A STATIC ANALYSIS FRAMEWORK
FOR LIVELOCK FREEDOM IN CSP [*]

JOËL OUAKNINE [a], HRISTINA PALIKAREVA [b], A. W. ROSCOE [c], AND JAMES WORRELL [d]

[a,c,d] Department of Computer Science, Oxford University, UK
     e-mail address: {joel, awr, jbw}@cs.ox.ac.uk

[b] Department of Computing, Imperial College London, UK
     e-mail address: h.palikareva@imperial.ac.uk

ABSTRACT. In a process algebra with hiding and recursion it is possible to create processes which compute internally without ever communicating with their environment. Such processes are said to diverge or livelock. In this paper we show how it is possible to conservatively classify processes as livelock-free through a static analysis of their syntax. In particular, we present a collection of rules, based on the inductive structure of terms, which guarantee livelock-freedom of the denoted process. This gives rise to an algorithm which conservatively flags processes that can potentially livelock. We illustrate our approach by applying both BDD-based and SAT-based implementations of our algorithm to a range of benchmarks, and show that our technique in general substantially outperforms the model checker FDR whilst exhibiting a low rate of inconclusive results.

1. INTRODUCTION

It is standard in process algebra to distinguish between the visible and invisible (or silent) actions of a process. The latter correspond to state changes arising from internal computations such as resolving of nondeterminism, unfolding of a recursion, abstraction of details. Their occurrence is silent and is not detectable or controllable by the environment. A process is said to *diverge* or *livelock* if it reaches a state from which it may forever compute internally through an infinite sequence of invisible actions. This is usually a highly undesirable feature of the process, described in the literature as "even worse than deadlock, in that like an endless loop it may consume unbounded computing resources without achieving anything" [10, page 156]. Livelock invalidates certain analysis methodologies, e.g., it signifies lack of progress, and is often symptomatic of a bug in the modelling. However the possibility of writing down divergent processes arises from the presence of two crucial

constructs, recursion and hiding. The latter converts visible actions into invisible ones and is a key device for abstraction.

We distinguish two ways in which a process may livelock. In the first, a process may be able to communicate an infinite unbroken sequence of some visible event, and this process then occurs inside the scope of an operator which hides that event. Alternatively, a process may livelock owing to the presence of an unguarded recursion. Roughly speaking, the latter means that the process may recurse without first communicating a visible action.

This paper is concerned with the problem of determining whether a process may livelock in the context of the process algebra CSP, although the principles upon which our analysis is based should be transferable to other process algebras as well. While it is straightforward to show that the problem is in general undecidable[1], we are still able to provide a conservative (i.e., sound but incomplete) method of checking for the possibility of livelock: this method either correctly asserts that a given process is livelock-free, or is inconclusive. The algorithm is based on a static analysis[2] of the given process, principally in terms of the interaction of hiding, renaming, and recursion. This analysis naturally divides into two parts according to the two sources of livelock outlined above.

The basic intuitions underlying our approach are fairly straightforward. In part they mirror the guardedness requirements which ensure that well-behaved CSP process equations have unique, livelock-free fixed points [21, Chap. 8]. However, we extend the treatment of [21] by allowing guarded recursions to include instances of the hiding operator. Incidentally, Milner's notion of guarded recursions in CCS is similarly restricted by the requirement that variables not occur inside parallel compositions [15]. Complications arise mainly because we want to be able to fully incorporate hiding and renaming in our treatment, both of which can have subtle indirect effects on guardedness.

We note that the idea of guarded recursions is standard in process algebra. For instance, in Milner's framework, a variable is 'strongly guarded' in a given term if every free occurrence of the variable in the term occurs within the scope of a prefixing operator [15]. This notion is introduced in order to justify certain proof principles, such as that guaranteeing the uniqueness of fixed points up to bisimilarity. Strong guardedness has also been extended to a calculus with hiding and action refinement [3]. A key difference between our approach and these notions is that we seek to guarantee livelock-freedom, rather than merely the existence of unique fixed points.

In fact, there are few papers which deal with the problem of guaranteeing livelock-freedom in the setting of concurrent process calculi.[3] The existing work on livelock-freedom has mostly been carried out in the context of mobile calculi. [23] presents an approach for guaranteeing livelock-freedom for a certain fragment of the π-calculus. Unlike the combinatorial treatment presented here, this approach makes use of the rich theory of types of the π-calculus, and in particular the technique of logical relations. Another study of divergence-freedom in the π-calculus appears in [30], and uses the notions of graph types.

Note that CSP is predicated upon *synchronous* communication. In terms of livelock analysis, different issues (and additional difficulties) arise in an asynchronous context (assuming unbounded communication buffers); see, e.g., [13, 14].

---

[1]For example, CSP can encode counters, and is therefore Turing-powerful.

[2]Here *static analysis* is used to distinguish our approach from the state-space exploration methods that underlie model checking or refinement checking.

[3]In contrast, there are numerous works treating termination for the λ-calculus or combinatory logic [8, 16, 7].

Of course, one way to check a process for divergence is to search for reachable cycles of silent actions in its state space, which is a labelled transition system built from the operational semantics. Assuming this graph is finite, this can be achieved by calculating its strongly connected components, using, e.g., Tarjan's algorithm [5]. The latter can be carried out in time linear in the size of the graph, which may however be exponential (or worse) in the syntactic size of the term describing the process. By circumventing the state-space exploration, we obtain a static analysis algorithm which in practice tends to substantially outperform state-of-the-art model-checking tools such as FDR—see Section 7 for experimental comparisons.

Naturally, there is a trade-off between the speed and accuracy of livelock checking. It is not hard to write down processes which are livelock-free but which our analysis indicates as potentially divergent. However, when modelling systems in practice, it makes sense to try to check for livelock-freedom using a simple and highly economical static analysis before invoking computationally expensive state-space exploration algorithms. Indeed, as Roscoe [21, page 208] points out, the calculations required to determine if a process diverges are significantly more costly than those for deciding other aspects of refinement, and it is advantageous to avoid these calculations if at all possible.

Recent works in which CSP livelock-freedom plays a key role include [6] as well as [25, 24]; see also references within.

2. CSP: Syntax and Conventions

Let Σ be a finite set of *events*, with $\checkmark \notin \Sigma$. We write $\Sigma^{\checkmark}$ to denote $\Sigma \cup \{\checkmark\}$ and $\Sigma^{*\checkmark}$ to denote the set of finite sequences of elements from Σ which may end with $\checkmark$. In the notation below, we have $a \in \Sigma$ and $A \subseteq \Sigma$. R denotes a binary (renaming) relation on Σ; its lifting to $\Sigma^{\checkmark}$ is understood to relate $\checkmark$ to itself. The variable X is drawn from a fixed infinite set of process variables.

CSP terms are constructed according to the following grammar:

$$P ::= STOP \mid a \longrightarrow P \mid SKIP \mid P_1 \sqcap P_2 \mid P_1 \square P_2 \mid P_1 \underset{A}{\parallel} P_2 \mid$$

$$P_1 \mathbin{;} P_2 \mid P \setminus A \mid P[R] \mid X \mid \mu X \boldsymbol{.} P \mid DIV .$$

$STOP$ is the deadlocked process. The prefixed process $a \longrightarrow P$ initially offers to engage in the event a, and subsequently behaves like P. $SKIP$ represents successful termination, and is willing to communicate $\checkmark$ at any time. $P \square Q$ denotes the external choice of P and Q, whereas $P \sqcap Q$ denotes the internal (or nondeterministic) alternative. The distinction is orthogonal to our concerns, and indeed both choice operators behave identically over our denotational model. The parallel composition $P_1 \underset{A}{\parallel} P_2$ requires P_1 and P_2 to synchronise (i.e., handshake) on all events in A, and to behave independently of each other with respect to all other events. $P \mathbin{;} Q$ is the sequential composition of P and Q: it denotes a process which behaves like P until P chooses to terminate (silently), at which point the process seamlessly starts to behave like Q. $P \setminus A$ is a process which behaves like P but with all communications in the set A hidden. The renamed process $P[R]$ derives its behaviours from those of P in that, whenever P can perform an event a, $P[R]$ can engage in any event b such that $a \mathrel{R} b$. To understand the meaning of $\mu X \boldsymbol{.} P$, consider the equation $X = P$, in terms of the unknown X. While this equation may have several solutions, it always has a

unique least[4] such, written $\mu\,X\,\textbf{.}\,P$. Moreover, as it turns out, if $\mu\,X\,\textbf{.}\,P$ is livelock-free then the equation $X = P$ has no other solutions. Lastly, the process DIV represents livelock, i.e., a process caught in an infinite loop of silent events.

A CSP term is *closed* if every occurrence of a variable X in it occurs within the scope of a $\mu\,X$ operator; we refer to such terms as *processes*. We denote by $\overline{\mathsf{CSP}}$ the set of all CSP processes and by CSP the set of all CSP terms, both open and closed.

Let us state a few conventions. When hiding a single event a, we write $P \setminus a$ rather than $P \setminus \{a\}$. For R a renaming relation on Σ and $U \subseteq \Sigma$, we denote by $R(U)$ the set $\{y \mid \exists x \in U \,\textbf{.}\, x\;R\;y\}$. The binding scope of the $\mu\,X$ operator extends as far to the right as possible. We also often express recursions by means of the equational notation $X = P$, rather than the functional $\mu\,X\,\textbf{.}\,P$.

Let us also remark that CSP processes are often defined via *vectors* of mutually recursive equations. These can always be converted to our present syntax, thanks to Bekič's theorem [28, Chap. 10].[5] Accordingly, we shall freely make use of the vectorised notation in this paper, viewed as syntactic sugar.

3. Operational and Denotational Semantics

We present congruent (equivalent) operational and denotational semantics for CSP. For reasons of space, some details and clauses are omitted. An extensive treatment of a variety of different CSP models can also be found in [21, 22]. The semantics presented below only distill those ideas from [21, 22] that are relevant in our setting.

3.1. **Operational semantics.** The operational semantics is presented as a list of inference rules in SOS form. In what follows, a stands for a visible event, i.e., belongs to $\Sigma^{\checkmark}$. $A \subseteq \Sigma$ and $A^{\checkmark} = A \cup \{\checkmark\}$. γ can be a visible event or a silent one ($\gamma \in \Sigma^{\checkmark} \cup \{\tau\}$). $P \xrightarrow{\gamma} P'$ means that P can perform an immediate and instantaneous γ-transition, and subsequently become P' (communicating γ in the process if γ is a visible event). If P is a term with a single free variable X and Q is a process, $[Q/X]P$ represents the process P with Q substituted for every free occurrence of X.

$$\overline{(a \longrightarrow P) \xrightarrow{a} P} \qquad \overline{SKIP \xrightarrow{\checkmark} STOP}$$

$$\overline{P_1 \sqcap P_2 \xrightarrow{\tau} P_1} \qquad \overline{P_1 \sqcap P_2 \xrightarrow{\tau} P_2}$$

$$\frac{P_1 \xrightarrow{\tau} P_1'}{P_1 \,\square\, P_2 \xrightarrow{\tau} P_1' \,\square\, P_2} \qquad \frac{P_2 \xrightarrow{\tau} P_2'}{P_1 \,\square\, P_2 \xrightarrow{\tau} P_1 \,\square\, P_2'}$$

---

[4]The relevant partial order is defined in Section 3.

[5]Our rules for livelock detection require that processes be defined using the fixed-point operator μ, as opposed to systems of mutually recursive process definitions. Bekič's theorem expresses fixed points of self-maps on the product space $X \times Y$ in terms of fixed points of self-maps on the respective components X and Y. For example, consider a mutually recursive process definition of the form $P = f(P, Q)$, $Q = g(P, Q)$. The idea is first to define a parameterised fixed point of g via the expression $\mu\,Y.g(X, Y)$, and then substitute in the expression for P, yielding $P = \mu\,X.f(X, \mu\,Y.g(X, Y))$. This process can be generalised to transform mutually recursive definitions of arbitrary dimension into expressions using only the single-variable fixed-point operator μ.

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \,\square\, P_2 \xrightarrow{a} P_1'} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 \,\square\, P_2 \xrightarrow{a} P_2'}$$

$$\frac{P_1 \xrightarrow{\gamma} P_1'}{P_1 \underset{A}{\parallel} P_2 \xrightarrow{\gamma} P_1' \underset{A}{\parallel} P_2} \, [\,\gamma \notin A^{\checkmark}\,] \qquad \frac{P_2 \xrightarrow{\gamma} P_2'}{P_1 \underset{A}{\parallel} P_2 \xrightarrow{\gamma} P_1 \underset{A}{\parallel} P_2'} \, [\,\gamma \notin A^{\checkmark}\,]$$

$$\frac{P_1 \xrightarrow{a} P_1' \quad P_2 \xrightarrow{a} P_2'}{P_1 \underset{A}{\parallel} P_2 \xrightarrow{a} P_1' \underset{A}{\parallel} P_2'} \, [\, a \in A^{\checkmark}\,]$$

$$\frac{P_1 \xrightarrow{\checkmark} P_1'}{P_1 \,\mathbin{;}\, P_2 \xrightarrow{\tau} P_2} \qquad \frac{P_1 \xrightarrow{\gamma} P_1'}{P_1 \,\mathbin{;}\, P_2 \xrightarrow{\gamma} P_1' \,\mathbin{;}\, P_2} \, [\,\gamma \neq \checkmark\,]$$

$$\frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{\tau} P' \setminus A} \, [\, a \in A\,] \qquad \frac{P \xrightarrow{\gamma} P'}{P \setminus A \xrightarrow{\gamma} P' \setminus A} \, [\,\gamma \notin A\,]$$

$$\frac{P \xrightarrow{\tau} P'}{P[R] \xrightarrow{\tau} P'[R]} \qquad \frac{P \xrightarrow{a} P'}{P[R] \xrightarrow{b} P'[R]} \, [\, a \, R \, b\,]$$

$$\frac{}{\mu X \boldsymbol{.} P \xrightarrow{\tau} [(\mu X \boldsymbol{.} P)/X]P} \qquad \frac{}{DIV \xrightarrow{\tau} DIV} \, .$$

These rules allow us to associate to any CSP process a labelled transition system (LTS) representing its possible executions. We say that a process *diverges* if it has an infinite path whose actions are exclusively τ's. A process is *livelock-free* if it never reaches a point from which it diverges.

3.2. **Denotational semantics.** The denotational semantics ascribes to any CSP process a pair $(\mathsf{traces}_{\perp}(P), \mathsf{divergences}(P))$, where $\mathsf{traces}_{\perp}(P) = \mathsf{traces}(P) \cup \mathsf{divergences}(P) \subseteq \Sigma^{*\checkmark}$ is the set of all finite visible-event traces that P may perform, and $\mathsf{divergences}(P) \subseteq \mathsf{traces}_{\perp}(P)$ is the set of traces after which it may diverge.[6] Following [22], we write $\mathcal{T}^{\Downarrow}$ for the set of pairs $(T, D) \in \mathcal{P}(\Sigma^{*\checkmark}) \times \mathcal{P}(\Sigma^{*\checkmark})$ satisfying the following axioms (where $\frown$ denotes trace concatenation):

(1) $D \subseteq T$.
(2) $s \frown \langle \checkmark \rangle \in D$ implies $s \in D$.
(3) $T \subseteq \Sigma^{*\checkmark}$ is non-empty and prefix-closed.
(4) $s \in D \cap \Sigma^{*}$ and $t \in \Sigma^{*\checkmark}$ implies $s \frown t \in D$.

Axiom 4 says that the set of divergences is postfix-closed. Indeed, since we are only interested in *detecting* divergence, we treat it as catastrophic and do not attempt to record any meaningful information past a point from which a process may diverge; accordingly, our semantic model takes the view that a process may perform *any* sequence of events after divergence. Thus the only reliable behaviours of a process are those in $T - D$.

Axiom 2 reflects the intuition that $\checkmark$ represents successful termination. In particular, there is no way a process may diverge after a $\checkmark$ unless it is already divergent.

---

[6]Standard models of CSP also take account of the liveness properties of a process by modelling its *refusals*, i.e., the sets of events it cannot perform after a given trace. However, this information is orthogonal to our concerns: the divergences of a process are independent of its refusals—see [21, Section 8.4].

$$\mathsf{traces}(STOP) = \{\langle\rangle\}$$
$$\mathsf{traces}(SKIP) = \{\langle\rangle, \langle\checkmark\rangle\}$$
$$\mathsf{traces}(DIV) = \emptyset$$
$$\mathsf{traces}(a \longrightarrow P) = \{\langle\rangle\} \cup \{\langle a\rangle^\frown t \mid t \in \mathsf{traces}(P)\}$$
$$\mathsf{traces}(P \,\square\, Q) = \mathsf{traces}(P) \cup \mathsf{traces}(Q)$$
$$\mathsf{traces}(P \,\sqcap\, Q) = \mathsf{traces}(P) \cup \mathsf{traces}(Q)$$
$$\mathsf{traces}(P \,\fatsemi\, Q) = (\mathsf{traces}(P) \cap \Sigma^*) \cup \{t^\frown s \mid t^\frown\langle\checkmark\rangle \in \mathsf{traces}(P), s \in \mathsf{traces}(Q)\}$$
$$\mathsf{traces}(P \setminus A) = \{t \restriction (\Sigma \setminus A) \mid t \in \mathsf{traces}(P)\}$$
$$\mathsf{traces}(P[R]) = \{t \mid \exists\, s \in \mathsf{traces}(P) \boldsymbol{.} s\ R\ t\}$$
$$\mathsf{traces}(P \parallel_A Q) = \bigcup \{s \parallel_A t \mid s \in \mathsf{traces}(P), t \in \mathsf{traces}(Q)\}$$

Figure 1: The model $\mathcal{T}^{\Downarrow}$: inductive rules for calculating traces.

Given a process P, its denotation $[\![P]\!] = (\mathsf{traces}_\perp(P), \mathsf{divergences}(P)) \in \mathcal{T}^{\Downarrow}$ is calculated by induction on the structure of P; in other words, the model $\mathcal{T}^{\Downarrow}$ is compositional. The complete list of clauses can be found in [21, Chap. 8], and moreover the traces and divergences of a process may also be extracted from the operational semantics in straightforward fashion. We provide the inductive rules in Figures 1 and 2 to facilitate the proofs. In the last three rules in Figure 2, r ranges over $\Sigma^{*\checkmark}$, in accordance with Axiom 4. The lifting of the renaming relation R to traces is carried out element-wise. The precise definition of $s \parallel_A t$ in the rule for parallel composition is presented in Figure 3 [21].

Definition 3.1. A process P is *livelock-free* if $\mathsf{divergences}(P) = \emptyset$.

3.2.1. *Reasoning About Infinite Traces.* In general, reasoning about livelock requires reasoning about infinite behaviours. Hiding a set of events $A \subseteq \Sigma$ from a process P introduces divergence if P is capable of performing an infinite unbroken sequence of events from A. Although our model only records the finite traces of a process, the finitely branching nature of our operators[7] entails (via König's lemma) that a process may perform an infinite trace u if and only if it can perform all finite prefixes of u. In other words, the set of finite traces of a process conveys enough information for deducing the set of its infinite traces as well. To keep the notation simple, given an infinite trace $u \in \Sigma^\omega$, we will write

$$u \in \mathsf{traces}^\omega(P) \text{ whenever } \{t \in \Sigma^* \mid t < u\} \subseteq \mathsf{traces}(P),$$

where $<$ denotes the strong prefix order on $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. Furthermore, we will write $\mathsf{traces}^\infty(P)$ to denote $\mathsf{traces}(P) \cup \mathsf{traces}^\omega(P)$, the set of all finite and infinite traces of P. We note that traces in $\mathsf{traces}^\omega(P)$, and hence finite prefixes thereof, cannot contain $\checkmark$, which denotes successful termination.

---

[7] All CSP operators are finitely branching under the assumptions that the alphabet Σ is finite and that there is no unbounded nondeterminism [21].

$$\text{divergences}(STOP) = \emptyset$$
$$\text{divergences}(SKIP) = \emptyset$$
$$\text{divergences}(DIV) = \Sigma^{*\checkmark}$$
$$\text{divergences}(a \longrightarrow P) = \{\langle a \rangle ^\frown t \mid t \in \text{divergences}(P)\}$$
$$\text{divergences}(P \,\square\, Q) = \text{divergences}(P) \cup \text{divergences}(Q)$$
$$\text{divergences}(P \,\sqcap\, Q) = \text{divergences}(P) \cup \text{divergences}(Q)$$
$$\text{divergences}(P \,\mathbin{;}\, Q) = \text{divergences}(P) \cup \{t ^\frown s \mid t ^\frown \langle \checkmark \rangle \in \text{traces}_\bot(P), s \in \text{divergences}(Q)\}$$
$$\text{divergences}(P \setminus A) = \{(t \restriction (\Sigma \setminus A)) ^\frown r \mid t \in \text{divergences}(P)\} \cup$$
$$\{(u \restriction (\Sigma \setminus A)) ^\frown r \mid u \in \Sigma^\omega, u \restriction (\Sigma \setminus A) \text{ finite}, \forall\, t < u \mathbin{\raisebox{0.1ex}{\tiny$\bullet$}} t \in \text{traces}_\bot(P)\}$$
$$\text{divergences}(P[R]) = \{t ^\frown r \mid \exists\, s \in \text{divergences}(P) \cap \Sigma^* \mathbin{\raisebox{0.1ex}{\tiny$\bullet$}} s \, R \, t\}$$
$$\text{divergences}(P \,\|_A\, Q) = \{u ^\frown r \mid \exists\, s \in \text{traces}_\bot(P), \exists\, t \in \text{traces}_\bot(Q) \mathbin{\raisebox{0.1ex}{\tiny$\bullet$}} u \in (s \,\|_A\, t \cap \Sigma^*),$$
$$(s \in \text{divergences}(P) \text{ or } t \in \text{divergences}(Q))\}$$

Figure 2: The model $\mathcal{T}^{\Downarrow}$: inductive rules for calculating divergences.

$$s \,\|_A\, t = t \,\|_A\, s$$
$$\langle \rangle \,\|_A\, \langle \rangle = \{\langle \rangle\}$$
$$\langle \rangle \,\|_A\, \langle a \rangle = \{\langle \rangle\}$$
$$\langle \rangle \,\|_A\, \langle b \rangle = \{\langle b \rangle\}$$
$$\langle a \rangle ^\frown s \,\|_A\, \langle b \rangle ^\frown t = \{\langle b \rangle ^\frown u \mid u \in \langle a \rangle ^\frown s \,\|_A\, t\}$$
$$\langle a \rangle ^\frown s \,\|_A\, \langle a \rangle ^\frown t = \{\langle a \rangle ^\frown u \mid u \in s \,\|_A\, t\}$$
$$\langle a \rangle ^\frown s \,\|_A\, \langle a' \rangle ^\frown t = \{\} \quad \text{if } a \neq a'$$
$$\langle b \rangle ^\frown s \,\|_A\, \langle b' \rangle ^\frown t = \{b ^\frown u \mid u \in s \,\|_A\, \langle b' \rangle ^\frown t\} \cup \{b' ^\frown u \mid u \in \langle b \rangle ^\frown s \,\|_A\, t\}$$

Figure 3: Interleaving operator on traces (where $s, t \in \Sigma^{*\checkmark}$, $A \subseteq \Sigma^\checkmark$, $a \in A$, $b \notin A$).

We now state the semantic properties we use in case of infinite traces. The proofs for all lemmas can be found in Appendix A.

Lemma 3.2. *Let $u \in \text{traces}^\omega(a \longrightarrow P)$. Then there exists $u' \in \text{traces}^\omega(P)$ such that $u = \langle a \rangle ^\frown u'$.*

Lemma 3.3. *Let $u \in \text{traces}^\omega(P \oplus Q)$ for $\oplus \in \{\square, \sqcap\}$. Then $u \in \text{traces}^\omega(P)$ or $u \in \text{traces}^\omega(Q)$.*

Lemma 3.4. *Let $u \in \mathsf{traces}^\omega(P \,\mathring{,}\, Q)$. Then $u \in \mathsf{traces}^\omega(P)$ or $u = t ^\frown u'$ with $t ^\frown \langle\checkmark\rangle \in \mathsf{traces}(P)$, $u' \in \mathsf{traces}^\omega(Q)$.*

Lemma 3.5. *Let $u \in \mathsf{traces}^\omega(P \setminus A)$ and $P \setminus A$ be livelock-free. Then there exists $v \in \mathsf{traces}^\omega(P)$ such that $u = v \upharpoonright (\Sigma \setminus A)$.*

Lemma 3.6. *Let $u \in \mathsf{traces}^\omega(P[R])$. Then there exists $v \in \mathsf{traces}^\omega(P)$ such that $v \, R \, u$.*

Lemma 3.7. *Let $u \in \mathsf{traces}^\omega(P \underset{A}{\|} Q)$. Then there exist $u_1 \in \mathsf{traces}^\infty(P)$ and $u_2 \in \mathsf{traces}^\infty(Q)$ such that $u \in u_1 \underset{A}{\|} u_2$, and $u_1 \in \Sigma^\omega$ or $u_2 \in \Sigma^\omega$.*

3.2.2. *Handling Recursion.* We interpret recursive processes in the standard way by introducing a partial order $\sqsubseteq$ on $\mathcal{T}^{\Downarrow}$. We write $(T_1, D_1) \sqsubseteq (T_2, D_2)$ if $T_2 \subseteq T_1$ and $D_2 \subseteq D_1$. In other words, the order on $\mathcal{T}^{\Downarrow}$ is reverse inclusion on both the trace and the divergence components. The resulting partial order $(\mathcal{T}^{\Downarrow}, \sqsubseteq)$ is a complete lattice. The bottom element of $(\mathcal{T}^{\Downarrow}, \sqsubseteq)$ is $(\Sigma^{*\checkmark}, \Sigma^{*\checkmark})$, i.e., the denotation of the immediately divergent process DIV. The top element is $(\{\langle\rangle\}, \emptyset)$, i.e., the denotation of the immediately deadlocking process $STOP$. The least upper bound and the greatest lower bound of a family $\{(T_i, D_i) \mid i \in I\}$ are given by $\bigsqcup_{i \in I}(T_i, D_i) = (\bigcap_{i \in I} T_i, \bigcap_{i \in I} D_i)$ and $\bigsqcap_{i \in I}(T_i, D_i) = (\bigcup_{i \in I} T_i, \bigcup_{i \in I} D_i)$, respectively.

