbitrary process such that $M \subseteq \mathcal{FL}[\![P']\!]$. Then following the same reasoning that led to Equation 22, $\mathcal{FL}[\![P']\!] \notin Safe(Pred)$ so $\mathcal{FL}[\![P']\!] \in \overline{Safe(Pred)}$. So $\mathcal{FL}[\![P']\!] \in Live(Pred)$ again. Hence $Live(Pred)$ satisfies Definition 11 and so is a liveness predicate.

We now show that $Pred \subseteq Safe(Pred)$. Consider any process $P \in CSP$ for which $\mathcal{FL}[\![P]\!] \in Pred$. Then for any finite set $M \subseteq \mathcal{FL}[\![P]\!]$ there exists a process P' such that $M \subseteq \mathcal{FL}[\![P']\!]$ and $\mathcal{FL}[\![P']\!] \in Pred$, namely P itself. So $\mathcal{FL}[\![P]\!] \in Safe(Pred)$. Hence, $Safe(Pred) = Pred \cup Safe(Pred)$.

Let $Pred_S = Safe(Pred)$ and $Pred_L = Live(Pred)$. Then

$$\begin{aligned} Pred_S \cap Pred_L &= Safe(Pred) \cap Live(Pred) \\ &= (Pred \cup Safe(Pred)) \cap (Pred \cup \overline{Safe(Pred)}) \\ &= Pred \cap (Safe(Pred) \cup \overline{Safe(Pred)}) \\ &= Pred \cap \{\mathcal{FL}[\![P]\!] \mid P \in CSP\} \\ &= Pred. \end{aligned}$$

□

Lemma 29. Let $\mathcal{M}$ and $\mathcal{M}'$ be two CSP models such that $M \preceq \mathcal{M}'$. Then for all $Pred \in \mathbf{Pred}$,

$$\forall\, P \in CSP \,.\, RC_{\mathcal{M}}(Pred)(P) \Rightarrow RC_{\mathcal{M}'}(Pred)(P).$$

Proof. Suppose the conditions of the lemma and consider some predicate $Pred \in \mathbf{Pred}$ and process $P \in CSP$ such that $RC_{\mathcal{M}}(Pred)(P)$ holds. Let $Q \in CSP$ be a process such that $P \sqsubseteq_{\mathcal{M}'} Q$; we show that $Pred(Q)$ holds. Since $\mathcal{M} \preceq \mathcal{M}'$ and $P \sqsubseteq_{\mathcal{M}'} Q$, we have $P \sqsubseteq_{\mathcal{M}} Q$. Hence, since $RC_{\mathcal{M}}(Pred)(P)$, $Pred(Q)$ holds. □