It is readily verified that each n-ary CSP operator other than recursion can be interpreted as a Scott-continuous function $(\mathcal{T}^{\Downarrow})^n \to \mathcal{T}^{\Downarrow}$. The continuity of hiding rests on our assumption that Σ is finite (cf. [21, Lemma 8.3.5]). By induction we have that any CSP expression P in variables $X_1, \ldots, X_n$ is interpreted as a Scott-continuous map $(\mathcal{T}^{\Downarrow})^n \to \mathcal{T}^{\Downarrow}$. Recursion is then interpreted using the least fixed point operator $\mathrm{fix} : [\mathcal{T}^{\Downarrow} \to \mathcal{T}^{\Downarrow}] \to \mathcal{T}^{\Downarrow}$. For instance $[\![\mu X \centerdot X]\!]$ is the least fixed point of the identity function on $\mathcal{T}^{\Downarrow}$, i.e., the immediately divergent process. Our analysis of livelock-freedom is based around an alternative treatment of fixed points in terms of metric spaces.

4. A Family of Metrics

In what follows, we make repeated use of standard definitions and facts concerning metric spaces. We refer the reader who might be unfamiliar with this subject matter to the accessible text [26].

Let $F(X)$ be a CSP term with a free variable X. F can be seen as a selfmap of $\mathcal{T}^{\Downarrow}$. Assume that there exists some metric on $\mathcal{T}^{\Downarrow}$ which is complete[8] and under which F is a contraction[9]. Then it follows from the Banach fixed point theorem [26] that F has a unique (possibly divergent) fixed point $\mu X \centerdot F(X)$ in $\mathcal{T}^{\Downarrow}$. Furthermore, starting from *any point* in $\mathcal{T}^{\Downarrow}$, iterated application of F is guaranteed to converge to this unique fixed point.

There may be several such metrics, or none at all. Fortunately, a class of suitable metrics can be systematically elicited from the sets of guards of a particular recursion. Roughly speaking, the metrics that we consider are all variants of the well-known 'longest common

---

[8]A metric space $(\mathcal{T}^{\Downarrow}, d)$ is *complete* if every Cauchy sequence converges.

[9]A selfmap F on a metric space $(\mathcal{T}^{\Downarrow}, d)$ is a *contraction* if there exists a non-negative constant $c < 1$ such that, for any $P, Q \in \mathcal{T}^{\Downarrow}$, $d(F(P), F(Q)) \leqslant c \cdot d(P, Q)$. Intuitively this means that the distance between any $P, Q \in \mathcal{T}^{\Downarrow}$ is strictly greater (by some factor) than the distance between their image under F, as depicted in Figure 4(a).

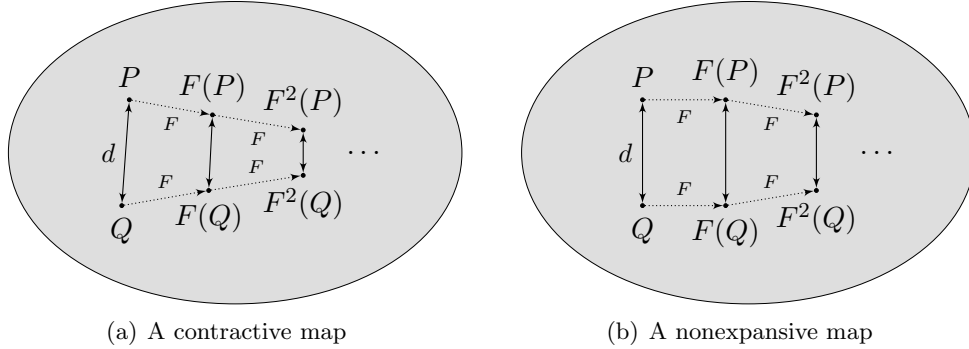(a) A contractive map   (b) A nonexpansive map

Figure 4: Contractive and nonexpansive maps.

prefix' metric on traces[10], which were first studied by Roscoe in his doctoral dissertation [20], and independently by de Bakker and Zucker [2]. The reason we need to consider such variants is that hiding fails to be nonexpansive[11] in the 'longest common prefix' metric. For instance, the distance between the traces $\langle a, a, b \rangle$ and $\langle a, a, c \rangle$ is $\frac{1}{4}$. However, after the event a is hidden, the distance becomes 1. The solution, in this particular case, is to change the definition of the length of a trace by only counting non-a events. To formalise these ideas let us introduce a few auxiliary definitions. These are all parametric in a given set of events $U \subseteq \Sigma$.

Given a trace $s \in \Sigma^{*\checkmark}$, the U-length of s, denoted $\mathsf{length}_U(s)$, is defined to be the number of occurrences of events from U occurring in s. Given a set of traces $T \subseteq \Sigma^{*\checkmark}$ and $n \in \mathbb{N}$ the restriction of T to U-length n is defined by $T \upharpoonright_U n \mathbin{\widehat{=}} \{s \in T \mid \mathsf{length}_U(s) \leqslant n\}$. We extend this restriction operator to act on our semantic domain $\mathcal{T}^{\Downarrow}$ by defining $(T, D) \upharpoonright_U n \mathbin{\widehat{=}} (T', D')$, where

(1) $D' = D \cup \{s^\frown t \mid s \in T \cap \Sigma^* \text{ and } \mathsf{length}_U(s) = n\}$.
(2) $T' = D' \cup \{s \in T \mid \mathsf{length}_U(s) \leqslant n\}$.

Thus $P \upharpoonright_U n$ denotes a process which behaves like P until n events from the set U have occurred, after which it diverges unless it has already terminated. It is the least process which agrees with P on traces with U-length no greater than n.

We now define a metric d_U on $\mathcal{T}^{\Downarrow}$ by

$$d_U(P, Q) \mathbin{\widehat{=}} \inf\{2^{-n} \mid P \upharpoonright_U n = Q \upharpoonright_U n\} \ ,$$

where the infimum is taken in the interval $[0, 1]$.

Proposition 4.1. *Let $U \subseteq \Sigma$. Then $(\mathcal{T}^{\Downarrow}, d_U)$ is an ultrametric space.*

Proof. It is easy to prove that $(\mathcal{T}^{\Downarrow}, d_U)$ satisfies the following laws for each $P, Q, R \in \mathcal{T}^{\Downarrow}$:

---

[10]In this metric the distance between two traces s and t is the infimum in $[0, 1]$ of the set $\{2^{-k} \mid s$ and t possess a common prefix of length $k\}$. I.e., the longer prefix two traces share, the closer they are, with the standard lifting to sets of traces and, therefore, to processes.

[11]A selfmap F on a metric space $(\mathcal{T}^{\Downarrow}, d)$ is *nonexpansive* if, for any $P, Q \in \mathcal{T}^{\Downarrow}$, $d(F(P), F(Q)) \leqslant d(P, Q)$, as illustrated in Figure 4(b).

$$
\begin{array}{rcll}
d_U(P,Q) = 0 & \Leftrightarrow & P = Q & \text{diagonal law} \\
d_U(P,Q) & = & d_U(Q,P) & \text{symmetry} \\
d_U(P,Q) & \leq & d_U(P,R) + d_U(R,Q) & \text{triangle inequality} \\
d_U(P,Q) & \leq & \max(d_U(P,R), d_U(R,Q)) & \text{ultrametric inequality}
\end{array}
$$

The proofs for the first two laws are trivial. Regarding the triangle and ultrametric laws, let us suppose that $d_U(P,R) = 2^{-n}$, $d_U(R,Q) = 2^{-m}$ and $k = \min(n,m)$. Then, $P \upharpoonright_U k = R \upharpoonright_U k = Q \upharpoonright_U k$. Therefore,

$$
d_u(P,Q) \leq 2^{-k} = \max(d_U(P,R), d_U(R,Q)) \leq d_U(P,R) + d_U(R,Q).
$$

$\square$

Notice that the function $U \mapsto d_U$ is antitone: if $U \subseteq V$ then $d_U \geqslant d_V$, i.e., for any $P,Q \in \mathcal{T}^\Downarrow$, $d_U(P,Q) \geq d_V(P,Q)$. In particular, the greatest of all the d_U is $d_\emptyset$; this is the discrete metric on $\mathcal{T}^\Downarrow$. Furthermore, the least of all the d_U is d_Σ; this is the standard metric on $\mathcal{T}^\Downarrow$ as defined in [21, Chap. 8].

Proposition 4.2. *Let $U \subseteq \Sigma$. Then $\mathcal{T}^\Downarrow$ equipped with the metric d_U is a complete ultrametric space and the set of livelock-free processes is a closed subset of $\mathcal{T}^\Downarrow$. Furthermore, if $F : \mathcal{T}^\Downarrow \to \mathcal{T}^\Downarrow$ is contractive with respect to d_U, then F has a unique fixed point given by $\lim_{n\to\infty} F^n(STOP)$. (Note that this fixed point may be divergent.)*

Proof. By Proposition 4.1, $(\mathcal{T}^\Downarrow, d_U)$ is an ultrametric space. The proofs that $(\mathcal{T}^\Downarrow, d_U)$ is a complete metric space and that the set of livelock-free processes is a closed subset of $\mathcal{T}^\Downarrow$ are presented in Appendix B (as Proposition B.2 and Proposition B.3, respectively).

Let $F : \mathcal{T}^\Downarrow \to \mathcal{T}^\Downarrow$ be contractive with respect to d_U. Since $(\mathcal{T}^\Downarrow, d_U)$ is a complete metric space, it follows from Banach's fixed point theorem [26, 21] that F has a unique fixed point given by $\lim_{n\to\infty} F^n(\theta)$, where θ can be any element of $\mathcal{T}^\Downarrow$ and, in particular, the process $STOP$. The unique fixed point may or may not be livelock free, however. $\square$

In the rest of this paper, the only metrics we are concerned with are those associated with some subset of Σ; accordingly, we freely identify metrics and sets when the context is unambiguous.

4.1. **Nonexpansiveness of CSP operators.** Let us fix $U \subseteq \Sigma$. The following lemmas prove that each CSP operator, other than recursion, is at least nonexpansive with respect to d_U in each of its arguments (for some operators we need to impose certain conditions). Proofs can be found in Appendix C.

Lemma 4.3. *For any CSP processes P, P', Q, and Q' the following inequalities hold:*

$$
d_U(P \,\square\, Q, P' \,\square\, Q) \leq d_U(P,P') \text{ and } d_U(P \,\square\, Q, P \,\square\, Q') \leq d_U(Q,Q')
$$

$$
d_U(P \,\sqcap\, Q, P' \,\sqcap\, Q) \leq d_U(P,P') \text{ and } d_U(P \,\sqcap\, Q, P \,\sqcap\, Q') \leq d_U(Q,Q')
$$

$$
d_U(P \,\fatsemi\, Q, P' \,\fatsemi\, Q) \leq d_U(P,P') \text{ and } d_U(P \,\fatsemi\, Q, P \,\fatsemi\, Q') \leq d_U(Q,Q')
$$

$$
d_U(P \,\underset{A}{\|}\, Q, P' \,\underset{A}{\|}\, Q) \leq d_U(P,P') \text{ and } d_U(P \,\underset{A}{\|}\, Q, P \,\underset{A}{\|}\, Q') \leq d_U(Q,Q').
$$

Lemma 4.4. *Let P and Q be CSP processes and let $a \in \Sigma$. Then:*

$$
d_U(a \longrightarrow P, a \longrightarrow Q) \leq d_U(P,Q).
$$

Furthermore, if $a \in U$, then the inequality is strict.

Lemma 4.5. *Let P and Q be CSP processes and let $A \subseteq \Sigma$ satisfy $A \cap U = \emptyset$. Then:*

$$d_U(P \setminus A, Q \setminus A) \leq d_U(P, Q).$$

Lemma 4.6. *Let P and Q be CSP processes, $R \subseteq \Sigma \times \Sigma$ be a renaming relation on Σ and $R(U) = \{y \mid \exists\, x \in U \bullet x\, R\, y\}$. Then:*

$$d_{R(U)}(P[R], Q[R]) \leq d_U(P, Q).$$

Lemma 4.7. *Let P, Q and Q' be CSP processes. Let P always communicate an event from U before it does a $\checkmark$. Then:*

$$d_U(P \,\mathring{,}\, Q, P \,\mathring{,}\, Q') \leq \frac{1}{2} d_U(Q, Q').$$

5. STATIC LIVELOCK ANALYSIS

We present an algorithm based on a static analysis which conservatively flags processes that may livelock. In other words, any process classified as livelock-free really is livelock-free, although the converse may not hold.

Divergent behaviours originate in three different ways, two of which are non-trivial. The first is through direct use of the process DIV; the second comes from unguarded recursions; and the third is through hiding an event, or set of events, which the process can perform infinitely often to the exclusion of all others.

Roscoe [21, Chap. 8] addresses the second and third points by requiring that all recursions be *guarded*, i.e., always perform some event prior to recursing, and by banning use of the hiding operator under recursion. Our idea is to extend Roscoe's requirement that recursions should be guarded by stipulating that one may never hide *all* the guards. In addition, one may not hide a set of events which a process is able to perform infinitely often to the exclusion of all others. This will therefore involve a certain amount of book-keeping.

5.1. **Nonexpansiveness and guardedness.** We first treat the issue of guardedness of the recursions. Our task is complicated by the renaming operator, in that a purported guard may become hidden only after several unwindings of a recursion. The following example illustrates some of the ways in which a recursion may fail to be guarded, and thus diverge.

Example 5.1. Let $\Sigma = \{a, b, a_0, a_1, \ldots, a_n\}$ and let $R = \{(a_i, a_{i+1}) \mid 0 \leqslant i < n\}$ and $S = \{(a, b), (b, a)\}$ be renaming relations on Σ. Consider the following processes.
(1) $\mu X \bullet X$.
(2) $\mu X \bullet a \longrightarrow (X \setminus a)$.
(3) $\mu X \bullet (a \longrightarrow (X \setminus b)) \sqcap (b \longrightarrow (X \setminus a))$.
(4) $\mu X \bullet (a_0 \longrightarrow (X \setminus a_n)) \sqcap (a_0 \longrightarrow X[R])$.
(5) $\mu X \bullet SKIP \sqcap a \longrightarrow (X \,\mathring{,}\, (X[S] \setminus b))$.

The first recursion is trivially unguarded. In the second recursion the guard a is hidden after the first recursive call. In the third process the guard in each summand is hidden in the other summand; this process will also diverge once it has performed a single event. In the fourth example we cannot choose a set of guards which is both stable under the renaming operator and does not contain a_n. This process, call it P, makes the following sequence of visible transitions:

$$P \xrightarrow{a_0} P \setminus a_n \xrightarrow{a_0} P[R] \setminus a_n \xrightarrow{a_1} P[R][R] \setminus a_n \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} P[R][R] \ldots [R] \setminus a_n.$$

But the last process diverges, since P can make an infinite sequence of a_0-transitions which get renamed to a_n by successive applications of R and are then hidden at the outermost level.

A cursory glance at the last process might suggest that it is guarded in $\{a\}$. However, similarly to the previous example, hiding and renaming conspire to produce divergent behaviour. In fact the process, call it P, can make an a-transition to $P \,\mathbin{\raise.3ex\hbox{$\scriptstyle;$}}\, (P[S] \setminus b)$, and thence to $(P[S] \setminus b)[S] \setminus b$ via two τ-transitions. But this last process can diverge. □

The intuitions underlying our definitions of nonexpansiveness and guardedness are as follows. Let $U \subseteq \Sigma$ be fixed, giving rise to a metric d_U on $\mathcal{T}^{\Downarrow}$, and let $P = P(X)$ be a CSP term with a single free variable X. Then P—viewed as a selfmap on $\mathcal{T}^{\Downarrow}$—is by definition contractive with respect to d_U (with contraction factor $1/2$) provided that, for every $T_1, T_2 \in \mathcal{T}^{\Downarrow}$, it is the case that

$$d_U(P(T_1), P(T_2)) \le \frac{1}{2} d_U(T_1, T_2) \,. \tag{5.1}$$

Now if P happens to apply a one-to-one renaming operator R to its argument, say, then it becomes necessary to rephrase Equation 5.1 above as requiring that

$$d_V(P(T_1), P(T_2)) \le \frac{1}{2} d_U(T_1, T_2) \,, \tag{5.2}$$

where d_V is a new metric such that $R(U) = V$. Indeed, since P renames events in U to ones in V, the distance between $P(T_1)$ and $P(T_2)$ must be measured with respect to the renamed events, rather than the original ones.

This leads us to the concept of a function that is contractive *with respect to two different metrics d_U and d_V*, in which the first metric is used to measure the distance between two inputs, whereas the second metric measures the distance between the corresponding two outputs of the function under consideration—see Figure 5. Following our convention of identifying sets and metrics, we would say that P is *contractive in the pair* (U, V).

This reasoning needs to be slightly refined in order to handle non-injective renamings as well as hiding. Our goal is then to define, by induction on the structure of CSP terms, a function $\mathsf{C}_X : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$, which associates to each CSP term $P(X)$ a set of pairs of metrics (U, V) such that Equation 5.2 holds. Of course, such a definition would also need to handle terms with several free variables (in addition to X), which can be done using a standard projection.

It turns out that, in order to define such a function C_X, it is first necessary to compute a function $\mathsf{N}_X : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ which calculates, for every CSP term $P(X)$, a set of pairs of metrics (U, V) such that P is nonexpansive in (U, V), following the same convention of measuring the distance between inputs via the metric d_U and the distance between outputs via the metric d_V.

It is also necessary to calculate an auxiliary function $\mathsf{G} : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma))$, which itself depends on a certain function $\mathsf{F} : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$. This may seem problematic, since (as we shall see) F itself depends on C_X, but this mutual recursion is well-defined because uses of F in the definition of G only occur on subterms, and likewise for uses of G in C_X and uses of C_X in F.

We provide the intuitions underlying the definitions of G and F later on, as these functions are introduced. For now let us finally remark that all the functions that we define are *conservative underapproximations*, i.e., sound, but not necessarily complete. For
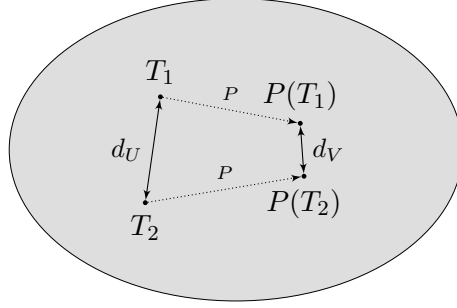
Figure 5: P is contractive in (U,V), i.e., when the distance between inputs is measured with d_U and the distance between outputs is measured with d_V.

example, $N_X(P)$ as defined below generates some but not necessarily all of the pairs of metrics that witness the nonexpansiveness of P.

Intuitively, the role of $N_X(P)$ is to keep track of all hiding and renaming in P. The key property of the function N_X is given by the following proposition.

Proposition 5.2. *Let $P(X, Y_1, \ldots, Y_n) = P(X, \overline{Y})$ be a CSP term whose free variables are contained within the set $\{X, Y_1, \ldots, Y_n\}$. Let $N_X : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ be defined recursively on the structure of P as shown in Figure 6. If $(U, V) \in N_X(P)$, then for all $T_1, T_2, \Theta_1, \ldots, \Theta_n \in \mathcal{T}^{\Downarrow}$, we have $d_V(P(T_1, \overline{\Theta}), P(T_2, \overline{\Theta})) \le d_U(T_1, T_2)$.*

$$N_X(P) \mathrel{\widehat{=}} \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma) \;\; \textbf{whenever } X \textbf{ is not free in } P\textbf{; otherwise:}$$

$$N_X(a \longrightarrow P) \mathrel{\widehat{=}} N_X(P)$$
$$N_X(P_1 \oplus P_2) \mathrel{\widehat{=}} N_X(P_1) \cap N_X(P_2) \;\; \text{if } \oplus \in \{\sqcap, \Box, \mathbin{\raisebox{0.2ex}{\scriptsize$\circ$}}_{9}, \underset{A}{\|}\}$$
$$N_X(P \setminus A) \mathrel{\widehat{=}} \{(U, V) \mid (U, V') \in N_X(P) \wedge V' \cap A = \emptyset \wedge V' \subseteq V\}$$
$$N_X(P[R]) \mathrel{\widehat{=}} \{(U, V) \mid (U, V') \in N_X(P) \wedge R(V') \subseteq V\}$$
$$N_X(X) \mathrel{\widehat{=}} \{(U, V) \mid U \subseteq V\}$$
$$N_X(\mu Y \boldsymbol{.} P) \mathrel{\widehat{=}} \{(U, V) \mid (U', V') \in N_X(P) \wedge (V', V') \in N_Y(P) \wedge U \subseteq U' \wedge V' \subseteq V\}$$
$$\text{if } Y \neq X \ .$$

Figure 6: Nonexpansive sets.

Proof. The proof proceeds by structural induction on P and is presented in Appendix D. $\qquad\square$

Note that, by construction, $N_X(P)$ is always downwards-closed in its first component and upwards-closed in its second component, which is sound due to antitoneness (if $U \subseteq U'$ then $d_U \geqslant d_{U'}$). Some of the rules are plainly straightforward, whereas others (hiding and especially recursion) require careful thought. An intuition for correctness is probably best obtained by an examination of the proof.

We now move to the function G. Intuitively, $\mathsf{G}(P) \subseteq \mathcal{P}(\Sigma)$ lists the 'guards' of $\checkmark$ for P. Formally:

Proposition 5.3. *Let $P(X, Y_1, \ldots, Y_n) = P(X, \overline{Y})$ be a term whose free variables are contained within the set $\{X, Y_1, \ldots, Y_n\}$. If $V \in \mathsf{G}(P)$, then, with any processes—and in particular DIV —substituted for the free variables of P, P must communicate an event from V before it can do a $\checkmark$.* □

The inductive clauses for G are given in Figure 7. As mentioned earlier, note that these make use of the collection of *fair sets* $\mathsf{F}(P_i)$ of P_i, which is presented later on in Section 5.2. The definition is nonetheless well-founded since F is here only applied to subterms. The salient property of $\mathsf{F}(P_i) \neq \emptyset$ is that the process P_i is guaranteed to be livelock-free. The proof for Proposition 5.3 proceeds by structural induction on P and is presented together with Proposition D.1 in Appendix D.

$$\mathsf{G}(STOP) \mathrel{\widehat{=}} \mathcal{P}(\Sigma)$$
$$\mathsf{G}(a \longrightarrow P) \mathrel{\widehat{=}} \mathsf{G}(P) \cup \{V \mid a \in V\}$$
$$\mathsf{G}(SKIP) \mathrel{\widehat{=}} \emptyset$$
$$\mathsf{G}(P_1 \oplus P_2) \mathrel{\widehat{=}} \mathsf{G}(P_1) \cap \mathsf{G}(P_2) \quad \text{if } \oplus \in \{\Box, \sqcap\}$$
$$\mathsf{G}(P_1 \,\fatsemi\, P_2) \mathrel{\widehat{=}} \begin{cases} \mathsf{G}(P_1) \cup \mathsf{G}(P_2) & \text{if } P_1 \text{ is closed and } \mathsf{F}(P_1) \neq \emptyset \\ \mathsf{G}(P_1) & \text{otherwise} \end{cases}$$
$$\mathsf{G}(P_1 \underset{A}{\parallel} P_2) \mathrel{\widehat{=}} \begin{cases} \mathsf{G}(P_1) \cup \mathsf{G}(P_2) & \text{if, for } i = 1, 2, P_i \text{ is closed and } \mathsf{F}(P_i) \neq \emptyset \\ \mathsf{G}(P_1) \cap \mathsf{G}(P_2) & \text{otherwise} \end{cases}$$
$$\mathsf{G}(P \setminus A) \mathrel{\widehat{=}} \begin{cases} \{V \mid V' \in \mathsf{G}(P) \wedge V' \cap A = \emptyset \wedge V' \subseteq V\} & \text{if } P \text{ is closed and} \\ & (\emptyset, \Sigma - A) \in \mathsf{F}(P) \\ \emptyset & \text{otherwise} \end{cases}$$
$$\mathsf{G}(P[R]) \mathrel{\widehat{=}} \{V \mid V' \in \mathsf{G}(P) \wedge R(V') \subseteq V\}$$
$$\mathsf{G}(X) \mathrel{\widehat{=}} \emptyset$$
$$\mathsf{G}(\mu X \boldsymbol{.} P) \mathrel{\widehat{=}} \mathsf{G}(P) \ .$$

Figure 7: Guard sets.

We are now ready to define $\mathsf{C}_X(P) \subseteq \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$, whose central property is given by the following proposition.

Proposition 5.4. *Let $P(X, Y_1, \ldots, Y_n) = P(X, \overline{Y})$ be a term whose free variables are contained within the set $\{X, Y_1, \ldots, Y_n\}$. Let $\mathsf{C}_X : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ be defined recursively on the structure of P as shown in Figure 8. If $(U, V) \in \mathsf{C}_X(P)$, then for all $T_1, T_2, \theta_1, \ldots, \theta_n \in \mathcal{T}^{\Downarrow}$, we have $d_V(P(T_1, \overline{\theta}), P(T_2, \overline{\theta})) \leq \frac{1}{2} d_U(T_1, T_2)$.*
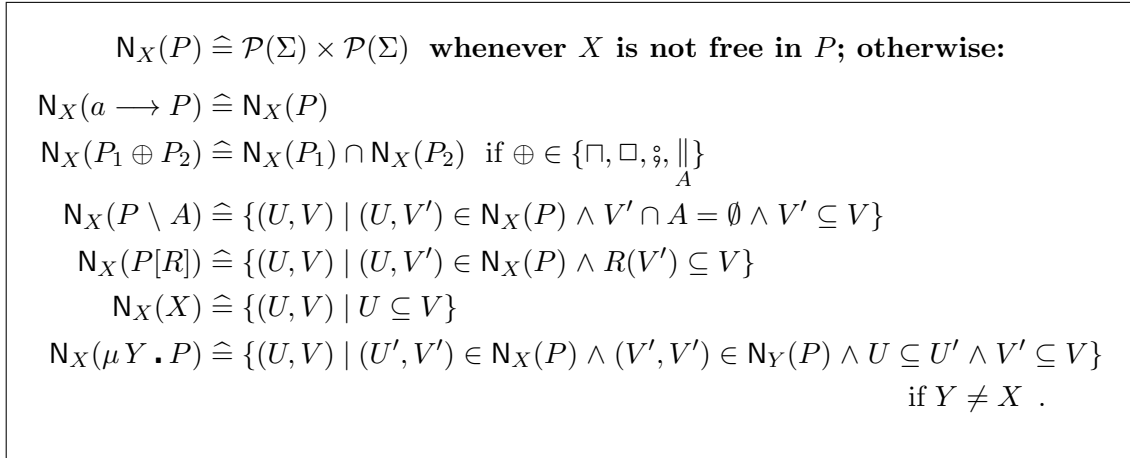
Proof. The proof proceeds by structural induction on P and is presented together with Proposition D.1 in Appendix D. □

$$\mathsf{C}_X(P) \mathrel{\widehat{=}} \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma) \ \ \textbf{whenever } X \textbf{ is not free in } P\textbf{; otherwise:}$$

$$\mathsf{C}_X(a \longrightarrow P) \mathrel{\widehat{=}} \mathsf{C}_X(P) \cup \{(U,V) \in \mathsf{N}_X(P) \mid a \in V\}$$

$$\mathsf{C}_X(P_1 \oplus P_2) \mathrel{\widehat{=}} \mathsf{C}_X(P_1) \cap \mathsf{C}_X(P_2) \ \ \text{if } \oplus \in \{\Box, \sqcap, \parallel_A\}$$

$$\mathsf{C}_X(P_1 \,\mathring{,}\, P_2) \mathrel{\widehat{=}} \mathsf{C}_X(P_1) \cap (\mathsf{C}_X(P_2) \cup \{(U,V) \in \mathsf{N}_X(P_2) \mid V \in \mathsf{G}(P_1)\})$$

$$\mathsf{C}_X(P \setminus A) \mathrel{\widehat{=}} \{(U,V) \mid (U,V') \in \mathsf{C}_X(P) \wedge V' \cap A = \emptyset \wedge V' \subseteq V\}$$

$$\mathsf{C}_X(P[R]) \mathrel{\widehat{=}} \{(U,V) \mid (U,V') \in \mathsf{C}_X(P) \wedge R(V') \subseteq V\}$$

$$\mathsf{C}_X(X) \mathrel{\widehat{=}} \emptyset$$

$$\mathsf{C}_X(\mu Y \boldsymbol{.} P) \mathrel{\widehat{=}} \{(U,V) \mid (U',V') \in \mathsf{C}_X(P) \wedge (V',V') \in \mathsf{N}_Y(P) \wedge U \subseteq U' \wedge V' \subseteq V\}$$
$$\text{if } Y \neq X \ .$$

Figure 8: Contractive sets.

Note that contraction guarantees a unique fixed point, albeit not necessarily a livelock-free one. For instance, $P(X) = (a \longrightarrow X \setminus b) \Box (\mu Y \boldsymbol{.} b \longrightarrow Y)$ has a unique fixed point which can diverge after a single event.

5.2. **Fair sets and hiding.** In order to prevent livelock, we must ensure that, whenever a process can perform an infinite[12] unbroken sequence of events from a particular set A, then we never hide the whole of A. To this end, we now associate to each CSP term P a collection of (pairs of) *fair sets* $\mathsf{F}(P) \subseteq \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$: intuitively, this allows us to keep track of the events which the process is guaranteed to perform infinitely often in any infinite execution of P. As with nonexpansiveness and contractiveness, the potential presence of renaming and hiding requires us separately to keep track of events performed by the input processes and the output (or compound) process.

Given a set $W \subseteq \Sigma$, we say that a process is W-fair if any of its infinite traces contains infinitely many events from W. We now have:

Proposition 5.5. *Let* $P(X_1, \ldots, X_n) = P(\overline{X})$ *be a CSP term whose free variables are contained within the set* $\{X_1, \ldots, X_n\}$. *Let* $\mathsf{F} : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ *be defined recursively on the structure of* P *as shown in Figure 9. If* $(U,V) \in \mathsf{F}(P)$, *then, for any collection of livelock-free, U-fair processes* $\theta_1, \ldots, \theta_n \in \mathcal{T}^{\Downarrow}$, *the process* $P(\theta_1, \ldots, \theta_n)$ *is livelock-free and V-fair.*

Proof. The proof proceeds by structural induction on P and is presented together with Proposition D.1 in Appendix D. $\qquad\Box$

---

[12]Recall our understanding that a process can 'perform' an infinite trace iff it can perform all its finite prefixes.

$$
\begin{aligned}
\mathsf{F}(STOP) &\mathrel{\hat{=}} \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma) \\
\mathsf{F}(a \longrightarrow P) &\mathrel{\hat{=}} \mathsf{F}(P) \\
\mathsf{F}(SKIP) &\mathrel{\hat{=}} \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma) \\
\mathsf{F}(P_1 \oplus P_2) &\mathrel{\hat{=}} \mathsf{F}(P_1) \cap \mathsf{F}(P_2) \quad \text{if } \oplus \in \{\sqcap, \square, \mathbin{\raisebox{0.5ex}{\scriptsize\ensuremath{\,\circ\atop\circ\,}}}\} \\
\mathsf{F}(P_1 \parallel_A P_2) &\mathrel{\hat{=}} (\mathsf{F}(P_1) \cap \mathsf{F}(P_2)) \cup \\
&\qquad \{(U_1 \cap U_2, V_1) \mid (U_1, V_1) \in \mathsf{F}(P_1) \wedge (U_2, A) \in \mathsf{F}(P_2)\} \cup \\
&\qquad \{(U_1 \cap U_2, V_2) \mid (U_2, V_2) \in \mathsf{F}(P_2) \wedge (U_1, A) \in \mathsf{F}(P_1)\} \\
\mathsf{F}(P \setminus A) &\mathrel{\hat{=}} \{(U, V) \mid (U, V') \in \mathsf{F}(P) \wedge V' \cap A = \emptyset \wedge V' \subseteq V\} \\
\mathsf{F}(P[R]) &\mathrel{\hat{=}} \{(U, V) \mid (U, V') \in \mathsf{F}(P) \wedge R(V') \subseteq V\} \\
\mathsf{F}(X) &\mathrel{\hat{=}} \{(U, V) \mid U \subseteq V\} \\
\mathsf{F}(\mu X \boldsymbol{.} P) &\mathrel{\hat{=}} \begin{cases} \{(U, V) \mid (W, W) \in \mathsf{C}_X(P) \cap \mathsf{F}(P) \wedge U \subseteq W \subseteq V\} & \text{if } \mu X \boldsymbol{.} P \text{ is open} \\ \mathcal{P}(\Sigma) \times \{V \mid (W, W) \in \mathsf{C}_X(P) \cap \mathsf{F}(P) \wedge W \subseteq V\} & \text{otherwise .} \end{cases}
\end{aligned}
$$

Figure 9: Fair sets.

Note that, by construction, $\mathsf{F}(P)$ is always downwards-closed in its first component and upwards-closed in its second component; this is sound since if $U \subseteq U'$ and P is U-fair, then P is automatically U'-fair as well.

We now obtain one of our main results as an immediate corollary:

Theorem 5.6. *Let P be a CSP process (i.e., closed term) not containing DIV in its syntax. If $\mathsf{F}(P) \neq \emptyset$, then P is livelock-free.*

Proof. Let $\mathsf{F}(P) \neq \emptyset$ and $(U, V) \in \mathsf{F}(P)$ for some $U, V \subseteq \Sigma$. Since P is closed, P has no free variables. Then, by Proposition 5.5, P is livelock-free (and V-fair). $\qquad\square$

Theorem 5.6 gives rise to a procedure for establishing livelock-freedom of a given process P over alphabet Σ, whose complexity is at most quadratic in the syntactic size of P and exponential in the cardinality of Σ: indeed, for fixed Σ, one computes $\mathsf{N}_X(Q)$, $\mathsf{G}(Q)$, $\mathsf{C}_X(Q)$, and $F(Q)$ for every variable X appearing in P and every subterm Q of P. Since the number of variables and the number of subterms are both at most linear in the size of P, the computation is at most quadratic in P. On the other hand, each of $\mathsf{N}_X(Q)$, $\mathsf{C}_X(Q)$, and $F(Q)$ is a collection of pairs of subsets of Σ, whereas $\mathsf{G}(Q)$ is a collection of subsets of Σ. Thus for Σ not fixed, these pieces of data are potentially exponentially large.

In practice, applications often make use of moderately large alphabets, making the direct set-based approach described above prohibitively expensive. However, an inspection of the rules defining $\mathsf{N}_X(Q)$, $\mathsf{G}(Q)$, $\mathsf{C}_X(Q)$, and $F(Q)$ reveals that these objects can be represented *symbolically*, either as propositional formulas or as BDDs—further implementation details are provided in Section 7. As a result, the problem of deciding whether $\mathsf{F}(P) \neq \emptyset$ can be seen to lie in NP.

6. Structurally Finite-State Processes

The techniques developed in Section 5 allow us to handle the widest range of CSP processes; among others, they enable one to establish livelock-freedom of numerous infinite-state processes including examples making use of infinite buffers or unbounded counters. Such processes are of course beyond the reach of explicit-state model checkers such as FDR. In order to create them in CSP, it is necessary to use devices such as recursing under the parallel operator. In practice, however, the vast majority of processes tend to be finite state.

Let us therefore define a CSP process to be *structurally finite state* if it never syntactically recurses under any of parallel, the left-hand side of a sequential composition, hiding, or renaming.

More precisely, we first define a notion of *sequential* CSP terms: $STOP$, $SKIP$, and X are sequential; if P and Q are sequential, then so are $a \longrightarrow P$, $P \sqcap Q$, $P \square Q$, and $\mu X . P$; and if in addition P is closed, then $P \,\mathbin{;}\, Q$, $P \setminus A$, and $P[R]$ are sequential. Observe that sequential processes give rise to labelled transition systems of size linear in the length of their syntax.

Now any closed sequential term is deemed to be structurally finite state; and if P and Q are structurally finite state, then so are $a \longrightarrow P$, $P \sqcap Q$, $P \square Q$, $P \parallel_{A} Q$, $P \,\mathbin{;}\, Q$, $P \setminus A$, and $P[R]$. Note that structurally finite-state CSP terms are always closed, i.e., are processes. Let us write $\overline{\mathsf{SFS}}$ to denote the collection of all structurally finite-state processes.

Whether a given process is structurally finite state can easily be established by syntactic inspection, for example by using Bekič's theorem [28] (see Section 2) and analysing the resulting μ expression. For such processes, it turns out that we can substantially both simplify and sharpen our livelock analysis. More precisely, the computation of nonexpansive and contractive data is circumvented by instead directly examining closed sequential components in isolation. Furthermore, the absence of free variables in compound processes makes some of the earlier fairness calculations unnecessary, thereby allowing more elaborate and finer data to be computed efficiently, as we now explain.

Let u be an infinite trace over Σ, and let $F, C \subseteq \Sigma$ be two sets of events. We say that u is *fair in F* if, for each $a \in F$, u contains infinitely many occurrences of a,[13] and we say that u is *co-fair in C* if, for each $b \in C$, u contains at most finitely many occurrences of b. We lift this to sets of traces in the following way: let $T \subseteq \Sigma^{\omega}$ be a set of infinite traces over Σ, and let $\mathcal{F} = \{(F_1, C_1), \ldots, (F_k, C_k)\} \subseteq \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$ be a collection of pairs of subsets of Σ. We say that T is *fair/co-fair in $\mathcal{F}$* provided that, for every infinite trace $u \in T$, there exists a pair $(F_i, C_i) \in \mathcal{F}$ such that u is both fair in F_i and co-fair in C_i.

Our aim is the following. Given a structurally finite-state process P, we wish to compute:
- a Boolean-valued *livelock flag* $\delta(P) \in \{\text{true}, \text{false}\}$, together with
- a collection of pairs of disjoint sets $\Phi(P) = \{(F_1, C_1), \ldots, (F_k, C_k)\} \subseteq \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$,

such that if $\delta(P) = \text{false}$, then (i) P is livelock-free, and moreover (ii) $\mathsf{traces}^{\omega}(P)$ is fair/co-fair in $\Phi(P)$.

6.1. **Handling Sequential $\overline{\mathsf{SFS}}$ Processes.** For P a sequential $\overline{\mathsf{SFS}}$ process, let us denote by M_P its associated labelled transition system as derived from the operational semantics; let us assume that we construct M_P so that all states are reachable from the initial state. As

---
[13]Note that this notion of 'fairness' differs from that used in the previous section.

noted earlier, M_P has size linear in the syntactic description of P. We can then compute the livelock flag $\delta(P)$ and the set of fair/co-fair pairs $\Phi(P) = \{(F_1, C_1), \ldots, (F_k, C_k)\}$ exactly, directly from M_P. More precisely, we set $\delta(P)$ to true or false depending on whether or not P can eventually diverge, i.e., whether M_P contains a τ-cycle. This can be carried out using Tarjan's algorithm in time linear in the number of states in M_P.

Assuming the livelock flag $\delta(P)$ is false, we compute the set of fair/co-fair pairs $\Phi(P)$ as follows. We add a pair of disjoint sets of events (F, C) to $\Phi(P)$ if and only if M_P comprises some infinite trace which is fair in F and co-fair in C. Note that if P has no infinite trace, $\Phi(P)$ will therefore be empty.

It is worth pointing out how the computation of $\Phi(P)$ can be achieved efficiently. Given a non-empty set $L \subseteq \Sigma$ of events, we delete all $(\Sigma - L)$-labelled transitions from M_P. If the resulting graph contains a (not necessarily reachable) strongly connected component which comprises every event in L, we include $(L, \Sigma - L)$ as a fair/co-fair pair for P, and otherwise we do not.

Of course, in actual implementations, it is likely not desirable to iterate explicitly over all possible subsets of Σ. The computation we described can be carried out symbolically using a Boolean circuit of size polynomial in P, using well-known circuit algorithms for computing the transitive closure of relations. Consequently, $\Phi(P)$ can be represented symbolically and compactly either as a BDD or as a propositional formula. Further implementation details are provided in Section 7 and Appendix G.

6.2. Compositional Rules for $\overline{\mathsf{SFS}}$ Processes.

Theorem 6.1. *Let P be a structurally finite-state process. Let $\Phi : \overline{\mathsf{SFS}} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ and $\delta : \overline{\mathsf{SFS}} \longrightarrow \{\text{true}, \text{false}\}$ be defined recursively on the structure of P as shown in Figures 10 and 11, respectively. Then, if $\delta(P) = \text{false}$, P is livelock-free. Moreover, if in addition $\Phi(P) = \{(F_1, C_1), \ldots, (F_k, C_k)\}$, then, for every infinite trace u of P, there exists $1 \leq i \leq k$, such that u is both fair in F_i and co-fair in C_i.*

The proof of Theorem 6.1 proceeds by structural induction on P and is presented in Appendix E.

Note that by construction, all fair/co-fair pairs of sets thus generated remain disjoint; this is key in the rule for parallel composition, where the fair/co-fair data of individual subcomponents enables one to rule out certain pairs for the resulting parallel process. Also, as shown in the proof of Theorem 6.1, whenever (F, P) appears as a fair/co-fair pair in some $\Phi(P)$, F is never empty.

Let us also remark that the δ clause for the hiding operator is here phrased in a way so as to make the rule as intuitively clear as possible. In practice, one however need not iterate over all possible pairs $(F, C) \in \Phi(P)$: it is simpler instead to evaluate the negation, an existential calculation which is easily integrated within either a SAT or BDD implementation.

6.3. Static Livelock Analysis Algorithm.
Theorems 5.6 and 6.1 yield a conservative algorithm for livelock-freedom: given a CSP process P (which we will assume does not contain DIV in its syntax), determine first whether P is structurally finite state. If so, assert that P is livelock-free if $\delta(P) = \text{false}$, and otherwise report an inconclusive result. If P is not structurally finite state, assert that P is livelock-free if $\mathsf{F}(P) \neq \emptyset$, and otherwise report an inconclusive result.

$$\Phi(P) \mathrel{\widehat{=}} \text{computed from } P\text{'s LTS (see Section 6.1)}$$

$$\textbf{whenever } P \textbf{ is a sequential } \overline{\mathsf{SFS}} \textbf{ process; otherwise:}$$

$$\Phi(a \longrightarrow P) \mathrel{\widehat{=}} \Phi(P)$$
$$\Phi(P_1 \oplus P_2) \mathrel{\widehat{=}} \Phi(P_1) \cup \Phi(P_2) \;\text{ if } \oplus \in \{\sqcap, \square, \mathbin{;}\}$$
$$\Phi(P_1 \parallel_A P_2) \mathrel{\widehat{=}} \{(F,C) \mid F \cap C = \emptyset \wedge (F_i, C_i) \in \Phi(P_i) \text{ for } i = 1,2 \wedge$$
$$F = F_1 \cup F_2 \wedge$$
$$C = (C_1 \cap A) \cup (C_2 \cap A) \cup ((C_1 - A) \cap (C_2 - A))\} \cup$$
$$\{(F,C) \mid (F,C) \in \Phi(P_1) \wedge F \cap A = \emptyset\} \cup$$
$$\{(F,C) \mid (F,C) \in \Phi(P_2) \wedge F \cap A = \emptyset\}$$
$$\Phi(P \setminus A) \mathrel{\widehat{=}} \{(F - A, C \cup A) \mid (F,C) \in \Phi(P)\}$$
$$\Phi(P[R]) \mathrel{\widehat{=}} \{(F,C) \mid (F',C') \in \Phi(P) \wedge F' \subseteq R^{-1}(F) \wedge F \subseteq R(F') \wedge$$
$$C = \{b \in \Sigma \mid R^{-1}(b) \subseteq C'\}\} \;.$$

Figure 10: Fair/co-fair sets.

$$\delta(P) \mathrel{\widehat{=}} \text{computed from } P\text{'s LTS (see Section 6.1)}$$

$$\textbf{whenever } P \textbf{ is a sequential } \overline{\mathsf{SFS}} \textbf{ process; otherwise:}$$

$$\delta(a \longrightarrow P) \mathrel{\widehat{=}} \delta(P)$$
$$\delta(P_1 \oplus P_2) \mathrel{\widehat{=}} \delta(P_1) \vee \delta(P_2) \;\text{ if } \oplus \in \{\sqcap, \square, \parallel_A, \mathbin{;}\}$$
$$\delta(P \setminus A) \mathrel{\widehat{=}} \begin{cases} \text{false} & \text{if } \delta(P) = \text{false and, for each } (F,C) \in \Phi(P),\, F - A \neq \emptyset \\ \text{true} & \text{otherwise} \end{cases}$$
$$\delta(P[R]) \mathrel{\widehat{=}} \delta(P) \;.$$

Figure 11: δ-bit.

The complexity of this procedure is in the worst case quadratic in the syntactic size of P and exponential in the cardinality of Σ, by invoking a similar line of reasoning as that presented following Theorem 5.6. Likewise, determining for an $\overline{\mathsf{SFS}}$ process P whether $\delta(P)$ is true is easily seen to lie in NP.

It is perhaps useful to illustrate how the inherent incompleteness of our procedure can manifest itself in very simple ways. For example, let $P = a \longrightarrow Q$ and $Q = (a \longrightarrow P) \,\square\, (b \longrightarrow Q)$, and let $R = (P \parallel_{\{a,b\}} Q) \setminus b$. Using Bekič's procedure, R is readily seen to be (equivalent to) a structurally finite-state process. Moreover, R is clearly livelock-free, yet $\delta(R) = \text{true}$ and $\mathsf{F}(R) = \emptyset$. Intuitively, establishing livelock-freedom here requires some form of state-space exploration, to see that the 'divergent' state $(Q \parallel_{\{a,b\}} Q) \setminus b$ of R is in fact

unreachable, but that is precisely the sort of reasoning that our static analysis algorithm is not geared to do.

Nonetheless, we have found in practice that our approach succeeded in establishing livelock-freedom for a wide range of existing benchmarks; we report on some of our experiments in Section 7, and also present in Appendix F a small case study illustrating the intuitions underlying the rules given in Figures 10 and 11.

We conclude by noting that, for structurally finite-state processes, Theorem 6.1 is stronger than Theorem 5.6, i.e., it correctly classifies a larger class of processes as being livelock-free, as stated in the following proposition. Empirically, algorithms based on Theorem 6.1 have also been found to run considerably faster in practice.

Proposition 6.2. *For any structurally finite-state process P, if $\mathsf{F}(P) \neq \emptyset$ then $\delta(P) = $ false.*

Proof. A proof sketch is given in Appendix E. $\square$

7. Implementation and Experimental Results

We have implemented both the general framework and the framework for structurally finite-state processes in a tool called SLAP, which is an acronym for Static Livelock Analyser of Processes. Computationally, the crux of our algorithms revolves around the generation and manipulation of sets. The algorithms fit very naturally into a symbolic paradigm; hence SLAP is fully symbolic. The choice of an underlying symbolic engine is configurable, with support for using a SAT engine (based on MiniSAT 2.0), a BDD engine (based on CUDD 2.4.2), or running a SAT and a BDD analyser in parallel and reporting the results of the first one to finish. Some details regarding the symbolic part of our frameworks and algorithms are presented in Appendix G.

We have also integrated the framework for analysing structurally finite-state processes directly into FDR [1], where it now constitutes an alternative back-end for establishing livelock freedom. The binaries for the latter can be downloaded from the following location:

http://www.cs.ox.ac.uk/projects/concurrency-tools/slap/

We experimented with a wide range of benchmarks, including parameterised, parallelised, and piped versions of Milner's Scheduler, the Alternating Bit Protocol, the Sliding Window Protocol, the Dining Philosophers, Yantchev's Mad Postman Algorithm [29], as well as a Distributed Database algorithm.[14] In all our examples, internal communications were hidden, so that livelock-freedom can be viewed as a progress or liveness property. All benchmarks were livelock-free, although the reader familiar with the above examples will be aware that manually establishing livelock-freedom for several of these can be a subtle exercise.

In all cases apart from the Distributed Database algorithm, SLAP was indeed correctly able to assert livelock-freedom (save for rare instances of timing out). (Livelock-freedom for the Distributed Database algorithm turns out to be remarkably complex; see [21] for details.) In almost all instances, both BDD-based and SAT-based implementations of SLAP substantially outperformed the state-of-the-art CSP model checker FDR, often completing orders of magnitude faster. On the whole, BDD-based and SAT-based implementations performed comparably, with occasional discrepancies. All experiments were carried out

---

[14]Scripts and descriptions for all benchmarks are available from the website associated with [22].

on a 3.07GHz Intel Xeon processor running under Ubuntu with 8 GB of RAM. Times in seconds are given in Table 1, with * indicating a 30-minute timeout.

Benchmark	FDR	Static (BDD)	Static (SAT)
Milner-10	0	0.06	0.05
Milner-15	0	0.19	0.14
Milner-20	409	0.63	0.28
Milner-21	948	0.73	0.23
Milner-22	*	0.93	0.25
Milner-25	*	1.63	0.41
Milner-30	*	7.56	0.8
ABP-0	0	0.03	0.11
ABP-0-inter-2	0	0.03	0.23
ABP-0-inter-3	23	0.06	0.35
ABP-0-inter-4	*	0.08	0.47
ABP-0-inter-5	*	0.09	0.63
ABP-0-pipe-2	0	0.04	0.35
ABP-0-pipe-3	2	0.06	0.75
ABP-0-pipe-4	175	0.08	1.27
ABP-0-pipe-5	*	0.10	1.85
ABP-0-pipe-6	*	0.11	2.91
ABP-4	0	0.11	*
ABP-4-inter-2	39	0.16	*
ABP-4-inter-3	*	0.22	*
ABP-4-inter-7	*	0.39	*
ABP-4-pipe-2	12	0.38	*
ABP-4-pipe-3	*	0.38	*
ABP-4-pipe-7	*	0.39	*

Benchmark	FDR	Static (BDD)	Static (SAT)
SWP-1	0	0.03	7.06
SWP-2	0	0.46	*
SWP-3	0	46.81	*
SWP-1-inter-2	0	0.04	14.84
SWP-1-inter-3	31	0.06	24.02
SWP-1-inter-4	*	0.08	29.44
SWP-1-inter-7	*	0.13	58.82
SWP-2-inter-2	170	0.71	*
SWP-2-inter-3	*	0.94	*
SWP-1-pipe-2	0	0.04	28.09
SWP-1-pipe-3	0	0.07	66.71
SWP-1-pipe-4	3	0.09	121.09
SWP-1-pipe-5	246	0.10	192.39
SWP-1-pipe-7	*	0.14	399.55
Philosophers-5	0	0.30	0.10
Philosophers-7	2	1.62	0.21
Philosophers-8	20	2.51	0.35
Philosophers-9	140	3.98	0.50
Philosophers-10	960	7.49	0.72
Mad Postman-2	0	0.06	0.03
Mad Postman-3	6	*	0.20
Mad Postman-4	*	*	0.89
Mad Postman-5	*	*	4.21
Mad Postman-6	*	*	20.75

Table 1: Times reported are in seconds, with * denoting a 30-minute timeout.

8. Future Work

An interesting property of our approach is the possibility for our algorithm to produce a *certificate* of livelock-freedom, consisting among others in the various sets supporting the final judgement. Such a certificate could then be checked by an independent tool.

Other directions for future work include improving the efficiency of SLAP by incorporating various abstractions (such as collapsing all events on a given channel, or placing *a priori* bounds on the size of sets), or conversely increasing accuracy at modest computational cost, for example by making use of algebraic laws at the syntactic level, such as bounded unfoldings of parallel compositions.

References

[1] P. Armstrong, M. Goldsmith, G. Lowe, J. Ouaknine, H. Palikareva, A. W. Roscoe, and J. Worrell. Recent developments in FDR. In *Proceedings of CAV*, volume 7358, pages 699–704. Springer LNCS, 2012.

[2] J. W. De Bakker and J. I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54:70–120, 1982.

[3] M. Bravetti and R. Gorrieri. Deciding and axiomatizing weak ST bisimulation for a process algebra with recursion and action refinement. *ACM Transactions on Computational Logic*, 3(4):465–520, 2002.

[4] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[6] A. Dimovski. A compositional method for deciding program termination. In *ICT Innovations*, volume 83, pages 71–80. Springer CCIS, 2010.

[7] R. O. Gandy. An early proof of normalization by A.M. Turing. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, volume 267, pages 453–455. Academic Press, 1980.

[8] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Science 7. Cambridge University Press, 1988.

[9] A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In *Proceedings of TACAS*, volume 3920, pages 441–444. Springer LNCS, 2006.

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

[11] R. H. Katz and G. Borriello. *Contemporary Logic Design, Second Edition*. Prentice Hall, 2005.

[12] M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Proceedings of CAV*, volume 6806, pages 585–591. Springer LNCS, 2011.

[13] S. Leue, A. Ştefănescu, and W. Wei. A livelock freedom analysis for infinite state asynchronous reactive systems. In *Proceedings of CONCUR*, volume 4137, pages 79–94. Springer LNCS, 2006.

[14] S. Leue, A. Ştefănescu, and W. Wei. Dependency analysis for control flow cycles in reactive communicating processes. In *Proceedings of SPIN*, volume 5156, pages 176–195. Springer LNCS, 2008.

[15] R. Milner. *Communication and Concurrency*. Prentice-Hall International, London, 1989.

[16] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, Cambridge, MA, 1996.

[17] J. Ouaknine, H. Palikareva, A. W. Roscoe, and J. Worrell. Static livelock analysis in CSP. In *Proceedings of CONCUR*, volume 6901, pages 389–403. Springer LNCS, 2011.

[18] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 2004.

[19] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[20] A. W. Roscoe. *A Mathematical Theory of Communicating Processes*. PhD thesis, Oxford University, 1982.

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

[22] A. W. Roscoe. *Understanding Concurrent Systems*. Springer, 2011. `www.cs.ox.ac.uk/ucs/`.

[23] D. Sangiorgi. Types, or: Where's the difference between CCS and π? In *Proceedings of CONCUR*, volume 2421, pages 76–97. Springer LNCS, 2002.

[24] S. Schneider, H. Treharne, and H. Wehrheim. A CSP approach to control in Event-B. In *Proceedings of IFM*, volume 6396. Springer CCIS, 2010.

[25] S. Schneider, H. Treharne, and H. Wehrheim. A CSP account of Event-B refinement. Unpublished, 2011.

[26] W. A. Sutherland. *Introduction to Metric and Topological Spaces*. Oxford University Press, 1975.

[27] G. S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125, 1968.

[28] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

[29] J. Yantchev and C. R. Jesshope. Adaptive, low latency, deadlock-free packet routing for networks of processors. In *IEE Proceedings of Computers and Digital Techniques*, 1989.

[30] N. Yoshida, M. Berger, and K. Honda. Strong normalisation in the π-Calculus. In *Proceedings of LICS 01*, pages 311–322. IEEE Computer Society Press, 2001.

Appendix A. Proofs for Section 3.2.1

Throughout the section we will use the following notation. For every $u \in \Sigma^\omega$ and $i \in \mathbb{N}$ we will denote by u_i the prefix of u of length i. Then, as explained in Section 3.2.1, $u \in \mathsf{traces}^\omega(P)$ if and only if for each $i \in \mathbb{N}$, $u_i \in \mathsf{traces}(P) \cap \Sigma^*$. Let us recall that for every $i \in \mathbb{N}$, u_i cannot contain $\checkmark$ and is therefore an element of Σ^*. We will frequently make use of the following observation which relies on the set $\mathsf{traces}(P)$ being prefix-closed. If $u_i \in \mathsf{traces}(P)$ for infinitely many $i \in \mathbb{N}$, then $u_i \in \mathsf{traces}(P)$ for all $i \in \mathbb{N}$ and, therefore, $u \in \mathsf{traces}^\omega(P)$. Most proofs will be based on König's Lemma, which we now recall.

Theorem A.1 (König's Lemma). *Suppose that for each $i \in \mathbb{N}$, X_i is a non-empty finite set and $f_i : X_{i+1} \to X_i$ is a total function. Then there is a sequence $\langle x_i \mid i \in \mathbb{N} \rangle$, such that $x_i \in X_i$ and $f_i(x_{i+1}) = x_i$.* $\square$

In our proofs we will define the sets X_i as specific subsets of $\mathsf{traces}(P) \cap \Sigma^i$. For each $i \in \mathbb{N}$, $x_i \in X_i$ and $x_{i+1} \in X_{i+1}$, $f_i(x_{i+1}) = x_i$ will imply that $x_i < x_{i+1}$, where $<$ denotes the strict prefix order of traces on Σ^*. For a given $x_{i+1} \in X_{i+1}$, the choice for $f_i(x_{i+1})$ might not be unique, but we can take an arbitrary prefix x_i of x_{i+1} from X_i satisfying certain properties. Then the sequence $\langle x_i \mid i \in \mathbb{N} \rangle$ will form an infinite chain $x_0 < x_1 < x_2 < \ldots x_n \ldots$ under prefix and $x = \lim_{i=0}^{\infty} x_i \in \mathsf{traces}^\omega(P)$.

Lemma 3.2. *Let $u \in \mathsf{traces}^\omega(a \longrightarrow P)$. Then there exists $u' \in \mathsf{traces}^\omega(P)$, such that $u = \langle a \rangle^\frown u'$.*

Proof. Let $u \in \mathsf{traces}^\omega(a \longrightarrow P)$. By definition, for each $t < u$, $t \in \mathsf{traces}(a \longrightarrow P)$. Then, for each $t < u$, $t = \langle a \rangle^\frown t'$ for some $t' \in \mathsf{traces}(P)$. Let $u = \langle a \rangle^\frown u'$ for some $u' \in \Sigma^\omega$. Then, for each $t' < u'$, $t' \in \mathsf{traces}(P)$. Therefore, by definition, $u' \in \mathsf{traces}^\omega(P)$. $\square$

Lemma 3.3. *Let $u \in \mathsf{traces}^\omega(P \oplus Q)$ for $\oplus \in \{\Box, \sqcap\}$. Then $u \in \mathsf{traces}^\omega(P)$ or $u \in \mathsf{traces}^\omega(Q)$.*

Proof. Let $u \in \mathsf{traces}^\omega(P \oplus Q)$. By definition, for each $i \in \mathbb{N}$, $u_i \in \mathsf{traces}(P \oplus Q)$. Therefore, for each $i \in \mathbb{N}$, $u_i \in \mathsf{traces}(P)$ or $u_i \in \mathsf{traces}(Q)$. Then, due to the pigeonhole principle, $u_i \in \mathsf{traces}(P)$ for infinitely many $i \in \mathbb{N}$ or $u_i \in \mathsf{traces}(Q)$ for infinitely many $i \in \mathbb{N}$. Let without loss of generality the former holds. Then, $u_i \in \mathsf{traces}(P)$ for all $i \in \mathbb{N}$ and, hence, $u \in \mathsf{traces}^\omega(P)$. $\square$

Lemma 3.4. *Let $u \in \mathsf{traces}^\omega(P \,\mathring{,}\, Q)$. Then $u \in \mathsf{traces}^\omega(P)$ or $u = t^\frown u'$ with $t^\frown \langle \checkmark \rangle \in \mathsf{traces}(P)$, $u' \in \mathsf{traces}^\omega(Q)$.*

Proof. Let $u \in \mathsf{traces}^\omega(P \,\mathring{,}\, Q)$. By definition, for each $i \in \mathbb{N}$, $u_i \in \mathsf{traces}(P \,\mathring{,}\, Q)$. Therefore, for each $i \in \mathbb{N}$, $u_i \in \mathsf{traces}(P)$ or $u_i = t_1 {}^\frown t_2$ with $t_1 {}^\frown \langle \checkmark \rangle \in \mathsf{traces}(P) \cap \Sigma^{*\checkmark}$, $t_2 \in \mathsf{traces}(Q) \cap \Sigma^*$. If for each $i \in \mathbb{N}$, $u_i \in \mathsf{traces}(P)$, then, by definition, $u \in \mathsf{traces}^\omega(P)$. Otherwise, there exists $N \in \mathbb{N}$, such that $u_0, u_1, \ldots, u_N \in \mathsf{traces}(P)$, but $u_{N+1} \notin \mathsf{traces}(P)$. Therefore, for $j \geq 1$, $u_{N+j} \notin \mathsf{traces}(P)$. By assumption, for every $i \in \mathbb{N}$, $u_i \in \mathsf{traces}(P \,\mathring{,}\, Q)$. Therefore, for $j \geq 1$, $u_{N+j} = t_j {}^\frown v_j$ where $t_j {}^\frown \langle \checkmark \rangle \in \mathsf{traces}(P)$ (and therefore $t_j \leq u_N$) and $v_j \in \mathsf{traces}(Q)$. Then, there must be some $t \leq u_N$, such that $t_j = t$ for infinitely many u_{N+j}'s. Let us write $u_j = t^\frown w_j$ for $j \geq |t|$. We have that $t^\frown \langle \checkmark \rangle \in \mathsf{traces}(P)$ and infinitely often $w_j \in \mathsf{traces}(Q)$. Since for $j < j'$, $w_j < w_{j'}$, and the set of traces is prefix-closed, $w_j \in \mathsf{traces}(Q)$ for each $j \geq |t|$. Then, by definition, $u' = \lim_{j=|t|}^{\infty} w_j \in \mathsf{traces}^\omega(Q)$. $\square$

Lemma 3.5. *Let $u \in \mathsf{traces}^\omega(P \setminus A)$ and $P \setminus A$ be livelock-free. Then there exists $v \in \mathsf{traces}^\omega(P)$, such that $u = v \upharpoonright (\Sigma \backslash A)$.*

Proof. Let $u \in \mathsf{traces}^\omega(P \setminus A)$. By definition, for each $i \in \mathbb{N}$, $u_i \in \mathsf{traces}(P \setminus A)$, i.e., there exists $v_{j_i} \in \mathsf{traces}(P)$, such that $u_i = v_{j_i} \upharpoonright (\Sigma \backslash A)$.

Let, for $i \in \mathbb{N}$, $\backslash^{-1}(u_i) = \{v \in \mathsf{traces}^\infty(P) \mid v \upharpoonright (\Sigma \backslash A) = u_i\}$. We claim that, for each $i \in \mathbb{N}$, $\backslash^{-1}(u_i)$ is finite. Suppose, for the sake of the argument, that $\backslash^{-1}(u_k)$ is infinite. We will prove that $P \setminus A$ is divergent, which will be a contradiction with $P \setminus A$ being livelock-free. Let $u_k = \langle a_1, a_2, \dots a_k \rangle$. It is clear that $\{a_1, a_2, \dots, a_k\} \cap A = \emptyset$. Then, $\backslash^{-1}(u_k) = ((A^* \cup A^\omega) \, a_1 \, (A^* \cup A^\omega) \, a_2 \, (A^* \cup A^\omega) \dots (A^* \cup A^\omega) \, a_k \, (A^* \cup A^\omega)) \cap \mathsf{traces}^\infty(P)$. Let for $i \in \{0, \dots, k-1\}$, n_i be the maximum number of occurrences of consecutive events from A before the occurrence of a_{i+1} and let n_k be the maximum number of consecutive events from A after a_k. Then, for $i = \{0, \dots, k\}$, $n_i \in \mathbb{N} \cup \{\omega\}$. Since $\backslash^{-1}(u_k)$ is infinite, there exists $j \in \{0, \dots, k\}$, such that $n_j = \omega$. Let $j_{\min}$ be the minimal j with this property. Then, for $i < j_{\min}$, $n_i \in \mathbb{N}$. Let $v \in (A^* a_1 A^* a_2 A^* \dots A^* a_{j_{\min}} A^\omega) \cap \mathsf{traces}^\infty(P)$. Therefore, $v \upharpoonright (\Sigma \backslash A) = \langle a_1, a_2, \dots a_{j_{\min}} \rangle = u_{j_{\min}} \in \mathsf{divergences}(P \setminus A)$ which is a contradiction with $P \setminus A$ being livelock-free. Hence, for $i = \{0, \dots, k\}$, $n_i \in \mathbb{N}$ and therefore, $\backslash^{-1}(u_k)$ is finite. Therefore, for each $i \in \mathbb{N}$, we have:

(1) $\backslash^{-1}(u_i) \neq \emptyset$ because $u_i \in \mathsf{traces}(P \setminus A)$
(2) $\backslash^{-1}(u_i)$ is finite
(3) For each $j > i$, for each $w \in \backslash^{-1}(u_j)$, there exists $v \in \backslash^{-1}(u_i)$, such that $v < w$. The trace v can be defined as an arbitrary prefix of w of $(\Sigma \backslash A)$-length i.

Therefore, by König's Lemma, there exists an infinite sequence $v_{j_1} < v_{j_2} < \dots < v_{j_n} < \dots$, such that for $i \in \mathbb{N}$, $v_{j_i} \in \backslash^{-1}(u_i)$, i.e., $v_{j_i} \in \mathsf{traces}(P)$ and $u_i = v_{j_i} \upharpoonright (\Sigma \backslash A)$. Therefore, $v = \lim_{i=0}^{\infty} v_{j_i} \in \mathsf{traces}^\omega(P)$ and $u = v \upharpoonright (\Sigma \backslash A)$. $\square$

Lemma 3.6. *Let $u \in \mathsf{traces}^\omega(P[R])$. Then there exists $v \in \mathsf{traces}^\omega(P)$, such that $v \, R \, u$.*

Proof. Let $u \in \mathsf{traces}^\omega(P[R])$. By definition, for each $i \in \mathbb{N}$, $u_i \in \mathsf{traces}(P[R]) \cap \Sigma^*$. Therefore, for each $i \in \mathbb{N}$, there exists $v_{j_i} \in \mathsf{traces}(P) \cap \Sigma^*$, such that $v_{j_i} \, R \, u_i$, i.e., $\mathsf{length}(u_i) = \mathsf{length}(v_{j_i}) = i$ and for each $0 \leq k \leq i$, $v_{j_i}(k) \, R \, u_i(k)$. Let, for $i \in \mathbb{N}$, $R^{-1}(u_i) = \{v \in \mathsf{traces}(P) \mid v \, R \, u_i\}$. Then, for $i \in \mathbb{N}$:

(1) $R^{-1}(u_i) \neq \emptyset$ because $u_i \in \mathsf{traces}(P[R])$
(2) $R^{-1}(u_i)$ is finite because Σ, and therefore R, are finite
(3) For each $j > i$ and each $w \in R^{-1}(u_j)$, there exists $v \in R^{-1}(u_i)$, such that $v < w$. The trace v can be constructed as the prefix of w of length i.

Therefore, by König's Lemma, there exists an infinite sequence $v_{j_1} < v_{j_2} < \dots < v_{j_n} < \dots$, such that for $i \in \mathbb{N}$, $v_{j_i} \in R^{-1}(u_i)$, i.e., $v_{j_i} \in \mathsf{traces}(P)$ and $v_{j_i} \, R \, u_i$. Therefore, $v = \lim_{i=0}^{\infty} v_{j_i} \in \mathsf{traces}^\omega(P)$ and $v \, R \, u$. $\square$

Lemma 3.7. *Let $u \in \mathsf{traces}^\omega(P \underset{A}{\|} Q)$. Then there exist $u_1 \in \mathsf{traces}^\infty(P)$, $u_2 \in \mathsf{traces}^\infty(Q)$, such that $u \in u_1 \underset{A}{\|} u_2$ and, $u_1 \in \Sigma^\omega$ or $u_2 \in \Sigma^\omega$.*

Proof. Let $u \in \mathsf{traces}^\omega(P \underset{A}{\|} Q)$. Then, by definition, for each $n \in \mathbb{N}$, there exist $v_{i_n} \in \mathsf{traces}(P) \cap \Sigma^*$ and $w_{j_n} \in \mathsf{traces}(Q) \cap \Sigma^*$, such that $u_n \in v_{i_n} \underset{A}{\|} w_{j_n}$ and $n \leq |v_{i_n}| + |w_{j_n}| \leq 2n$. Therefore, for each such triple (u_n, v_{i_n}, w_{j_n}) there exists a function $f^n : \{1, \dots, n\} \mapsto$

$\{0, 1, 2\}$ specifying a possible interleaving of v_{i_n} and w_{j_n} for obtaining u_n. More specifically, $f^n(i)$ indicates which process contributes for communicating the i-th event of u_n, with 0 denoting both P and Q (for events in A), 1 denoting only P, 2 denoting only Q. Given $u_n = \langle a_1, \ldots a_n \rangle$ and f^n, v_{i_n} and w_{j_n} are identified uniquely as $v_{i_n} = \langle a_i \mid 1 \le i \le n, f^n(i) \subseteq \{0, 1\}\rangle$, $w_{j_n} = \langle a_j \mid 1 \le j \le n, f^n(j) \subseteq \{0, 2\}\rangle$.

Let us define a partially ordered set $((\Sigma^{*\checkmark})^2, \le)$ with $(v, w) \le (v', w')$ iff $v \le v'$ and $w \le w'$, where $\le$ denotes a non-strict prefix on traces. We will prove that there exists an infinite chain $(v_{i_1}, w_{j_1}) \le \ldots \le (v_{i_n}, w_{j_n}) \le \ldots$, such that for each $n \in \mathbb{N}$, $v_{i_n} \in \mathsf{traces}(P)$, $w_{j_n} \in \mathsf{traces}(Q)$, $u_n \in v_{i_n} \underset{A}{\|} w_{j_n}$.

Let for $k \in \mathbb{N}$, $\underset{A}{\|}^{-1}(u_k) = \{(v_{i_k}, w_{j_k}) \mid v_{i_k} \in \mathsf{traces}(P), w_{j_k} \in \mathsf{traces}(Q), u_k \in v_{i_k} \underset{A}{\|} w_{j_k}\}$. Then:

(1) $\underset{A}{\|}^{-1}(u_k) \ne \emptyset$ because $u_k \in \mathsf{traces}(P \underset{A}{\|} Q)$.

(2) $\underset{A}{\|}^{-1}(u_k)$ is finite because Σ is finite.

(3) For each $k > l$ and each $(v_{i_k}, w_{j_k}) \in \underset{A}{\|}^{-1}(u_k)$, there exists $(v_{i_l}, w_{j_l}) \in \underset{A}{\|}^{-1}(u_l)$, such that $(v_{i_l}, w_{j_l}) < (v_{i_k}, w_{j_k})$. The pair of traces (v_{i_l}, w_{j_l}) can be constructed as follows. Let for the triple (u_k, v_{i_k}, w_{j_k}) the function $f^k : \{1, \ldots, n\} \mapsto \{0, 1, 2\}$ specifies a possible interleaving of v_{i_k} and w_{j_k} for obtaining u_k. We define $f^l(i) = f^k(i)$ for $1 \le i \le l$. Then, (v_{i_l}, w_{j_l}) is the pair that is uniquely identified by f^l and u_l.

Therefore, by König's Lemma, for each $n \in \mathbb{N}$, there exist $(v_{i_1}, w_{j_1}) \le (v_{i_2}, w_{j_2}) \le \ldots \le (v_{i_n}, w_{j_n})$, such that for each $1 \le k \le n$, $v_{i_k} \in \mathsf{traces}(P)$, $w_{j_k} \in \mathsf{traces}(Q)$, $u_k \in v_{i_k} \underset{A}{\|} w_{j_k}$, $k \le |v_{i_k}| + |w_{j_k}| \le 2k$. Let $v = \lim_{k=1}^{\infty} v_{i_k}$, $w = \lim_{k=1}^{\infty} w_{j_k}$. Then clearly, $v \in \mathsf{traces}^{\infty}(P)$, $w \in \mathsf{traces}^{\infty}(Q)$ and $u \in v \underset{A}{\|} w$. Let us assume that both v and w are finite, i.e., $|v| = l_v$, $|w| = l_w$ for some $l_v, l_w \in \mathbb{N}$. Then, each prefix of u will be of length at most $l_v + l_w \in \mathbb{N}$, which is a contradiction with u being infinite. Therefore, at least one of v and w is infinite. $\square$

Appendix B. Proofs for Section 4

Lemma B.1 ([26, Lemma 9.2.5]). *In any metric space, if s is a Cauchy sequence that has a subsequence that converges to a point x, then s also converges to x.* $\square$

Proposition B.2. *Let $U \subseteq \Sigma$. Then $\mathcal{T}^{\Downarrow}$ equipped with the metric d_U is a complete metric space.*

Proof. We will prove that every Cauchy sequence converges.

Let $\langle P_i \mid i \in \mathbb{N} \rangle$ be a Cauchy sequence in $(\mathcal{T}^{\Downarrow}, d_U)$. By definition, for every $\varepsilon > 0$, there exists $N_\varepsilon \in \mathbb{N}$ such that, for every $n, m \ge N_\varepsilon$, $d_U(P_n, P_m) < \varepsilon$. Therefore, for every $r \in \mathbb{N}$ and $\varepsilon = 2^{-r}$, there exists $N_r \in \mathbb{N}$ such that, for every $n, m \ge N_r$, $d_U(P_n, P_m) < 2^{-r}$, i.e., $P_n \upharpoonright_U r = P_m \upharpoonright_U r$. Then, for every $r, m \in \mathbb{N}$, $d_U(P_{N_r}, P_{N_{r+m}}) < 2^{-r}$. Therefore, the subsequence $\langle P_{N_r} \mid r \in \mathbb{N} \rangle$ of $\langle P_i \mid i \in \mathbb{N} \rangle$ is itself a Cauchy sequence.

Let us define $P = \prod_{q \in \mathbb{N}} \bigsqcup_{r \ge q} P_{N_r}$. $P \in \mathcal{T}^{\Downarrow}$ because $(\mathcal{T}^{\Downarrow}, \sqsubseteq)$ is a complete lattice. We will prove that the subsequence $\langle P_{N_r} \mid r \in \mathbb{N} \rangle$ converges to P, i.e., that for every $r \in \mathbb{N}$, $d_U(P_{N_r}, P) < 2^{-r}$.

Let us fix r. Suppose, for the sake of the argument, that $d_U(P_{N_r}, P) \geq 2^{-r}$ and let, without loss of generality, P_{N_r} and P disagree on the sets of their divergences. Therefore, there exists $t \in \Sigma^{*\checkmark}$ such that $\mathsf{length}_U(t) < r$ and, either $t \in \mathsf{divergences}(P_{N_r}) \backslash \mathsf{divergences}(P)$ or $t \in \mathsf{divergences}(P) \backslash \mathsf{divergences}(P_{N_r})$. To remind, by construction we have $\mathsf{divergences}(P) = \bigcup_{q \in \mathbb{N}} \bigcap_{r \geq q} \mathsf{divergences}(P_{N_r})$. We explore both alternatives.

- Suppose $t \in \mathsf{divergences}(P_{N_r}) \backslash \mathsf{divergences}(P)$. Since $t \notin \mathsf{divergences}(P)$, for every $q \in \mathbb{N}$ there exists $s_q \geq q$ such that $t \notin \mathsf{divergences}(P_{N_{s_q}})$. Therefore, for $q = r$ there exists $s_r \geq r$ such that $t \notin \mathsf{divergences}(P_{N_{s_r}})$. Hence, since $t \in \mathsf{divergences}(P_{N_r})$ and $\mathsf{length}_U(t) < r$, $d_U(P_{N_r}, P_{N_{s_r}}) \geq 2^{-r}$ which is a contradiction with $d_U(P_{N_r}, P_{N_{r+m}}) < 2^{-r}$ for $m \geq 0$.
- Therefore, $t \in \mathsf{divergences}(P) \backslash \mathsf{divergences}(P_{N_r})$. Since $t \in \mathsf{divergences}(P)$, there exists $q \in \mathbb{N}$ such that for every $s \geq q$, $t \in \mathsf{divergences}(P_{N_s})$. However, as $t \notin \mathsf{divergences}(P_{N_r})$ and $\mathsf{length}_U(t) < r$, for every $s \geq r$, $t \notin \mathsf{divergences}(P_{N_s})$ which again leads to a contradiction.

Therefore, for every $r \in \mathbb{N}$, $d_U(P_{N_r}, P) < 2^{-r}$ and, hence, the subsequence $\langle P_{N_r} \mid r \in \mathbb{N} \rangle$ converges to P. Therefore, from Lemma B.1, $\langle P_i \mid i \in \mathbb{N} \rangle$ also converges to P and, hence, $(\mathcal{T}^{\Downarrow}, d_U)$ is a complete metric space. □

Proposition B.3. *Let $U \subseteq \Sigma$. Then the set of livelock-free processes is a closed subset of $(\mathcal{T}^{\Downarrow}, d_U)$.*

Proof. Let $\langle P_i \mid i \in \mathbb{N} \rangle$ be a sequence of livelock-free elements of $\mathcal{T}^{\Downarrow}$ converging to a process $Q \in \mathcal{T}^{\Downarrow}$. Therefore, by definition, for every $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that, for every $n \geq N$, $d_U(P_n, Q) < \varepsilon$. We will prove that Q is also livelock-free.

Suppose for the sake of the argument that Q can diverge. Let $t \in \mathsf{divergences}(Q)$ and $\mathsf{length}_U(t) = k$. If we take $\varepsilon = 2^{-k}$, since $\langle P_i \mid i \in \mathbb{N} \rangle$ converges to Q, there exists $N_t \in \mathbb{N}$ such that, for every $n \geq N_t$, $d_U(P_n, Q) < 2^{-k}$ and, therefore, $P_n \upharpoonright_U k = Q \upharpoonright_U k$. Therefore, for every $n \geq N_t$, $t \in \mathsf{divergences}(P_n)$, which is a contradiction with $\langle P_i \mid i \in \mathbb{N} \rangle$ being all livelock-free.

Therefore, Q is livelock-free and, hence, the set of livelock-free processes is closed. □

APPENDIX C. PROOFS FOR SECTION 4.1

Throughout this section let us fix a set of events $U \subseteq \Sigma$.

Lemma 4.3 ($\mathbin{\raise0.3ex\hbox{$\scriptscriptstyle\circ$}\mkern-4mu\raise-0.3ex\hbox{$\scriptscriptstyle\circ$}}_1$). *For any CSP processes P, P', and Q:*
$$d_U(P \mathbin{\raise0.3ex\hbox{$\scriptscriptstyle\circ$}\mkern-4mu\raise-0.3ex\hbox{$\scriptscriptstyle\circ$}} Q, P' \mathbin{\raise0.3ex\hbox{$\scriptscriptstyle\circ$}\mkern-4mu\raise-0.3ex\hbox{$\scriptscriptstyle\circ$}} Q) \leq d_U(P, P').$$

Proof. Suppose $(T_P, D_P) \upharpoonright_U k = (T_{P'}, D_{P'}) \upharpoonright_U k$. We will prove that $(T_{P \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q}, D_{P \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q}) \upharpoonright_U k = (T_{P' \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q}, D_{P' \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q}) \upharpoonright_U k$, from which we can conclude that $d_U(P \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q, P' \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q) \leq d_U(P, P')$.

Let $t \in \mathsf{divergences}(P \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q)$ and $\mathsf{length}_U(t) \leq k$. We will prove that $t \in \mathsf{divergences}(P' \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q)$ and therefore, $D_{P \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q} \upharpoonright_U k \subseteq D_{P' \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q} \upharpoonright_U k$. The reverse containment is established similarly by symmetry.

Since $t \in \mathsf{divergences}(P \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q)$, by definition, $t \in \mathsf{divergences}(P)$ or $t = t_1 {}^\frown t_2$ with $t_1 {}^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(P)$, $t_2 \in \mathsf{divergences}(Q)$. We consider both cases.

- Suppose $t \in \mathsf{divergences}(P)$. Since $\mathsf{length}_U(t) \leq k$ and $(T_P, D_P) \upharpoonright_U k = (T_{P'}, D_{P'}) \upharpoonright_U k$, $t \in \mathsf{divergences}(P')$. Therefore, by definition, $t \in \mathsf{divergences}(P' \mathbin{\raise0.2ex\hbox{$\scriptscriptstyle\circ$}\mkern-3mu\raise-0.2ex\hbox{$\scriptscriptstyle\circ$}} Q)$.

- Suppose $t = t_1 ^\frown t_2$ with $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(P)$, $t_2 \in \mathsf{divergences}(Q)$. Observe that $\mathsf{length}_U(t_1 ^\frown \langle \checkmark \rangle) = \mathsf{length}_U(t_1) \leq \mathsf{length}_U(t) \leq k$. Then $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(P')$, since $(T_P, D_P) \restriction_U k = (T_{P'}, D_{P'}) \restriction_U k$. Hence, by definition, $t_1 ^\frown t_2 = t \in \mathsf{divergences}(P' \mathbin{\mathring{,}} Q)$.

Now let $t \in \mathsf{traces}_\perp(P \mathbin{\mathring{,}} Q)$ and $\mathsf{length}_U(t) \leq k$. We will prove that $t \in \mathsf{traces}_\perp(P' \mathbin{\mathring{,}} Q)$ and therefore, $T_{P \mathbin{\mathring{,}} Q} \restriction_U k \subseteq T_{P' \mathbin{\mathring{,}} Q} \restriction_U k$. The reverse containment is established similarly by symmetry. Since $t \in \mathsf{traces}_\perp(P \mathbin{\mathring{,}} Q)$, $t \in \mathsf{divergences}(P \mathbin{\mathring{,}} Q)$ or $t \in \mathsf{traces}(P \mathbin{\mathring{,}} Q)$. The latter reduces to $t \in \mathsf{traces}(P) \cap \Sigma^*$ or $t = t_1 ^\frown t_2$ with $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(P)$, $t_2 \in \mathsf{traces}(Q)$. We consider all three alternatives.

- Suppose first that $t \in \mathsf{divergences}(P \mathbin{\mathring{,}} Q)$. We already proved that $t \in \mathsf{divergences}(P' \mathbin{\mathring{,}} Q)$ and therefore, $t \in \mathsf{traces}_\perp(P' \mathbin{\mathring{,}} Q)$.
- Suppose now what $t \in \mathsf{traces}(P) \cap \Sigma^*$. Therefore, $t \in \mathsf{traces}_\perp(P) \cap \Sigma^*$. Then, since $(T_P, D_P) \restriction_U k = (T_{P'}, D_{P'}) \restriction_U k$, $t \in \mathsf{traces}_\perp(P') \cap \Sigma^*$.
  - If $t \in \mathsf{traces}(P') \cap \Sigma^*$, then by definition, $t \in \mathsf{traces}(P' \mathbin{\mathring{,}} Q) \subseteq \mathsf{traces}_\perp(P' \mathbin{\mathring{,}} Q)$.
  - If $t \in \mathsf{divergences}(P') \cap \Sigma^*$, then by definition, $t \in \mathsf{divergences}(P' \mathbin{\mathring{,}} Q) \subseteq \mathsf{traces}_\perp(P' \mathbin{\mathring{,}} Q)$.
- Suppose finally that $t = t_1 ^\frown t_2$ with $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(P)$, $t_2 \in \mathsf{traces}(Q)$. We note that $\mathsf{length}_U(t_1 ^\frown \langle \checkmark \rangle) = \mathsf{length}_U(t_1) \leq \mathsf{length}_U(t) \leq k$. Then, since $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(P)$ and $(T_P, D_P) \restriction_U k = (T_{P'}, D_{P'}) \restriction_U k$, $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(P')$.
  - Let $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(P')$. By definition, $t \in \mathsf{traces}(P' \mathbin{\mathring{,}} Q) \subseteq \mathsf{traces}_\perp(P' \mathbin{\mathring{,}} Q)$.
  - Let $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{divergences}(P')$. By Axiom 2 of $\mathcal{T}^\Downarrow$, $t_1 \in \mathsf{divergences}(P')$. Since $t_1 \in \Sigma^*$, by Axiom 4 of $\mathcal{T}^\Downarrow$, $t = t_1 ^\frown t_2 \in \mathsf{divergences}(P')$. Then by definition, $t \in \mathsf{divergences}(P' \mathbin{\mathring{,}} Q) \subseteq \mathsf{traces}_\perp(P' \mathbin{\mathring{,}} Q)$.

Therefore, $(T_{P \mathbin{\mathring{,}} Q}, D_{P \mathbin{\mathring{,}} Q}) \restriction_U k = (T_{P' \mathbin{\mathring{,}} Q}, D_{P' \mathbin{\mathring{,}} Q}) \restriction_U k$ and, hence, $d_U(P \mathbin{\mathring{,}} Q, P' \mathbin{\mathring{,}} Q) \leq d_U(P, P')$. □

Lemma 4.3 ($\mathbin{\mathring{,}}_2$). *For any CSP processes P, Q, and Q':*

$$d_U(P \mathbin{\mathring{,}} Q, P \mathbin{\mathring{,}} Q') \leq d_U(Q, Q').$$

Proof. Suppose $(T_Q, D_Q) \restriction_U k = (T_{Q'}, D_{Q'}) \restriction_U k$. We will prove that $(T_{P \mathbin{\mathring{,}} Q}, D_{P \mathbin{\mathring{,}} Q}) \restriction_U k = (T_{P \mathbin{\mathring{,}} Q'}, D_{P \mathbin{\mathring{,}} Q'}) \restriction_U k$, from which $d_U(P \mathbin{\mathring{,}} Q, P \mathbin{\mathring{,}} Q') \leq d_U(Q, Q')$ follows immediately.

Let $t \in \mathsf{divergences}(P \mathbin{\mathring{,}} Q)$ and $\mathsf{length}_U(t) \leq k$.

- Suppose $t \in \mathsf{divergences}(P)$. By definition, $t \in \mathsf{divergences}(P \mathbin{\mathring{,}} Q')$.
- Suppose $t = t_1 ^\frown t_2$ with $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(P)$, $t_2 \in \mathsf{divergences}(Q)$ and $\mathsf{length}_U(t_2) \leq \mathsf{length}_U(t) \leq k$. Since by assumption $D_Q \restriction_U k = D_{Q'} \restriction_U k$, $t_2 \in \mathsf{divergences}(Q')$. Then by definition, $t \in \mathsf{divergences}(P \mathbin{\mathring{,}} Q')$.

Let $t \in \mathsf{traces}_\perp(P \mathbin{\mathring{,}} Q)$ and $\mathsf{length}_U(t) \leq k$.

- Let first $t \in \mathsf{divergences}(P \mathbin{\mathring{,}} Q)$. We already proved that $t \in \mathsf{divergences}(P \mathbin{\mathring{,}} Q')$ and therefore, $t \in \mathsf{traces}_\perp(P \mathbin{\mathring{,}} Q')$.
- Let now $t \in \mathsf{traces}(P) \cap \Sigma^*$. Then by definition, $t \in \mathsf{traces}(P \mathbin{\mathring{,}} Q') \subseteq \mathsf{traces}_\perp(P \mathbin{\mathring{,}} Q')$.
- Let finally $t = t_1 ^\frown t_2$ with $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(P) \subseteq \mathsf{traces}_\perp(P)$, $t_2 \in \mathsf{traces}(Q) \subseteq \mathsf{traces}_\perp(Q)$. Since $\mathsf{length}_U(t) \leq k$, $\mathsf{length}_U(t_2) \leq k$. Then, by assumption, $t_2 \in \mathsf{traces}_\perp(Q')$.
  - If $t_2 \in \mathsf{traces}(Q')$, by definition, $t = t_1 ^\frown t_2 \in \mathsf{traces}(P \mathbin{\mathring{,}} Q') \subseteq \mathsf{traces}_\perp(P \mathbin{\mathring{,}} Q')$.
  - Let $t_2 \in \mathsf{divergences}(Q')$. Since $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(P)$, by definition, $t = t_1 ^\frown t_2 \in \mathsf{divergences}(P \mathbin{\mathring{,}} Q') \subseteq \mathsf{traces}_\perp(P \mathbin{\mathring{,}} Q')$.

Therefore, $D_{P\,\S\,Q} \restriction_U k \subseteq D_{P\,\S\,Q'} \restriction_U k$ and $T_{P\,\S\,Q} \restriction_U k \subseteq T_{P\,\S\,Q'} \restriction_U k$. The reverse containments are established similarly by symmetry. Therefore, $(T_{P\,\S\,Q}, D_{P\,\S\,Q}) \restriction_U k = (T_{P\,\S\,Q'}, D_{P\,\S\,Q'}) \restriction_U k$ and, hence, $d_U(P\,\S\,Q, P\,\S\,Q') \leq d_U(Q, Q')$. $\square$

Lemma 4.7. *Let P, Q, and Q' be CSP processes. Let P always communicate an event from $U \subseteq \Sigma$ before it does a $\checkmark$. Then:*

$$d_U(P\,\S\,Q, P\,\S\,Q') \leq \frac{1}{2} d_U(Q, Q').$$

Proof. Suppose $(T_Q, D_Q) \restriction_U k = (T_{Q'}, D_{Q'}) \restriction_U k$. We will prove that $(T_{P\,\S\,Q}, D_{P\,\S\,Q}) \restriction_U k+1 = (T_{P\,\S\,Q'}, D_{P\,\S\,Q'}) \restriction_U k+1$, which implies $d_U(P\,\S\,Q, P\,\S\,Q') \leq \frac{1}{2} d_U(Q, Q')$.

Let $t \in \mathsf{traces}_\perp(P\,\S\,Q)$ and $\mathsf{length}_U(t) \leq k+1$.

- Suppose $t \in \mathsf{divergences}(P\,\S\,Q)$.
  - If $t \in \mathsf{divergences}(P)$, by definition, $t \in \mathsf{divergences}(P\,\S\,Q') \subseteq \mathsf{traces}_\perp(P\,\S\,Q')$.
  - Let $t = t_1 \frown t_2$ with $t_1 \frown \langle\checkmark\rangle \in \mathsf{traces}_\perp(P)$, $t_2 \in \mathsf{divergences}(Q) \subseteq \mathsf{traces}_\perp(Q)$. Since P always communicates an event from $U \subseteq \Sigma$ before it can do a $\checkmark$, t_1 contains an event from U. Therefore, $\mathsf{length}_U(t_2) \leq k$. Then, since by assumption $(T_Q, D_Q) \restriction_U k = (T_{Q'}, D_{Q'}) \restriction_U k$, $t_2 \in \mathsf{divergences}(Q')$. Therefore, by definition, $t = t_1 \frown t_2 \in \mathsf{divergences}(P\,\S\,Q') \subseteq \mathsf{traces}_\perp(P\,\S\,Q')$.
- Suppose $t \in \mathsf{traces}(P\,\S\,Q)$.
  - If $t \in \mathsf{traces}(P) \cap \Sigma^*$, then by definition, $t \in \mathsf{traces}(P\,\S\,Q') \subseteq \mathsf{traces}_\perp(P\,\S\,Q')$.
  - Let $t = t_1 \frown t_2$ with $t_1 \frown \langle\checkmark\rangle \in \mathsf{traces}(P)$, $t_2 \in \mathsf{traces}(Q)$. Since P always communicates an event from $U \subseteq \Sigma$ before it does a $\checkmark$, t_1 contains an event from U. Therefore, $\mathsf{length}_U(t_2) \leq k$. Then, by assumption, $t_2 \in \mathsf{traces}_\perp(Q')$.
    * If $t_2 \in \mathsf{traces}(Q')$, by definition, $t = t_1 \frown t_2 \in \mathsf{traces}(P\,\S\,Q') \subseteq \mathsf{traces}_\perp(P\,\S\,Q')$.
    * Let $t_2 \in \mathsf{divergences}(Q')$. Since $t_1 \frown \langle\checkmark\rangle \in \mathsf{traces}(P)$, by definition, $t = t_1 \frown t_2 \in \mathsf{divergences}(P\,\S\,Q') \subseteq \mathsf{traces}_\perp(P\,\S\,Q')$.

Therefore, $D_{P\,\S\,Q} \restriction_U k+1 \subseteq D_{P\,\S\,Q'} \restriction_U k+1$ and $T_{P\,\S\,Q} \restriction_U k+1 \subseteq T_{P\,\S\,Q'} \restriction_U k+1$. The reverse containments are established similarly by symmetry. Therefore, $(T_{P\,\S\,Q}, D_{P\,\S\,Q}) \restriction_U k+1 = (T_{P\,\S\,Q'}, D_{P\,\S\,Q'}) \restriction_U k+1$ and, hence, $d_U(P\,\S\,Q, P\,\S\,Q') \leq \frac{1}{2} d_U(Q, Q')$. $\square$

Lemma 4.3 ($\sqcap$). *For any CSP processes P, P', and Q:*
$$d_U(P \sqcap Q, P' \sqcap Q) \leq d_U(P, P').$$

Proof. Suppose $(T_P, D_P) \restriction_U k = (T_{P'}, D_{P'}) \restriction_U k$. We will prove that $(T_{P\sqcap Q}, D_{P\sqcap Q}) \restriction_U k = (T_{P'\sqcap Q}, D_{P'\sqcap Q}) \restriction_U k$, which directly implies $d_U(P \sqcap Q, P' \sqcap Q) \leq d_U(P, P')$.

Let $t \in \mathsf{divergences}(P \sqcap Q)$ and $\mathsf{length}_U(t) \leq k$.

- Suppose $t \in \mathsf{divergences}(P)$. By assumption, $D_P \restriction_U k = D_{P'} \restriction_U k$. Therefore, $t \in \mathsf{divergences}(P') \subseteq \mathsf{divergences}(P' \sqcap Q)$.
- Suppose $t \in \mathsf{divergences}(Q)$. By definition, $t \in \mathsf{divergences}(P' \sqcap Q)$.

Let $t \in \mathsf{traces}_\perp(P \sqcap Q)$ and $\mathsf{length}_U(t) \leq k$. We have that $\mathsf{traces}_\perp(P \sqcap Q) = \mathsf{traces}(P \sqcap Q) \cup \mathsf{divergences}(P \sqcap Q) = \mathsf{traces}(P) \cup \mathsf{divergences}(P) \cup \mathsf{traces}(Q) \cup \mathsf{divergences}(Q) = \mathsf{traces}_\perp(P) \cup \mathsf{traces}_\perp(Q)$.

- Let $t \in \mathsf{traces}_\perp(P)$. By assumption, $T_P \restriction_U k = T_{P'} \restriction_U k$. Therefore, $t \in \mathsf{traces}_\perp(P') \subseteq \mathsf{traces}_\perp(P' \sqcap Q)$.

• Let $t \in \mathsf{traces}_\perp(Q)$. By definition, $t \in \mathsf{traces}_\perp(P' \sqcap Q)$.

Therefore, $D_{P \sqcap Q} \upharpoonright_U k \subseteq D_{P' \sqcap Q} \upharpoonright_U k$ and $T_{P \sqcap Q} \upharpoonright_U k \subseteq T_{P' \sqcap Q} \upharpoonright_U k$. The reverse containments are established similarly by symmetry. Therefore, $(T_{P \sqcap Q}, D_{P \sqcap Q}) \upharpoonright_U k = (T_{P' \sqcap Q}, D_{P' \sqcap Q}) \upharpoonright_U k$ and, hence, $d_U(P \sqcap Q, P' \sqcap Q) \leq d_U(P, P')$. $\square$

Lemma 4.3 ($\square$). *For any CSP processes P, P', and Q:*
$$d_U(P \square Q, P' \square Q) \leq d_U(P, P').$$

Proof. Same as for $\sqcap$. $\square$

Lemma 4.3 ($\underset{A}{\|}$). *For any CSP processes P, P', and Q and any $A \subseteq \Sigma$:*
$$d_U(P \underset{A}{\|} Q, P' \underset{A}{\|} Q) \leq d_U(P, P').$$

Proof. Suppose $(T_P, D_P) \upharpoonright_U k = (T_{P'}, D_{P'}) \upharpoonright_U k$. We will prove that $(T_{P \underset{A}{\|} Q}, D_{P \underset{A}{\|} Q}) \upharpoonright_U k = (T_{P' \underset{A}{\|} Q}, D_{P' \underset{A}{\|} Q}) \upharpoonright_U k$, which directly implies $d_U(P \underset{A}{\|} Q, P' \underset{A}{\|} Q) \leq d_U(P, P')$.

Let $t \in \mathsf{divergences}(P \underset{A}{\|} Q)$ and $\mathsf{length}_U(t) \leq k$. Therefore, $t = u^\frown v$ with $u \in (s \underset{A}{\|} r \cap \Sigma^*)$, $s \in \mathsf{traces}_\perp(P)$, $r \in \mathsf{traces}_\perp(Q)$ and, $s \in \mathsf{divergences}(P)$ or $r \in \mathsf{divergences}(Q)$. Let us recall that v ranges over $\Sigma^{*\checkmark}$, in accordance with Axiom 4. Let us observe that $\mathsf{length}_U(s) \leq \mathsf{length}_U(u) \leq \mathsf{length}_U(t) \leq k$. Therefore, by assumption, $s \in \mathsf{traces}_\perp(P')$.

• Let $s \in \mathsf{divergences}(P)$. By assumption, $s \in \mathsf{divergences}(P')$. Therefore by definition, $t \in \mathsf{divergences}(P' \underset{A}{\|} Q)$.

• Let $r \in \mathsf{divergences}(Q)$. Since $s \in \mathsf{traces}_\perp(P')$, by definition, $t \in \mathsf{divergences}(P' \underset{A}{\|} Q)$.

Let $t \in \mathsf{traces}_\perp(P \underset{A}{\|} Q)$ and $\mathsf{length}_U(t) \leq k$.

• Suppose $t \in \mathsf{divergences}(P \underset{A}{\|} Q)$. We already proved that $t \in \mathsf{divergences}(P \underset{A}{\|} Q') \subseteq \mathsf{traces}_\perp(P' \underset{A}{\|} Q)$.

• Suppose $t \in \mathsf{traces}(P \underset{A}{\|} Q)$. Therefore, there exist $s \in \mathsf{traces}(P) \subseteq \mathsf{traces}_\perp(P)$, $r \in \mathsf{traces}(Q) \subseteq \mathsf{traces}_\perp(Q)$, such that $t \in s \underset{A}{\|} r$. By assumption, $s \in \mathsf{traces}_\perp(P')$.

  – If $s \in \mathsf{traces}(P')$, by definition, $t \in \mathsf{traces}(P' \underset{A}{\|} Q) \subseteq \mathsf{traces}_\perp(P' \underset{A}{\|} Q)$.

  – If $s \in \mathsf{divergences}(P')$, by definition, $t \in \mathsf{divergences}(P' \underset{A}{\|} Q) \subseteq \mathsf{traces}_\perp(P' \underset{A}{\|} Q)$.

Therefore, $D_{P \underset{A}{\|} Q} \upharpoonright_U k \subseteq D_{P' \underset{A}{\|} Q} \upharpoonright_U k$ and $T_{P \underset{A}{\|} Q} \upharpoonright_U k \subseteq T_{P' \underset{A}{\|} Q} \upharpoonright_U k$. The reverse containments are established similarly by symmetry. Therefore, $(T_{P \underset{A}{\|} Q}, D_{P \underset{A}{\|} Q}) \upharpoonright_U k = (T_{P' \underset{A}{\|} Q}, D_{P' \underset{A}{\|} Q}) \upharpoonright_U k$ and, hence, $d_U(P \underset{A}{\|} Q, P' \underset{A}{\|} Q) \leq d_U(P, P')$. $\square$

Lemma 4.5. *Let P and Q be CSP processes and let $A \subseteq \Sigma$ satisfy $A \cap U = \emptyset$. Then:*

$$d_U(P \setminus A, Q \setminus A) \leq d_U(P, Q).$$

Proof. Suppose $(T_P, D_P) \upharpoonright_U k = (T_Q, D_Q) \upharpoonright_U k$. We will prove that $(T_{P \setminus A}, D_{P \setminus A}) \upharpoonright_U k = (T_{Q \setminus A}, D_{Q \setminus A}) \upharpoonright_U k$, which implies $d_U(P \setminus A, Q \setminus A) \leq d_U(P, Q)$.

Let $t \in \mathsf{divergences}(P \setminus A)$ and $\mathsf{length}_U(t) \leq k$. We consider the possible alternatives for t.
- Suppose that there exists $s \in \mathsf{divergences}(P)$, such that $t = (s \upharpoonright (\Sigma \setminus A))^\frown r$. Since $A \cap U = \emptyset$, $\mathsf{length}_U(s) = \mathsf{length}_U(s \upharpoonright (\Sigma \setminus A)) \leq \mathsf{length}_U(t) \leq k$. Then, by assumption, $s \in \mathsf{divergences}(Q)$. Therefore, by definition, $t \in \mathsf{divergences}(Q \setminus A)$.
- Now suppose that there exists $u \in \Sigma^\omega$, such that $u \upharpoonright (\Sigma \setminus A)$ is finite, for each $s < u$, $s \in \mathsf{traces}_\perp(P)$, and $t = u \upharpoonright (\Sigma \setminus A) ^\frown r$. Since $A \cap U = \emptyset$, $\mathsf{length}_U(u) = \mathsf{length}_U(u \upharpoonright (\Sigma \setminus A)) \leq \mathsf{length}_V(t) \leq k$. Then, by assumption, for each $s < u$, $s \in \mathsf{traces}_\perp(Q)$. Hence $t \in \mathsf{divergences}(Q \setminus A)$ follows by definition.

Let $t \in \mathsf{traces}_\perp(P \setminus A)$ and $\mathsf{length}_U(t) \leq k$.
- Let first $t \in \mathsf{divergences}(P \setminus A)$. We already proved that $t \in \mathsf{divergences}(Q \setminus A) \subseteq \mathsf{traces}_\perp(Q \setminus A)$.
- Let now $t \in \mathsf{traces}(P \setminus A)$. Therefore, there exists $s \in \mathsf{traces}(P) \subseteq \mathsf{traces}_\perp(P)$, such that $t = s \upharpoonright (\Sigma \setminus A)$. From $A \cap U = \emptyset$, $\mathsf{length}_U(s) = \mathsf{length}_U(s \upharpoonright (\Sigma \setminus A)) = \mathsf{length}_U(t) \leq k$. Then, by assumption, $s \in \mathsf{traces}_\perp(Q)$.
  - If $s \in \mathsf{divergences}(Q)$, by definition, $t \in \mathsf{divergences}(Q \setminus A) \subseteq \mathsf{traces}_\perp(Q \setminus A)$.
  - If $s \in \mathsf{traces}(Q)$, by definition, $t \in \mathsf{traces}(Q \setminus A) \subseteq \mathsf{traces}_\perp(Q \setminus A)$.

Therefore, $D_{P \setminus A} \upharpoonright_U k \subseteq D_{Q \setminus A} \upharpoonright_U k$ and $T_{P \setminus A} \upharpoonright_U k \subseteq T_{Q \setminus A} \upharpoonright_U k$. The reverse containments are established similarly by symmetry. Therefore, $(T_{P \setminus A}, D_{P \setminus A}) \upharpoonright_U k = (T_{Q \setminus A}, D_{Q \setminus A}) \upharpoonright_U k$ and, hence, $d_U(P \setminus A, Q \setminus A) \leq d_U(P, Q)$. $\square$

Lemma 4.6. *Let P and Q be CSP processes, $R \subseteq \Sigma \times \Sigma$ be a renaming relation on Σ and $R(U) = \{y \mid \exists x \in U . x \, R \, y\}$. Then:*

$$d_{R(U)}(P[R], Q[R]) \leq d_U(P, Q).$$

Proof. Suppose $(T_P, D_P) \upharpoonright_U k = (T_Q, D_Q) \upharpoonright_U k$. We will prove that $(T_{P[R]}, D_{P[R]}) \upharpoonright_{R(U)} k = (T_{Q[R]}, D_{Q[R]}) \upharpoonright_{R(U)} k$.

Let $t \in \mathsf{divergences}(P[R])$ and $\mathsf{length}_{R(U)}(t) \leq k$. Then there exist $s_1, t_1 \in \Sigma^*, r \in \Sigma^{*\checkmark}$, such that $s_1 \in \mathsf{divergences}(P) \cap \Sigma^*$, $s_1 \, R \, t_1$ and $t = t_1 ^\frown r$. Then, $\mathsf{length}(s_1) = \mathsf{length}(t_1)$ and for $1 \leq i \leq \mathsf{length}(s_1)$, $s_{1_i} \, R \, t_{1_i}$. Therefore, $\mathsf{length}_U(s_1) = \mathsf{length}_{R(U)}(t_1) \leq \mathsf{length}_{R(U)}(t) \leq k$ and, by assumption, $s_1 \in \mathsf{divergences}(Q) \cap \Sigma^*$. Hence, by definition, $t \in \mathsf{divergences}(Q[R])$.

Let $t \in \mathsf{traces}_\perp(P[R])$ and $\mathsf{length}_{R(U)}(t) \leq k$.
- If $t \in \mathsf{divergences}(P[R])$, we already proved that $t \in \mathsf{divergences}(Q[R]) \subseteq \mathsf{traces}_\perp(Q[R])$.
- Let $t \in \mathsf{traces}(P[R])$. Then there exists $s \in \mathsf{traces}(P)$, such that $s \, R \, t$. Therefore, $\mathsf{length}_U(s) = \mathsf{length}_{R(U)}(t) \leq k$ and, by assumption, $s \in \mathsf{traces}_\perp(Q)$.
  - If $s \in \mathsf{traces}(Q)$, by definition, $t \in \mathsf{traces}(Q[R]) \subseteq \mathsf{traces}_\perp(Q[R])$.
  - If $s \in \mathsf{divergences}(Q)$, by definition, $t \in \mathsf{divergences}(Q[R]) \subseteq \mathsf{traces}_\perp(Q[R])$.

Therefore, $D_{P[R]} \upharpoonright_{R(U)} k \subseteq D_{Q[R]} \upharpoonright_{R(U)} k$ and $T_{P[R]} \upharpoonright_{R(U)} k \subseteq T_{Q[R]} \upharpoonright_{R(U)} k$. The reverse containments are established similarly by symmetry. Therefore, $(T_{P[R]}, D_{P[R]}) \upharpoonright_{R(U)} k = (T_{Q[R]}, D_{Q[R]}) \upharpoonright_{R(U)} k$ and, hence, $d_{R(U)}(P[R], Q[R]) \leq d_U(P, Q)$. □

APPENDIX D. PROOFS FOR SECTION 5

Proposition 5.2.   *Let $P(X, Y_1, \ldots, Y_n) = P(X, \overline{Y})$ be a CSP term whose free variables are contained within the set $\{X, Y_1, \ldots, Y_n\}$. Let $N_X : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ be defined recursively on the structure of P as shown in Figure 6. If $(U, V) \in N_X(P)$, then for all $T_1, T_2, \Theta_1, \ldots, \Theta_n \in \mathcal{T}^{\Downarrow}$, $d_V(P(T_1, \overline{\Theta}), P(T_2, \overline{\Theta})) \leq d_U(T_1, T_2)$.*

Proof. Structural induction on P. Let us take arbitrary $T_1, T_2, \Theta_1, \ldots, \Theta_n \in \mathcal{T}^{\Downarrow}$.
- $N_X(P) \widehat{=} \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$ whenever X is not free in P.

  Proof. Let $(U, V) \in N_X(P)$. Then
  $$d_U(T_1, T_2) \geq 0 = d_V(P(T_1, \overline{\Theta}), P(T_2, \overline{\Theta})) = d_V(P(\overline{\Theta}), P(\overline{\Theta})). \qquad \square$$
- $N_X(a \longrightarrow P) \widehat{=} N_X(P)$.

  Proof. Suppose $(U, V) \in N_X(a \longrightarrow P)$. By construction, $(U, V) \in N_X(P)$. Then:
  $$\begin{aligned} d_U(T_1, T_2) \ &\geq d_V(P(T_1, \overline{\Theta}), P(T_2, \overline{\Theta})) && \text{// induction hypothesis} \\ &\geq d_V(a \longrightarrow P(T_1, \overline{\Theta}), a \longrightarrow P(T_2, \overline{\Theta})) && \text{// Lemma 4.4, > if } a \in V \\ &= d_V((a \longrightarrow P)(T_1, \overline{\Theta}), (a \longrightarrow P)(T_2, \overline{\Theta})) && \square \end{aligned}$$
- $N_X(P \setminus A) \widehat{=} \{(U, V) \mid (U, V') \in N_X(P) \wedge V' \cap A = \emptyset \wedge V' \subseteq V\}$.

  Proof. Suppose $(U, V) \in N_X(P \setminus A)$. By construction, there exists V', such that $(U, V') \in N_X(P)$, $V' \subseteq V$ and $V' \cap A = \emptyset$. We will prove that for any $T_1, T_2 \in \mathcal{T}^{\Downarrow}$, $d_U(T_1, T_2) \geq d_V(P(T_1, \overline{\Theta}) \setminus A, P(T_2, \overline{\Theta}) \setminus A)$.
  $$\begin{aligned} d_U(T_1, T_2) \ &\geq d_{V'}(P(T_1, \overline{\Theta}), P(T_2, \overline{\Theta})) && \text{// induction hypothesis} \\ &\geq d_{V'}(P(T_1, \overline{\Theta}) \setminus A, P(T_2, \overline{\Theta}) \setminus A) && \text{// } V' \cap A = \emptyset, \text{ Lemma 4.5} \\ &\geq d_V(P(T_1, \overline{\Theta}) \setminus A, P(T_2, \overline{\Theta}) \setminus A) && \text{// } V' \subseteq V, U \mapsto d_U \text{ antitone} \qquad \square \end{aligned}$$
- $N_X(P_1 \oplus P_2) \widehat{=} N_X(P_1) \cap N_X(P_2) = \{(U_1 \cap U_2, V_1 \cup V_2) \mid (U_i, V_i) \in N_X(P_i)\}$ for every $\oplus \in \{\sqcap, \square, \mathbin{;}, \underset{A}{\|}\}$.

  Proof. Suppose $(U, V) \in N_X(P_1 \oplus P_2)$. By construction, there exist $(U_1, V_1) \in N_X(P_1)$ and $(U_2, V_2) \in N_X(P_2)$, such that $U = U_1 \cap U_2$ and $V = V_1 \cup V_2$. Therefore, $(U, V) \in N_X(P_1)$, $(U, V) \in N_X(P_2)$ (antitoneness).

  $d_V((P_1 \oplus P_2)(T_1, \overline{\Theta}), (P_1 \oplus P_2)(T_2, \overline{\Theta}))$
  $= d_V(P_1(T_1, \overline{\Theta}) \oplus P_2(T_1, \overline{\Theta}), P_1(T_2, \overline{\Theta}) \oplus P_2(T_2, \overline{\Theta}))$
  // ultrametric inequality
  $\leq max\{\ d_V(P_1(T_1, \overline{\Theta}) \oplus P_2(T_1, \overline{\Theta}), P_1(T_2, \overline{\Theta}) \oplus P_2(T_1, \overline{\Theta})),$
  $\qquad\qquad d_V(P_1(T_2, \overline{\Theta}) \oplus P_2(T_1, \overline{\Theta}), P_1(T_2, \overline{\Theta}) \oplus P_2(T_2, \overline{\Theta}))\}$
  // Lemma 4.3
  $\leq max\{\ d_V(P_1(T_1, \overline{\Theta}), P_1(T_2, \overline{\Theta}))\ \ // \leq d_U(T_1, T_2) \text{ by induction hypothesis for } P_1$
  $\qquad\qquad d_V(P_2(T_1, \overline{\Theta}), P_2(T_2, \overline{\Theta}))\}\ // \leq d_U(T_1, T_2) \text{ by induction hypothesis for } P_2$
  $\leq d_U(T_1, T_2)$ □

- $\mathsf{N}_X(P[R]) \mathrel{\hat{=}} \{(U,V) \mid (U,V') \in \mathsf{N}_X(P) \wedge R(V') \subseteq V\}$.

  Proof. Suppose $(U,V) \in \mathsf{N}_X(P[R])$. By construction, there exists V', such that $(U,V') \in \mathsf{N}_X(P)$ and $R(V') \subseteq V$.

  $$
  \begin{aligned}
  d_U(T_1,T_2) \;&\geq\; d_{V'}(P(T_1,\overline{\Theta}), P(T_2,\overline{\Theta})) && \text{// induction hypothesis} \\
  &\geq\; d_{R(V')}(P(T_1,\overline{\Theta})[R], P(T_2,\overline{\Theta})[R]) && \text{// Lemma 4.6} \\
  &\geq\; d_V(P(T_1,\overline{\Theta})[R], P(T_2,\overline{\Theta})[R]) && \text{// } R(V') \subseteq V,\ U \mapsto d_U \text{ antitone}
  \end{aligned}
  $$
  $\square$

- $\mathsf{N}_X(X) \mathrel{\hat{=}} \{(U,V) \mid U \subseteq V\}$.

  Proof.
  $$
  \begin{aligned}
  d_U(T_1,T_2) \;&\geq\; d_V(T_1,T_2) && \text{// } U \subseteq V,\ U \mapsto d_U \text{ antitone} \\
  &=\; d_V((X)(T_1,\overline{\Theta}), (X)(T_2,\overline{\Theta}))
  \end{aligned}
  $$
  $\square$

- $\mathsf{N}_X(\mu Y \boldsymbol{.} P) \mathrel{\hat{=}} \{(U,V) \mid (U',V') \in \mathsf{N}_X(P) \wedge (V',V') \in \mathsf{N}_Y(P) \wedge U \subseteq U' \wedge V' \subseteq V\}$ if $Y \neq X$.

  Proof. Suppose $(U,V) \in \mathsf{N}_X(\mu Y \boldsymbol{.} P)$ for $X \neq Y$ and X,Y free in $P(X,Y,Z_1,\ldots,Z_n)$. By construction, there exist $U_X, V_X \subseteq \Sigma$ such that:
  (1) $(U_X, V_X) \in \mathsf{N}_X(P)$
  (2) $U \subseteq U_X,\ V_X \subseteq V$
  (3) $(V_X, V_X) \in \mathsf{N}_Y(P)$
  Therefore, by induction hypothesis, for all $T_1, T_2, \xi, \overline{\Theta} \in \mathcal{T}^{\Downarrow}$, we have:

  $$d_{U_X}(T_1,T_2) \geq d_{V_X}(P(T_1,\xi,\overline{\Theta}), P(T_2,\xi,\overline{\Theta})) \tag{D.1}$$

  $$d_{V_X}(T_1,T_2) \geq d_{V_X}(P(\xi,T_1,\overline{\Theta}), P(\xi,T_2,\overline{\Theta})) \tag{D.2}$$

  $$
  \begin{aligned}
  d_U(T_1,T_2) \;&\geq\; d_{U_X}(T_1,T_2) && \text{// } U \subseteq U_X, \text{ antitoneness} \\
  &\geq\; d_{V_X}(P(T_1,\xi,\overline{\Theta}), P(T_2,\xi,\overline{\Theta})) && \text{// from D.1}
  \end{aligned}
  $$

  Let $P_1(Y) = P(T_1,Y,\overline{\Theta})$, $P_2(Y) = P(T_2,Y,\overline{\Theta})$. $P_1(Y)$ and $P_2(Y)$ are continuous over $\sqsubseteq$. Therefore, there exist $\mu Y \boldsymbol{.} P_1(Y) = \bigcap_{n=0}^{\infty} P_1^n = P_1^*$ and $\mu Y \boldsymbol{.} P_2(Y) = \bigcap_{n=0}^{\infty} P_2^n = P_2^*$, where for $i = 1, 2$, $P_i^0 = \perp = DIV$, $P_i^{n+1} = P_i(P_i^n)$.
  We will prove by induction that

  $$d_{V_X}(P_1^n, P_2^n) \leq d_{U_X}(T_1,T_2) \text{ for } n \geq 1. \tag{D.3}$$

  - Let $n = 1$.
  $$
  \begin{aligned}
  d_{U_X}(T_1,T_2) \;&\geq\; d_{V_X}(P(T_1,DIV,\overline{\Theta}), P(T_2,DIV,\overline{\Theta})) && \text{// from D.1} \\
  &=\; d_{V_X}(P_1^1, P_2^1)
  \end{aligned}
  $$
  - Suppose $d_{V_X}(P_1^n, P_2^n) \leq d_{U_X}(T_1,T_2)$.
  $$
  \begin{aligned}
  d_{V_X}(P_1^{n+1}, P_2^{n+1}) \;&=\; d_{V_X}(P(T_1,P_1^n,\overline{\Theta}), P(T_2,P_2^n,\overline{\Theta})) \\
  & \quad \text{// ultrametric inequality} \\
  &\leq\; max\{d_{V_X}(P(T_1,P_1^n,\overline{\Theta}), P(T_2,P_1^n,\overline{\Theta})), \\
  & \qquad\quad d_{V_X}(P(T_2,P_1^n,\overline{\Theta}), P(T_2,P_2^n,\overline{\Theta}))\} \\
  &\leq\; max\{d_{U_X}(T_1,T_2), && \text{// from D.1} \\
  & \qquad\quad d_{V_X}(P_1^n, P_2^n)\} && \text{// from D.2} \\
  &\leq\; max\{d_{U_X}(T_1,T_2), \\
  & \qquad\quad d_{U_X}(T_1,T_2\} && \text{// from D.3, local i.h.} \\
  &\leq\; d_{U_X}(T_1,T_2)
  \end{aligned}
  $$

Let $d_{U_X}(T_1, T_2) = 2^{-k}$ for some $k \in \mathbb{N}$. Now suppose for the sake of contradiction that $d_{V_X}(P_1^*, P_2^*) > d_{U_X}(T_1, T_2) = 2^{-k}$ and let, without loss of generality, P_1^* and P_2^* differ on the sets of their divergences. Therefore, again without loss of generality, there exists $s \in \mathsf{divergences}(P_1^*)$ such that $s \notin \mathsf{divergences}(P_2^*)$ and $\mathsf{length}_{V_X}(s) \leq k$. Then, since $P_i^* = \bigcap_{n=0}^{\infty} P_i^n$, $s \in P_1^n$ for all $n \in \mathbb{N}$, but there exists $l \in \mathbb{N}$ such that $s \notin P_2^l$. But then $d_{V_X}(P_1^l, P_2^l) > 2^{-k} = d_{U_X}(T_1, T_2)$, which is a contradiction with D.3. Therefore, $d_{V_X}(P_1^*, P_2^*) \leq d_{U_X}(T_1, T_2)$. Then, since $U \subseteq U_X$ and $V_X \subseteq V$, by antitoneness, $d_V((\mu Y \boldsymbol{.} P)(T_1, \overline{\Theta}), (\mu Y \boldsymbol{.} P)(T_2, \overline{\Theta})) = d_V(P_1^*, P_2^*) \leq d_U(T_1, T_2)$. $\qquad \square$

$\square$

Proposition D.1. *Let $P(X, Y_1, \ldots, Y_n) = P(X, \overline{Y})$ be a CSP term whose free variables are contained within the set $\{X, Y_1, \ldots, Y_n\}$. Let $\mathsf{G} : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma))$, $\mathsf{C}_X : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ and $\mathsf{F} : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ be defined recursively on the structure of P as shown in Figures 7, 8 and 9, respectively. Then:*

(1) *If $V \in \mathsf{G}(P)$, then, with any processes substituted for the free variables of P (and in particular DIV), P must communicate an event from V before it can do a $\checkmark$.*

(2) *If $(U, V) \in \mathsf{C}_X(P)$, then for all processes $T_1, T_2, \Theta_1, \ldots, \Theta_n \in \mathcal{T}^{\Downarrow}$, $d_V(P(T_1, \overline{\Theta}), P(T_2, \overline{\Theta})) \leq \frac{1}{2} d_U(T_1, T_2)$.*

(3) *If $(U, V) \in \mathsf{F}(P)$, then, for any collection of U-fair livelock-free processes $\theta_0, \ldots, \theta_n \in \mathcal{T}^{\Downarrow}$, the process $P(\theta_0, \ldots, \theta_n)$ is livelock-free and V-fair.*

Proof. We carry out the proof by induction on the structure of P. For clarity, we prove (1), (2) and (3) one by one, in Propositions 5.3, 5.4 and 5.5, respectively. In each of these propositions, our induction hypothesis is that at any point all (1), (2) and (3) hold for any subterm of P. $\qquad \square$

Proposition 5.3. *Let $P(X, Y_1, \ldots, Y_n) = P(X, \overline{Y})$ be a CSP term whose free variables are contained within the set $\{X, Y_1, \ldots, Y_n\}$. Let $\mathsf{G} : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma))$ be defined recursively on the structure of P as shown in Figure 7. If $V \in \mathsf{G}(P)$, then, with any processes substituted for the free variables of P (and in particular DIV), P must communicate an event from V before it can do a $\checkmark$.*

Proof. Structural induction on P. We will write $\widehat{P}$ to denote the result of substituting all free variables in P with the most general process $\bot = DIV$. For each process ξ, $DIV \sqsubseteq \xi$. Therefore, by monotonicity of CSP operators [21], for any process term $C(X)$, $C(DIV) \sqsubseteq C(\xi)$.

- $\mathsf{G}(STOP) = \mathcal{P}(\Sigma)$.

  Proof. STOP cannot terminate and, therefore, the property holds vacuously. $\qquad \square$

- $\mathsf{G}(a \longrightarrow P) = \mathsf{G}(P) \cup \{V \mid a \in V\}$.

  Proof. Let $V \in \mathsf{G}(a \longrightarrow P)$, $t = s ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\bot(\widehat{a \longrightarrow P}) = \mathsf{traces}_\bot(a \longrightarrow \widehat{P})$. Therefore, $t = \langle a \rangle ^\frown r ^\frown \langle \checkmark \rangle$ for some $r \in \Sigma^*$ such that $r ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\bot(\widehat{P})$ and $s = \langle a \rangle ^\frown r$. Since $V \in \mathsf{G}(a \longrightarrow P)$, by construction, $V \in \mathsf{G}(P)$ or $a \in V$.
  - Suppose $V \in \mathsf{G}(P)$. Then, by induction hypothesis, $r ^\frown \langle \checkmark \rangle$ contains an event from V and, therefore, so do s and t.
  - Suppose $a \in V$. Then $t = \langle a \rangle ^\frown r ^\frown \langle \checkmark \rangle$ contains the event $a \in V$ before $\checkmark$.
  $\qquad \square$

- $\mathsf{G}(SKIP) = \emptyset$. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$
- $\mathsf{G}(P_1 \oplus P_2) = \mathsf{G}(P_1) \cap \mathsf{G}(P_2)$ for $\oplus \in \{\sqcap, \square\}$.

  Proof. Let $V \in \mathsf{G}(P_1 \oplus P_2)$ and $t = s ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(\widehat{P_1 \oplus P_2}) = \mathsf{traces}_\perp(\widehat{P_1}) \cup \mathsf{traces}_\perp(\widehat{P_2})$. Therefore, $t \in \mathsf{traces}_\perp(\widehat{P_1})$ or $t \in \mathsf{traces}_\perp(\widehat{P_2})$. Let, without loss of generality, $t \in \mathsf{traces}_\perp(\widehat{P_1})$. By construction, $V \in \mathsf{G}(P_1)$. Then, by induction hypothesis, s contains an event from V. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

- $\mathsf{G}(P_1 \,\fatsemi\, P_2) = \begin{cases} \mathsf{G}(P_1) \cup \mathsf{G}(P_2) & \text{if } P_1 \text{ is closed and } \mathsf{F}(P_1) \neq \emptyset \\ \mathsf{G}(P_1) & \text{otherwise} \end{cases}$

  Proof. Let $V \in \mathsf{G}(P_1 \,\fatsemi\, P_2)$ and $t = s ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(\widehat{P_1 \,\fatsemi\, P_2})$.

  Let first P_1 be closed and $\mathsf{F}(P_1) \neq \emptyset$. Then, by Proposition D.1 (3), P_1 is livelock-free and, therefore, $\mathsf{divergences}(P_1) = \emptyset$. Therefore, $t = t_1 ^\frown t_2 ^\frown \langle \checkmark \rangle$ with $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(\widehat{P_1})$ and $t_2 ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(\widehat{P_2})$. In this case by construction $V \in \mathsf{G}(P_1) \cup \mathsf{G}(P_2)$. Let without loss of generality $V \in \mathsf{G}(P_1)$. Then, by induction hypothesis, t_1 contains an event from V and therefore so does t.

  Let now P_1 be open or $\mathsf{F}(P_1) = \emptyset$. Then by construction $V \in \mathsf{G}(P_1)$. We consider the two possibilities for t.
  - $t = t_1 ^\frown t_2 ^\frown \langle \checkmark \rangle$ with $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(\widehat{P_1})$ and $t_2 ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(\widehat{P_2})$. Since $V \in \mathsf{G}(P_1)$, by induction hypothesis, t_1 contains an event from V and therefore so does t.
  - $t \in \mathsf{divergences}(P_1)$ and, therefore, $t \in \mathsf{traces}_\perp(P_1)$. Then again, by induction hypothesis, t_1 contains an event from V and therefore so does t. $\qquad\qquad\square$

- $\mathsf{G}(P_1 \underset{A}{\|} P_2) = \begin{cases} \mathsf{G}(P_1) \cup \mathsf{G}(P_2) & \text{if, for } i = 1, 2, \, P_i \text{ is closed and } \mathsf{F}(P_i) \neq \emptyset \\ \mathsf{G}(P_1) \cap \mathsf{G}(P_2) & \text{otherwise} \end{cases}$

  Proof. Let $V \in \mathsf{G}(P_1 \underset{A}{\|} P_2)$ and $t = s ^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(\widehat{P_1 \underset{A}{\|} P_2})$.

  Let first both P_1 and P_2 be closed, $\mathsf{F}(P_1) \neq \emptyset$ and $\mathsf{F}(P_2) \neq \emptyset$. Then, by Proposition D.1 (3), P_1 and P_2 are livelock-free and, therefore, $P_1 \underset{A}{\|} P_2$ is livelock-free. Therefore, $\mathsf{divergences}(\widehat{P_1 \underset{A}{\|} P_2}) = \emptyset$ and $\mathsf{traces}_\perp(\widehat{P_1 \underset{A}{\|} P_2}) = \mathsf{traces}(\widehat{P_1 \underset{A}{\|} P_2})$. By construction, $V \in \mathsf{G}(P_1) \cup \mathsf{G}(P_2)$. Let without loss of generality $V \in \mathsf{G}(P_2)$. Since $t \in \mathsf{traces}(\widehat{P_1 \underset{A}{\|} P_2})$, then, due to distributed termination, there exist t_1, t_2, such that $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(\widehat{P_1})$, $t_2 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(\widehat{P_2})$ and $t \in t_1 \underset{A}{\|} t_1$. By induction hypothesis, t_2 contains an event from V and therefore so does t.

  Otherwise, $t \in \mathsf{traces}(\widehat{P_1 \underset{A}{\|} P_2})$ or $t \in \mathsf{divergences}(\widehat{P_1 \underset{A}{\|} P_2})$. We consider both alternatives. By construction, $V \in \mathsf{G}(P_1) \cap \mathsf{G}(P_2)$, i.e., $V \in \mathsf{G}(P_1)$ and $V \in \mathsf{G}(P_2)$.
  - $t = s ^\frown \langle \checkmark \rangle \in \mathsf{traces}(\widehat{P_1 \underset{A}{\|} P_2})$. Then, due to distributed termination, there exist t_1, t_2, such that $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}((\widehat{P_1})$, $t_2 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(\widehat{P_2})$ and $t \in t_1 \underset{A}{\|} t_1$. By induction hypothesis, both t_1 and t_2 contain an event from V and therefore so does t.

– $t = s^\frown \langle \checkmark \rangle \in \mathsf{divergences}(\widehat{P_1 \underset{A}{\parallel} P_2})$. Therefore, there exist s_1, s_2, t_1, t_2, such that $t_1 \in \mathsf{traces}_\perp(\widehat{P_1})$, $t_2 \in \mathsf{traces}_\perp(\widehat{P_2})$, $s_1 \in (t_1 \underset{A}{\parallel} t_2) \cap \Sigma^*$, $t = s_1 {}^\frown s_2 {}^\frown \langle \checkmark \rangle$ and, $t_1 \in \mathsf{divergences}(\widehat{P_1})$ or $t_2 \in \mathsf{divergences}(\widehat{P_2})$. Let without loss of generality $t_1 \in \mathsf{divergences}(\widehat{P_1})$. Then $t_1 \in \Sigma^*$ and therefore, $t_1 {}^\frown \langle \checkmark \rangle \in \mathsf{divergences}(\widehat{P_1})$. Since $V \in \mathsf{G}(P_1)$, by induction hypothesis t_1 contains an event from V and therefore so does t. $\qquad\square$

- $\mathsf{G}(P[R]) = \{V \mid V' \in \mathsf{G}(P) \wedge R(V') \subseteq V\}$.

  Proof. Let $V \in \mathsf{G}(P[R])$. Then, by construction, there exists $V' \in \mathsf{G}(P)$ with $R(V') \subseteq V$. Let $t = s^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(\widehat{P[R]})$. Then, $t \in \mathsf{divergences}(\widehat{P[R]})$ or $t \in \mathsf{traces}(\widehat{P[R]})$. We consider both alternatives.
  – Suppose $t = s^\frown \langle \checkmark \rangle \in \mathsf{divergences}(\widehat{P[R]})$. Therefore, there exist $s_1, s_2, r_1 \in \Sigma^*$, such that $r_1 \in \mathsf{divergences}(P)$, $r_1 \ R \ s_1$ and $t = s_1 {}^\frown s_2 {}^\frown \langle \checkmark \rangle$. As $r_1 \in \mathsf{divergences}(P)$, by Axiom 4, $r_1 {}^\frown \langle \checkmark \rangle \in \mathsf{divergences}(P)$. Then, by induction hypothesis for P, r_1 contains an event from V'. Since $r_1 \ R \ s_1$, s_1 contains an event from $R(V') \subseteq V$. Therefore, since $t = s_1 {}^\frown s_2 {}^\frown \langle \checkmark \rangle$, t contains an event from V.
  – Suppose $t = s^\frown \langle \checkmark \rangle \in \mathsf{traces}(\widehat{P[R]})$. Therefore, there exist $t', s' \in \mathsf{traces}(\widehat{P})$, such that $t' = s'^\frown \langle \checkmark \rangle$ and $s' \ R \ s$. By induction hypothesis for P and t', s' contains an event from V'. Since $s' \ R \ s$, s contains an event from $R(V') \subseteq V$ and, hence, t contains an event from V.
  $\qquad\square$

- $\mathsf{G}(P \setminus A) = \begin{cases} \{V \mid V' \in \mathsf{G}(P) \wedge V' \cap A = \emptyset \wedge V' \subseteq V\} & \text{if } P \text{ is closed and} \\ & (\emptyset, \Sigma - A) \in \mathsf{F}(P) \\ \emptyset & \text{otherwise} \end{cases}$

  Proof. Let $V \in \mathsf{G}(P \setminus A)$ and let, furthermore, P be closed and $(\emptyset, \Sigma - A) \in \mathsf{F}(P)$. Then P does not have free process variables and by Proposition D.1 (3) we can conclude the following:
  (1) P is livelock-free, i.e., $\mathsf{divergences}(P) = \emptyset$ and $\mathsf{traces}_\perp(P) = \mathsf{traces}(P)$.
  (2) Any infinite trace u of P contains infinitely many events from $\Sigma \backslash A$ and therefore $u \restriction (\Sigma \backslash A)$ is infinite.
  Let $t = s^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(\widehat{P} \setminus A) = \mathsf{traces}_\perp(P \setminus A)$. Then $t \in \mathsf{divergences}(P \setminus A)$ or $t \in \mathsf{traces}(P \setminus A)$. We consider both alternatives.
  – Let $t = s^\frown \langle \checkmark \rangle \in \mathsf{divergences}(P \setminus A)$. As from (?1) $\mathsf{divergences}(P) = \emptyset$ (i.e., t cannot arise from a divergence of P), by definition there exists $u \in \mathsf{traces}^\omega(P)$ such that $s_1 = u \restriction (\Sigma \backslash A)$ is finite and $t = s_1 {}^\frown s_2 {}^\frown \langle \checkmark \rangle$. However, by (?2), $u \restriction (\Sigma \backslash A)$ cannot be finite for any infinite trace u of P. Due to the contradiction, this case is not possible.
  – Therefore, $t = s^\frown \langle \checkmark \rangle \in \mathsf{traces}(P \setminus A)$. Therefore, there exists $t' = s'^\frown \langle \checkmark \rangle \in \mathsf{traces}(P)$, such that $s = s' \restriction (\Sigma \backslash A)$. Since $V \in \mathsf{G}(P \setminus A)$, by construction there exists $V' \in \mathsf{G}(P)$ with $V' \subseteq V$ and $V' \cap A = \emptyset$. By induction hypothesis for t' and P, s' contains an event from $V' \subseteq V$. But $V' \cap A = \emptyset$. Hence, $s = s' \restriction (\Sigma \backslash A)$ contains an event from V' and therefore from V. $\qquad\square$

- $\mathsf{G}(X) = \emptyset$. $\qquad\square$

- $\mathsf{G}(\mu X \centerdot P) = \mathsf{G}(P)$.

  Proof. Let $V \in \mathsf{G}(\mu X \centerdot P)$ and $t = s^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(\widehat{\mu X \centerdot P})$. $\widehat{\mu X \centerdot P} = (\mu X \centerdot P)(\overline{DIV}) = P^* = \bigcap_{i=0}^\infty P^n$, where $P^0 = DIV$, $P^{n+1} = P(P^n, \overline{DIV})$. Since $t \in \mathsf{traces}_\perp(P^*)$, $t \in \mathsf{traces}_\perp(P^n)$, for each $n \in \mathbb{N}$. Therefore,

  $$t = s^\frown \langle \checkmark \rangle \in \mathsf{traces}_\perp(P^1) = \mathsf{traces}_\perp(P(DIV, \overline{DIV})) = \mathsf{traces}_\perp(\widehat{P}).$$

  By construction, $V \in \mathsf{G}(P)$. Therefore, by induction hypothesis for P and t, s contains an event from V. $\qquad\square$

  $\square$

Proposition 5.4. *Let $P(X, Y_1, \ldots, Y_n) = P(X, \overline{Y})$ be a CSP term whose free variables are contained within the set $\{X, Y_1, \ldots, Y_n\}$. Let $\mathsf{C}_X : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ be defined recursively on the structure of P as shown in Figure 8. If $(U, V) \in \mathsf{C}_X(P)$, then for all processes $T_1, T_2, \Theta_1, \ldots, \Theta_n \in \mathcal{T}^{\Downarrow}$, $d_V(P(T_1, \overline{\Theta}), P(T_2, \overline{\Theta})) \leq \frac{1}{2} d_U(T_1, T_2)$.*

Proof. Structural induction on P. Let us take arbitrary $T_1, T_2, \Theta_1, \ldots, \Theta_n \in \mathcal{T}^{\Downarrow}$.

- $\mathsf{C}_X(P) \widehat{=} \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$ whenever X is not free in P.

  Proof. Let $(U, V) \in \mathsf{C}_X(P)$. Then

  $$\frac{1}{2} d_U(T_1, T_2) \geq 0 = d_V(P(T_1, \overline{\Theta}), P(T_2, \overline{\Theta})) = d_V(P(\overline{\Theta}), P(\overline{\Theta})). \qquad\square$$

- $\mathsf{C}_X(a \longrightarrow P) \widehat{=} \mathsf{C}_X(P) \cup \{(U, V) \in \mathsf{N}_X(P) \mid a \in V\}$.

  Proof. Suppose $(U, V) \in \mathsf{C}_X(a \longrightarrow P)$. By construction, $(U, V) \in \mathsf{C}_X(P)$ or, $(U, V) \in \mathsf{N}_X(P)$ and $a \in V$. We consider both cases.
  - Suppose $(U, V) \in \mathsf{C}_X(P)$. Then:
    $$\begin{aligned} \tfrac{1}{2} d_U(T_1, T_2) &\geq d_V(P(T_1, \overline{\Theta}), P(T_2, \overline{\Theta})) &&\text{// induction hypothesis} \\ &\geq d_V(a \longrightarrow P(T_1, \overline{\Theta}), a \longrightarrow P(T_2, \overline{\Theta})) &&\text{// Lemma 4.4} \end{aligned}$$
  - Suppose $(U, V) \in \mathsf{N}_X(P)$ and $a \in V$.
    $$\begin{aligned} \tfrac{1}{2} d_U(T_1, T_2) &\geq \tfrac{1}{2} d_V(P(T_1, \overline{\Theta}), P(T_2, \overline{\Theta})) &&\text{// } (U, V) \in \mathsf{N}_X(P), \\ &&&\text{// Proposition 5.2} \\ &= d_V(a \longrightarrow P(T_1, \overline{\Theta}), a \longrightarrow P(T_2, \overline{\Theta})) &&\text{// } a \in V \qquad\square \end{aligned}$$

- $\mathsf{C}_X(P_1 \oplus P_2) \widehat{=} \mathsf{C}_X(P_1) \cap \mathsf{C}_X(P_2) = \{(U_1 \cap U_2, V_1 \cup V_2) \mid (U_i, V_i) \in \mathsf{C}_X(P_i)\}$ for $\oplus \in \{\sqcap, \square, \underset{A}{\|}\}$.

  Proof. Suppose $(U, V) \in \mathsf{C}_X(P_1 \oplus P_2)$. By construction, there exist $(U_1, V_1) \in \mathsf{C}_X(P_1)$ and $(U_2, V_2) \in \mathsf{C}_X(P_2)$, such that $U = U_1 \cap U_2$ and $V = V_1 \cup V_2$. Therefore, $(U, V) \in \mathsf{C}_X(P_1)$, $(U, V) \in \mathsf{C}_X(P_2)$ (antitoneness). Then:
  $d_V((P_1 \oplus P_2)(T_1, \overline{\Theta}), (P_1 \oplus P_2)(T_2, \overline{\Theta}))$
  $= d_V(P_1(T_1, \overline{\Theta}) \oplus P_2(T_1, \overline{\Theta}), P_1(T_2, \overline{\Theta}) \oplus P_2(T_2, \overline{\Theta}))$
  // ultrametric inequality
  $\leq max\{\; d_V(P_1(T_1, \overline{\Theta}) \oplus P_2(T_1, \overline{\Theta}), P_1(T_2, \overline{\Theta}) \oplus P_2(T_1, \overline{\Theta})),$
  $\qquad\qquad d_V(P_1(T_2, \overline{\Theta}) \oplus P_2(T_1, \overline{\Theta}), P_1(T_2, \overline{\Theta}) \oplus P_2(T_2, \overline{\Theta}))\}$
  // Lemma 4.3

$$\leq max\{ \quad d_V(P_1(T_1,\overline{\Theta}),P_1(T_2,\overline{\Theta})) \quad // \leq \tfrac{1}{2}d_U(T_1,T_2), \text{ induction hypothesis for } P_1$$
$$d_V(P_2(T_1,\overline{\Theta}),P_2(T_2,\overline{\Theta}))\} \quad // \leq \tfrac{1}{2}d_U(T_1,T_2), \text{ induction hypothesis for } P_2$$
$$\leq \tfrac{1}{2}d_U(T_1,T_2) \qquad\qquad \square$$

- $\mathsf{C}_X(P_1 \,\mathbin{\fatsemi}\, P_2) \mathrel{\widehat{=}} \mathsf{C}_X(P_1) \cap (\mathsf{C}_X(P_2) \cup \{(U,V) \in \mathsf{N}_X(P_2) \mid V \in \mathsf{G}(P_1)\}).$

  Proof. Suppose $(U,V) \in \mathsf{C}_X(P_1 \,\mathbin{\fatsemi}\, P_2)$. By construction, $(U,V) \in \mathsf{C}_X(P_1 \,\mathbin{\fatsemi}\, P_2)$ yields 2 possibilities:
  - $(U,V) \in \mathsf{C}_X(P_1) \cap \mathsf{C}_X(P_2)$. The proof is the same as the proof for $\sqcap, \square$ and $\underset{A}{\parallel}$.
  - $(U,V) \in \mathsf{C}_X(P_1) \cap \{(U,V) \in \mathsf{N}_X(P_2) \mid V \in \mathsf{G}(P_1)\}$. Again, using the ultrametric inequality:

  $$d_V(P_1(T_1,\overline{\Theta}) \,\mathbin{\fatsemi}\, P_2(T_1,\overline{\Theta}), P_1(T_2,\overline{\Theta}) \,\mathbin{\fatsemi}\, P_2(T_2,\overline{\Theta}))$$
  $$\leq max\{ \quad d_V(P_1(T_1,\overline{\Theta}) \,\mathbin{\fatsemi}\, P_2(T_1,\overline{\Theta}), P_1(T_2,\overline{\Theta}) \,\mathbin{\fatsemi}\, P_2(T_1,\overline{\Theta})), \quad // \text{ Lemma 4.3}$$
  $$d_V(P_1(T_2,\overline{\Theta}) \,\mathbin{\fatsemi}\, P_2(T_1,\overline{\Theta}), P_1(T_2,\overline{\Theta}) \,\mathbin{\fatsemi}\, P_2(T_2,\overline{\Theta}))\}$$
  $$\leq max\{ \quad d_V(P_1(T_1,\overline{\Theta}),P_1(T_2,\overline{\Theta})), \quad // \leq \tfrac{1}{2}d_U(T_1,T_2), \text{ induction hypothesis}$$
  $$d_V(P_2(T_1,\overline{\Theta}),P_2(T_2,\overline{\Theta}))\} \quad // \leq \tfrac{1}{2}d_U(T_1,T_2), \text{ Prop. D.1 (1) and 4.7}$$
  $$\leq \tfrac{1}{2}d_U(T_1,T_2)$$

  $\square$

- $\mathsf{C}_X(P \setminus A) \mathrel{\widehat{=}} \{(U,V) \mid (U,V') \in \mathsf{C}_X(P) \wedge V' \cap A = \emptyset \wedge V' \subseteq V\}.$

  Proof. Suppose $(U,V) \in \mathsf{C}_X(P \backslash A)$. By construction, there exists V', such that $(U,V') \in \mathsf{C}_X(P)$, $V' \subseteq V$ and $V' \cap A = \emptyset$.

  $$\tfrac{1}{2}d_U(T_1,T_2) \geq d_{V'}(P(T_1,\overline{\Theta}),P(T_2,\overline{\Theta})) \qquad // \text{ induction hypothesis}$$
  $$\geq d_{V'}(P(T_1,\overline{\Theta}) \setminus A, P(T_2,\overline{\Theta}) \setminus A) \quad // V' \cap A = \emptyset, \text{ Lemma 4.5}$$
  $$\geq d_V(P(T_1,\overline{\Theta}) \setminus A, P(T_2,\overline{\Theta}) \setminus A) \quad // V' \subseteq V, U \mapsto d_U \text{ antitone}$$

  $\square$

- $\mathsf{C}_X(P[R]) \mathrel{\widehat{=}} \{(U,V) \mid (U,V') \in \mathsf{C}_X(P) \wedge R(V') \subseteq V\}.$

  Proof. Suppose $(U,V) \in \mathsf{C}_X(P[R])$. By construction, there exists V', such that $(U,V') \in \mathsf{C}_X(P)$ and $R(V') \subseteq V$.

  $$\tfrac{1}{2}d_U(T_1,T_2) \geq d_{V'}(P(T_1,\overline{\Theta}),P(T_2,\overline{\Theta})) \qquad // \text{ induction hypothesis}$$
  $$\geq d_{R(V')}(P(T_1,\overline{\Theta})[R], P(T_2,\overline{\Theta})[R]) \quad // \text{ Lemma 4.6}$$
  $$\geq d_V(P(T_1,\overline{\Theta})[R], P(T_2,\overline{\Theta})[R]) \qquad // R(V') \subseteq V, U \mapsto d_U \text{ antitone}$$

  $\square$

- $\mathsf{C}_X(X) \mathrel{\widehat{=}} \emptyset.$ $\qquad\qquad \square$

- $\mathsf{C}_X(\mu Y \centerdot P) \mathrel{\widehat{=}} \{(U,V) \mid (U',V') \in \mathsf{C}_X(P) \wedge (V',V') \in \mathsf{N}_Y(P) \wedge U \subseteq U' \wedge V' \subseteq V\}$ if $Y \neq X$

  Proof. Suppose $(U,V) \in \mathsf{C}_X(\mu Y \centerdot P)$ for $X \neq Y$ and X, Y free in $P(X,Y,Z_1,\ldots,Z_n)$. Then, by construction, there exist $U', V' \subseteq \Sigma$ such that:
  (a) $(U',V') \in \mathsf{C}_X(P)$
  (b) $U \subseteq U', V' \subseteq V$
  (c) $(V',V') \in \mathsf{N}_Y(P)$
  Since $(U',V') \in \mathsf{C}_X(P)$, by induction hypothesis, for all $T_1, T_2, \xi, \overline{\Theta} \in \mathcal{T}^{\Downarrow}$ we have:

  $$\frac{1}{2}d_{U'}(T_1,T_2) \geq d_{V'}(P(T_1,\xi,\overline{\Theta}), P(T_2,\xi,\overline{\Theta})) \tag{D.4}$$

Since $(V', V') \in \mathsf{N}_Y(P)$, from Proposition 5.2:

$$d_{V'}(T_1, T_2) \geq d_{V'}(P(\xi, T_1, \overline{\Theta}), P(\xi, T_2, \overline{\Theta})) \tag{D.5}$$

Let $P_1(Y) = P(T_1, Y, \overline{\Theta})$, $P_2(Y) = P(T_2, Y, \overline{\Theta})$. Then, $(\mu Y \centerdot P)(T_1, \overline{\theta}) = P_1^* = \bigcap_{i=0}^{\infty} P_1^n$ and $(\mu Y \centerdot P)(T_2, \overline{\theta}) = P_2^* = \bigcap_{i=0}^{\infty} P_2^n$, where for $i = 1, 2$, $P_i^0 = \bot = DIV$, $P_i^{n+1} = P(T_i, P_i^n, \overline{\theta}) = P_i(P_i^n)$.

We will prove by induction that for $n \geq 1$:

$$d_{V'}(P_1^n, P_2^n) \leq \frac{1}{2} d_{U'}(T_1, T_2) \tag{D.6}$$

- n=1.
  $\frac{1}{2} d_{U'}(T_1, T_2) \geq d_{V'}(P(T_1, DIV, \overline{\Theta}), P(T_2, DIV, \overline{\Theta}))$ // from (D.4)
  $= d_{V'}(P_1^1, P_2^1)$

- Suppose $d_{V'}(P_1^n, P_2^n) \leq \frac{1}{2} d_{U'}(T_1, T_2)$.
  $d_{V'}(P_1^{n+1}, P_2^{n+1}) = d_{V'}(P(T_1, P_1^n, \overline{\Theta}), P(T_2, P_2^n, \overline{\Theta}))$
  $\leq max\{d_{V'}(P(T_1, P_1^n, \overline{\Theta}), P(T_2, P_1^n, \overline{\Theta})),$
  $\qquad\qquad d_{V'}(P(T_2, P_1^n, \overline{\Theta}), P(T_2, P_2^n, \overline{\Theta}))\}$
  $\leq max\{\frac{1}{2} d_{U'}(T_1, T_2),$ // from (D.4)
  $\qquad\qquad d_{V'}(P_1^n, P_2^n)\}$ // from (D.5)
  $\leq max\{\frac{1}{2} d_{U'}(T_1, T_2),$
  $\qquad\qquad \frac{1}{2} d_{U'}(T_1, T_2)\}$ // from (D.6)
  $\leq \frac{1}{2} d_{U'}(T_1, T_2)$

Now suppose that $d_{V'}(P_1^*, P_2^*) > \frac{1}{2} d_{U'}(T_1, T_2)$ and let $d_{U'}(T_1, T_2) = 2^{-k}$. Let, without loss of generality, P_1^* and P_2^* differ on the sets of their divergences and let there exist $s \in \mathsf{divergences}(P_1^*)$ such that $s \notin \mathsf{divergences}(P_2^*)$ and $\mathsf{length}_{V'}(s) < k + 1$. Then, since $P_i^* = \bigcap_{n=0}^{\infty} P_i^n$, there exists $l \in \mathbb{N}$ such that $s \notin P_2^l$, but for all $n \in \mathbb{N}$, $s \in P_1^n$. But then $d_{V'}(P_1^l, P_2^l) > 2^{-(k+1)} = \frac{1}{2} d_{U'}(T_1, T_2)$, which is a contradiction with (D.6). Therefore, $d_{V'}(P_1^*, P_2^*) \leq \frac{1}{2} d_{U'}(T_1, T_2)$. Then, since $U \subseteq U'$ and $V' \subseteq V$, by antitoneness, $d_V((\mu Y \centerdot P)(T_1, \overline{\Theta}), (\mu Y \centerdot P)(T_2, \overline{\Theta})) = d_V(P_1^*, P_2^*) \leq \frac{1}{2} d_U(T_1, T_2)$. □

□

Proposition 5.5. *Let $P(X_1, \ldots, X_n) = P(\overline{X})$ be a CSP term whose free variables are contained within the set $\{X_1, \ldots, X_n\}$. Let $\mathsf{F} : \mathsf{CSP} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ be defined recursively on the structure of P as shown in Figure 9. If $(U, V) \in \mathsf{F}(P)$, then, for any collection of U-fair livelock-free processes $\theta_1, \ldots, \theta_n \in \mathcal{T}^{\Downarrow}$, the process $P(\theta_1, \ldots, \theta_n)$ is livelock-free and V-fair.*

Proof. Structural induction on P.

- $\mathsf{F}(STOP) = \mathsf{F}(SKIP) = \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$.

  Proof. STOP and *SKIP* are livelock-free and do not contain infinite traces. □

- $\mathsf{F}(a \longrightarrow P) = \mathsf{F}(P)$

  Proof. Let $(U, V) \in \mathsf{F}(a \longrightarrow P)$, $\theta_1, \ldots, \theta_n$ be a collection of livelock-free U-fair processes.
  Since $(U, V) \in \mathsf{F}(a \longrightarrow P)$, by construction, $(U, V) \in \mathsf{F}(P)$. Therefore, by induction
  hypothesis, $P(\overline{\Theta})$ is livelock-free and V-fair. Therefore, $a \longrightarrow P(\overline{\Theta})$ is livelock-free.
  We will prove that $a \longrightarrow P(\overline{\Theta})$ is V-fair. Let $u \in \mathsf{traces}^\omega(a \longrightarrow P(\overline{\Theta}))$. Therefore, by
  Lemma 3.2, there exists $u' \in \mathsf{traces}^\omega(P(\overline{\Theta}))$, such that $u = \langle a \rangle ^\frown u'$. Since $P(\overline{\Theta})$ is V-fair,
  u' contains infinitely many events from V, and so does therefore u. Hence, $a \longrightarrow P(\overline{\Theta})$
  is V-fair.   □

- $\mathsf{F}(P_1 \oplus P_2) = \mathsf{F}(P_1) \cap \mathsf{F}(P_2)$ for $\{\sqcap, \Box\}$

  Proof. Let $(U, V) \in \mathsf{F}(P_1 \oplus P_2)$, $\theta_1, \ldots, \theta_n$ be a collection of livelock-free U-fair processes.
  Since $(U, V) \in \mathsf{F}(P_1 \oplus P_2)$, by construction, $(U, V) \in \mathsf{F}(P_1)$ and $(U, V) \in \mathsf{F}(P_2)$. There-
  fore, by induction hypothesis, $P_1(\overline{\Theta})$ and $P_2(\overline{\Theta})$ are livelock-free and V-fair. Therefore,
  $P_1(\overline{\Theta}) \oplus P_2(\overline{\Theta})$ is livelock-free.
  Let $u \in \mathsf{traces}^\omega(P_1(\overline{\Theta}) \oplus P_2(\overline{\Theta}))$. Then, by Lemma 3.3, $u \in \mathsf{traces}^\omega(P_1(\overline{\Theta}))$ or $u \in$
  $\mathsf{traces}^\omega(P_2(\overline{\Theta}))$. Let without loss of generality the former holds. Then, since $P_1(\overline{\Theta})$ is
  V-fair, u contains infinitely many events from V. Therefore, $P_1(\overline{\Theta}) \oplus P_2(\overline{\Theta})$ is V-fair. □

- $\mathsf{F}(P_1 \mathbin{\raise0.3ex\hbox{$\fgemini$}} P_2) = \mathsf{F}(P_1) \cap \mathsf{F}(P_2)$

  Proof. Let $(U, V) \in \mathsf{F}(P_1 \mathbin{\raise0.3ex\hbox{$\fgemini$}} P_2)$, $\theta_1, \ldots, \theta_n$ be a collection of livelock-free U-fair processes.
  Since $(U, V) \in \mathsf{F}(P_1 \mathbin{\raise0.3ex\hbox{$\fgemini$}} P_2)$, by construction, $(U, V) \in \mathsf{F}(P_1)$ and $(U, V) \in \mathsf{F}(P_2)$. There-
  fore, by induction hypothesis, $P_1(\overline{\Theta})$ and $P_2(\overline{\Theta})$ are livelock-free and V-fair. Therefore,
  $P_1(\overline{\Theta}) \mathbin{\raise0.3ex\hbox{$\fgemini$}} P_2(\overline{\Theta})$ is livelock-free.
  Let $u \in \mathsf{traces}^\omega(P_1(\overline{\Theta}) \mathbin{\raise0.3ex\hbox{$\fgemini$}} P_2(\overline{\Theta}))$. Then, by Lemma 3.4, $u \in \mathsf{traces}^\omega(P_1(\overline{\Theta}))$ or $u =$
  $t_1 ^\frown u_2$ with $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(P_1(\overline{\Theta})) \cap \Sigma^{*\checkmark}$, $u_2 \in \mathsf{traces}^\omega(P_2(\overline{\Theta}))$.
  – Suppose $u \in \mathsf{traces}^\omega(P_1(\overline{\Theta}))$. Since $P_1(\overline{\Theta})$ is V-fair, u contains infinitely many events
    from V.
  – Suppose $u = t_1 ^\frown u_2$ with $t_1 ^\frown \langle \checkmark \rangle \in \mathsf{traces}(P_1(\overline{\Theta})) \cap \Sigma^{*\checkmark}$, $u_2 \in \mathsf{traces}^\omega(P_2(\overline{\Theta}))$. Since
    $P_2(\overline{\Theta})$ is V-fair, u_2 contains infinitely many events from V and so does therefore u.
  Therefore, $P_1(\overline{\Theta}) \mathbin{\raise0.3ex\hbox{$\fgemini$}} P_2(\overline{\Theta})$ is V-fair.   □

- $\mathsf{F}(P_1 \underset{A}{\|} P_2) = \begin{aligned}[t] &(\mathsf{F}(P_1) \cap \mathsf{F}(P_1)) \cup \\ &\{(U_1 \cap U_2, V_1) \mid (U_1, V_1) \in \mathsf{F}(P_1) \wedge (U_2, A) \in \mathsf{F}(P_2)\} \cup \\ &\{(U_1 \cap U_2, V_2) \mid (U_2, V_2) \in \mathsf{F}(P_2) \wedge (U_1, A) \in \mathsf{F}(P_1)\} \end{aligned}$

  Proof. Let $(U, V) \in \mathsf{F}(P_1 \underset{A}{\|} P_2)$, $\theta_1, \ldots, \theta_n$ be a collection of livelock-free U-fair processes.
  Since $(U, V) \in \mathsf{F}(P_1 \underset{A}{\|} P_2)$, by construction, $\mathsf{F}(P_1) \neq \emptyset$, $\mathsf{F}(P_2) \neq \emptyset$ and, by induction
  hypothesis, $P_1(\overline{\Theta})$ and $P_2(\overline{\Theta})$ are livelock-free. Therefore, $P_1(\overline{\Theta}) \underset{A}{\|} P_2(\overline{\Theta})$ is livelock-free.
  Let $u \in \mathsf{traces}^\omega(P_1(\overline{\Theta}) \underset{A}{\|} P_2(\overline{\Theta}))$. Hence, by Lemma 3.7, there exist $u_1 \in \mathsf{traces}^\infty(P_1(\overline{\Theta}))$,
  $u_2 \in \mathsf{traces}^\infty(P_2(\overline{\Theta}))$, such that $u \in u_1 \underset{A}{\|} u_2$ and, $u_1 \in \Sigma^\omega$ or $u_2 \in \Sigma^\omega$. Let without loss
  of generality $u_1 \in \Sigma^\omega$. By construction, we have three alternatives for (U, V).
  – Suppose $(U, V) \in \mathsf{F}(P_1) \cap \mathsf{F}(P_2)$. By induction hypothesis, $P_1(\overline{\Theta})$ is V-fair. Therefore,
    u_1 contains infinitely many events from V and so does u.

– Suppose (U, V) is $(U_1 \cap U_2, V)$ with $(U_1, V) \in \mathsf{F}(P_1)$, $(U_2, A) \in \mathsf{F}(P_2)$. Then $U = U_1 \cap U_2 \subseteq U_1$ and $\theta_1, \ldots, \theta_n$ are U_1-fair. Hence, by induction hypothesis, $P_1(\overline{\Theta})$ is V-fair. Then, u_1 contains infinitely many events from V and so does therefore u.

– Suppose (U, V) is $(U_1 \cap U_2, V)$ with $(U_2, V) \in \mathsf{F}(P_2)$, $(U_1, A) \in \mathsf{F}(P_1)$. Since $U = U_1 \cap U_2$, we have $U \subseteq U_1$, $U \subseteq U_2$ and therefore, $\theta_1, \ldots, \theta_n$ are U_1-fair and U_2-fair. By induction hypothesis for P_1, u_1 contains infinitely many events from A. Since u_1 and u_2 synchronise on the events in A, u_2 contains infinitely many events from A. Therefore, $u_2 \in \Sigma^\omega$ and by induction hypothesis for P_2, u_2 contains infinitely many events from V. Hence, u contains infinitely many events from V.

Therefore, $P_1(\overline{\Theta}) \underset{A}{\|} P_2(\overline{\Theta})$ is V-fair. $\qquad\square$

- $\mathsf{F}(P \setminus A) = \{(U, V) \mid (U, V') \in \mathsf{F}(P) \land V' \cap A = \emptyset \land V' \subseteq V\}$

  Proof. Let $(U, V) \in \mathsf{F}(P \setminus A)$, $\theta_1, \ldots, \theta_n$ be a collection of livelock-free U-fair processes.
  Since $(U, V) \in \mathsf{F}(P \setminus A)$, by construction, there exists $V' \subseteq V$, such that $V' \cap A = \emptyset$ and $(U, V') \in \mathsf{F}(P)$. Therefore, by induction hypothesis, $P(\overline{\Theta})$ is livelock-free and V'-fair. Suppose $P(\overline{\Theta}) \setminus A$ is not livelock-free. Therefore, since $P(\overline{\Theta})$ is livelock-free, there exists $u \in \mathsf{traces}^\omega(P)$ such that $u \restriction (\Sigma \backslash A)$ is finite. Since $P(\overline{\Theta})$ is V'-fair, u contains infinitely many events from V'. Since $V' \cap A = \emptyset$, $u \restriction (\Sigma \backslash A)$ contains infinitely events from V', which is a contradiction with $u \restriction (\Sigma \backslash A)$ being finite. Therefore, $P(\overline{\Theta}) \setminus A$ is livelock-free.
  Let $u \in \mathsf{traces}^\omega(P(\overline{\Theta}) \setminus A)$. Then, by Lemma 3.5, there exists $u' \in \mathsf{traces}^\omega(P(\overline{\Theta}))$, such that $u = u' \restriction (\Sigma \backslash A)$. Since $P(\overline{\Theta})$ is V'-fair, u' contains infinitely many events from V'. Since $V' \cap A = \emptyset$, $u = u' \restriction (\Sigma \backslash A)$ contains infinitely many events from $V' \subseteq V$. Therefore, u contains infinitely many events from V. Therefore, $P(\overline{\Theta}) \setminus A$ is V-fair. $\quad\square$

- $\mathsf{F}(P[R]) = \{(U, V) \mid (U, V') \in \mathsf{F}(P) \land R(V') \subseteq V\}$

  Proof. Let $(U, V) \in \mathsf{F}(P[R])$, $\theta_1, \ldots, \theta_n$ be a collection of livelock-free U-fair processes.
  Since $(U, V) \in \mathsf{F}(P[R])$, by construction, there exists V', such that $R(V') \subseteq V$ and $(U, V') \in \mathsf{F}(P)$. Therefore, by induction hypothesis, $P(\overline{\Theta})$ is livelock-free and V'-fair. Hence, $P(\overline{\Theta})[R]$ is also livelock-free.
  Let $u \in \mathsf{traces}^\omega(P(\overline{\Theta})[R])$. Then, by Lemma 3.6, there exists $u' \in \mathsf{traces}^\omega(P(\overline{\Theta}))$, such that $u' \ R \ u$. Since $P(\overline{\Theta})$ is V'-fair, u' contains infinitely many events from V'. Since $u' \ R \ u$, u contains infinitely many events from $R(V') \subseteq V$. Therefore, u contains infinitely many events from V. Hence, $P(\overline{\Theta})[R]$ is V-fair. $\qquad\square$

- $\mathsf{F}(X) = \{(U, V) \mid U \subseteq V\}$

  Proof. Let $(U, V) \in \mathsf{F}(X)$, θ be a livelock-free U-fair process. $X(\theta) = \theta$ is then livelock-free and, since $U \subseteq V$, $X(\theta) = \theta$ is V-fair. $\qquad\square$

- $\mathsf{F}(\mu X \boldsymbol{.} P) = \begin{cases} \{(U, V) \mid (W, W) \in \mathsf{C}_X(P) \cap \mathsf{F}(P) \land U \subseteq W \subseteq V\} & \text{if } \mu X \boldsymbol{.} P \text{ is open} \\ \mathcal{P}(\Sigma) \times \{V \mid (W, W) \in \mathsf{C}_X(P) \cap \mathsf{F}(P) \land W \subseteq V\} & \text{otherwise} \end{cases}$

  Proof. Let $P(X, Y_1, \ldots, Y_n)$ be a CSP term whose free variable are contained within the set $\{X, Y_1, \ldots, Y_n\}$. Let $(U, V) \in \mathsf{F}(\mu X \boldsymbol{.} P)$ and $(\mu X \boldsymbol{.} P)$ be open. Let $\theta_1, \ldots, \theta_n$ be a collection of livelock-free U-fair processes.
  Since $(U, V) \in \mathsf{F}(\mu X \boldsymbol{.} P)$, by construction, there exists W, such that $U \subseteq W \subseteq V$ and $(W, W) \in \mathsf{C}_X(P) \cap \mathsf{F}(P)$. Therefore $\theta_1, \ldots, \theta_n$ are W-fair and, by induction hypothesis:

  $$P(\xi, \overline{\Theta}) \text{ is livelock-free and } W\text{-fair for any livelock-free } W\text{-fair process } \xi. \qquad \text{(D.7)}$$

Since $(W, W) \in \mathsf{C}_X(P)$, by Proposition D.1 (2), $P(X, \overline{\Theta})$ is contractive in X with respect to the metric d_W. Therefore, from Banach's fixed point theorem, $P(X, \overline{\Theta})$ has a unique fixed point $(\mu X \centerdot P)(\overline{\theta}) = \bigcup_{n=0}^{\infty} P^n = P^*$, where $P^0 = \top = STOP$, $P^{n+1} = P(P^n, \overline{\Theta})$.

We will prove by induction that for each $n \in \mathbb{N}$, P^n is livelock-free and W-fair.
- $n = 0$. $STOP$ is livelock-free and does not contain infinite traces.
- Suppose that P^n is livelock-free and W-fair. From (D.7), $P^{n+1} = P(P^n, \overline{\Theta})$ is also livelock-free and W-fair.

Therefore, for each $n \in \mathbb{N}$, P^n is livelock-free. Then, since by Proposition B.3 the set of livelock-free processes is closed, $(\mu X \centerdot P)(\overline{\theta}) = \bigcup_{n=0}^{\infty} P^n = P^*$ is livelock-free.

Now, let $u \in \mathsf{traces}^{\omega}((\mu X \centerdot P)(\overline{\Theta}))$. Then, for each finite prefix t of u, $t \in \mathsf{traces}(P^*)$, i.e., there exists some sufficiently large n_t, such that $t \in \mathsf{traces}(P^{n_t})$.

If there exists $m \in \mathbb{N}$ such that for each prefix t of u, $t \in \mathsf{traces}(P^m)$, then $u \in \mathsf{traces}^{\omega}(P^m)$. In this case, since P^m is W-fair, u contains infinitely many events from W. Then, since $W \subseteq V$, u contains infinitely many events from V.

Otherwise, we can conclude the following:

$$\text{for each } m \in \mathbb{N}, \text{ there exists a prefix } t \text{ of } u, \text{ such that } t \notin \mathsf{traces}(P^m). \qquad \text{(D.8)}$$

Let $\varepsilon = 2^{-k}$ for some $k \in \mathbb{N}$. Since the sequence $\langle P^i \mid i \in \mathbb{N} \rangle$ converges to P^* with respect to the metric d_W, there exists $n_\varepsilon \in \mathbb{N}$, such that for each $n \geq n_\varepsilon$, $d_W(P^*, P^n) < \varepsilon$. From our assumption (D.8), for n_ε there exists t_ε, such that t_ε is a prefix of u and $t_\varepsilon \notin \mathsf{traces}(P^{n_\varepsilon})$. Then, since $d_W(P^*, P^{n_\varepsilon}) < \varepsilon$ and $t_\varepsilon \in \mathsf{traces}(P^*)$, $\mathsf{length}_W t_\varepsilon \geq k$. Since k was arbitrary, we can conclude that u contains infinitely many events from W. Then again, since $W \subseteq V$, u contains infinitely many events from V. Therefore, $(\mu X \centerdot P)(\overline{\theta})$ is V-fair. $\qquad \square$

APPENDIX E. PROOFS FOR SECTION 6

Proposition 6.1. *Let P be a structurally finite state process. Let $\Phi : \overline{\mathsf{SFS}} \longrightarrow \mathcal{P}(\mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma))$ and $\delta : \overline{\mathsf{SFS}} \longrightarrow \{\text{true}, \text{false}\}$ be defined recursively on the structure of P as shown in Figures 10 and 11, respectively. Then, if $\delta(P) = \text{false}$, P is livelock-free. Moreover, if in addition $\Phi(P) = \{(F_1, C_1), \ldots, (F_k, C_k)\}$, then, for each infinite trace u of P, there exists $1 \leq i \leq k$, such that u is fair in F_i and u is co-fair in C_i.*

Proof. Induction on the structure of the $\overline{\mathsf{SFS}}$ process P.

Note that by construction, all fair/co-fair pairs of sets thus generated remain disjoint, i.e., for each $(F, C) \in \Phi(P)$, $F \cap C = \emptyset$. This is key in the rule for parallel composition, where the fair/co-fair data of individual sub-components enables one to rule out certain pairs for the resulting parallel process. We prove this property only for the case of renaming as for all other cases it follows trivially from the induction hypothesis and the specific construction.

Let us also remark that it might be the case that $\delta(P) = \text{false}$ and $\Phi(P) = \emptyset$ and this indicates that P is livelock free but exhibits only finite traces. We note, however, that if $\delta(P) = \text{false}$ and $\Phi(P) \neq \emptyset$, then for every $(F, C) \in \Phi(P)$, $F \neq \emptyset$. This is true for sequential $\overline{\mathsf{SFS}}$ processes by construction and follows for compound $\overline{\mathsf{SFS}}$ processes by induction hypothesis and construction. We prove the property only for the cases of hiding and renaming where the argument is more subtle.

- For P being a closed sequential process, $\Phi(P)$ and $\delta(P)$ are computed directly from the labelled transition system associated with P as described in Section 6.
- $\delta(a \longrightarrow P) = \delta(P)$ and $\Phi(a \longrightarrow P) = \Phi(P)$.

  Proof. Let $\delta(a \longrightarrow P) =$ false. By construction, $\delta(P) =$ false and, therefore, by induction hypothesis, P is livelock-free. Hence, by definition, $a \longrightarrow P$ is also livelock-free.

  Let $u \in \mathsf{traces}^\omega(a \longrightarrow P)$. Then, by Lemma 3.2, there exists $u' \in \mathsf{traces}^\omega(P)$, such that $u = \langle a \rangle^\frown u'$. By induction hypothesis for P, there exists $(F, C) \in \Phi(P)$, such that u' is fair in F and co-fair in C. But then u is also fair in F and co-fair in C and, by construction, $(F, C) \in \Phi(a \longrightarrow P)$. $\qquad\qquad\square$

- $\delta(P_1 \oplus P_2) = \delta(P_1) \vee \delta(P_2)$ and $\Phi(P_1 \oplus P_2) = \Phi(P_1) \cup \Phi(P_2)$ if $\oplus \in \{\sqcap, \square\}$.

  Proof. Let $\delta(P_1 \oplus P_2) =$ false. By construction, $\delta(P_1) =$ false and $\delta(P_2) =$ false. Therefore, by induction hypothesis, P_1 and P_2 are livelock-free. Hence, by definition, $P_1 \oplus P_2$ is livelock-free.

  Let $u \in \mathsf{traces}^\omega(P_1 \oplus P_2)$. By Lemma 3.3, $u \in \mathsf{traces}^\omega(P_1)$ or $u \in \mathsf{traces}^\omega(P_2)$. Let without loss of generality the former holds. Then, by induction hypothesis for P_1, there exists $(F, C) \in \Phi(P_1)$, such that u is fair in F and co-fair in C. By construction, $\Phi(P_1) \subseteq \Phi(P_1 \oplus P_2)$ and, therefore, $(F, C) \in \Phi(P_1 \oplus P_2)$. $\qquad\square$

- $\delta(P_1 \mathbin{;} P_2) = \delta(P_1) \vee \delta(P_2)$ and $\Phi(P_1 \mathbin{;} P_2) = \Phi(P_1) \cup \Phi(P_2)$.

  Proof. Let $\delta(P_1 \mathbin{;} P_2) =$ false. By construction, $\delta(P_1) =$ false and $\delta(P_2) =$ false. Therefore, by induction hypothesis, P_1 and P_2 are livelock-free. Hence, by definition, $P_1 \mathbin{;} P_2$ is livelock-free.

  Let $u \in \mathsf{traces}^\omega(P_1 \mathbin{;} P_2)$. By Lemma 3.4, $u \in \mathsf{traces}^\omega(P_1)$ or $u = t^\frown u'$ with $t^\frown \langle \checkmark \rangle \in \mathsf{traces}(P_1) \cap \Sigma^{*\checkmark}$, $u' \in \mathsf{traces}^\omega(P_2)$. We consider both alternatives.
  - If $u \in \mathsf{traces}^\omega(P_1)$, by induction hypothesis for P_1, there exists $(F, C) \in \Phi(P_1)$, such that u is fair in F and co-fair in C. By construction, $\Phi(P_1) \subseteq \Phi(P_1 \oplus P_2)$ and, therefore, $(F, C) \in \Phi(P_1 \oplus P_2)$.
  - Let $u = t^\frown u'$ where $t^\frown \langle \checkmark \rangle \in \mathsf{traces}(P_1) \cap \Sigma^{*\checkmark}$ and $u' \in \mathsf{traces}^\omega(P_2)$. By induction hypothesis for P_2, there exists $(F, C) \in \Phi(P_2)$, such that u' is fair in F and co-fair in C. The finite prefix t of u does not affect fairness and co-fairness. Therefore, $u = t^\frown u'$ is fair in F and co-fair in C and $(F, C) \in \Phi(P_1 \oplus P_2)$ by construction.

    $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

- $\delta(P \setminus A) = \begin{cases} \text{false} & \text{if } \delta(P) = \text{false and, for each } (F, C) \in \Phi(P),\ F - A \neq \emptyset \\ \text{true} & \text{otherwise} \end{cases}$
  and $\Phi(P \setminus A) = \{(F - A, C \cup A) \mid (F, C) \in \Phi(P)\}$.

  Proof. Let $\delta(P \setminus A) =$ false. By construction, $\delta(P) =$ false and for each $(F, C) \in \Phi(P)$, $F - A \neq \emptyset$. Since $\delta(P) =$ false, by induction hypothesis, P is livelock-free. Suppose for the sake of the argument that $P \setminus A$ can diverge. Since P is livelock-free, by definition, the only alternative is that there exists $u \in \mathsf{traces}^\omega(P)$, such that $u \upharpoonright (\Sigma \setminus A)$ is finite. By induction hypothesis for P, there exists $(F, C) \in \Phi(P)$, such that u is fair in F and co-fair in C. By construction, since $\delta(P \setminus A) =$ false, $F - A \neq \emptyset$. Therefore, there exists $b \in F$ such that $b \notin A$ and b occurs infinitely many times in u. But then b should also occur infinitely many times in $u \upharpoonright (\Sigma \setminus A)$, which is a contradiction with $u \upharpoonright (\Sigma \setminus A)$ being finite. Therefore, $P \setminus A$ is livelock-free.

Now, let $u \in \mathsf{traces}^\omega(P \setminus A)$. Since $P \setminus A$ is livelock-free, by Lemma 3.5, there exists $v \in \mathsf{traces}^\omega(P)$ such that $u = v \restriction (\Sigma \setminus A)$. By induction hypothesis for P, there exists $(F, C) \in \Phi(P)$ such that v is fair in F and co-fair in C. Then, since u is obtained by deleting all A-events from v, u is fair in $F - A$ and co-fair in $C \cup A$. Both $F - A \neq \emptyset$ and $(F - A, C \cup A) \in \Phi(P \setminus A)$ are guaranteed by construction.

Let $\delta(P \setminus A) = \text{false}$ and let $(F, C) \in \Phi(P \setminus A)$. We now prove that $F \neq \emptyset$. Since $\delta(P \setminus A) = \text{false}$, by construction we have the following:

$$\text{for each } (F', C') \in \Phi(P), F' - A \neq \emptyset. \tag{E.1}$$

As $(F, C) \in \Phi(P \setminus A)$, by construction $F = F' - A$ for some F' with $(F', C') \in \Phi(P)$. By (E.1), $F' - A \neq \emptyset$ and hence $F \neq \emptyset$. $\qquad\square$

- $\delta(P[R]) = \delta(P)$ and
$\Phi(P[R]) = \{(F, C) \mid (F', C') \in \Phi(P) \wedge F' \subseteq R^{-1}(F) \wedge F \subseteq R(F') \wedge$
$\qquad\qquad C = \{b \in \Sigma \mid R^{-1}(b) \subseteq C'\}\}$

Proof. In the proof we use the following notation. For any $A \subseteq \Sigma$, $a, b \in \Sigma$, $R(A) = \{b \mid \exists a \in A \centerdot a \ R \ b\}$ and $R^{-1}(b) = \{a \mid a \ R \ b\}$. Let us also clarify that in the setting of CSP [21] renaming relations are assumed to be total. If an event $a \in \Sigma$ is not renamed to any other event $b \in \Sigma$, it is assumed that a is renamed to itself and, hence, $R(\{a\}) \neq \emptyset$.

Let $\delta(P[R]) = \text{false}$. By construction, $\delta(P) = \text{false}$. Then, by induction hypothesis, P is livelock-free and, hence, by definition, so is $P[R]$.

Let $u \in \mathsf{traces}^\omega(P[R])$. By Lemma 3.6, there exists $v \in \mathsf{traces}^\omega(P)$, such that $v \ R \ u$, i.e., for every $i \in \mathbb{N}$, $v(i) \ R \ u(i)$. By induction hypothesis for P, there exists $(F', C') \in \Phi(P)$, such that v is fair in F' and co-fair in C'.

Let $C = \{b \in \Sigma \mid R^{-1}(b) \subseteq C'\}$ and let $b \in C$. By construction, for each $a \in R^{-1}(b)$, $a \in C'$ and, therefore, v is co-fair in a. Now suppose for the sake of contradiction that u contains infinitely many occurrences of b. By definition, there exists $a \in \Sigma$, such that $a \ R \ b$ and a occurs infinitely many times in v. Therefore, $a \notin C'$ and $R^{-1}(b) \not\subseteq C'$, which is a contradiction with $R^{-1}(b) \subseteq C'$. Therefore, u contains only finitely many b's and, more generally, u is co-fair in C.

We will construct F from F' such that $F \subseteq R(F')$ (which will bound F from above), $F' \subseteq R^{-1}(F)$ (which will bound F from below and will guarantee $F \neq \emptyset$) and u is fair in F. Then $(F, C) \in \Phi(P[R])$ by construction.

By induction hypothesis, $F' \neq \emptyset$. Let $F' = \{a_1, \ldots, a_m\}$ and for each $1 \leq i \leq m$, $R(\{a_i\}) = \{b_{i_1}, \ldots, b_{i_{n_i}}\}$. As each a_i occurs infinitely many times in v and $v \ R \ u$, for each $1 \leq i \leq m$, there exists b_{j_i}, such that $a_i \ R \ b_{j_i}$ and b_{j_i} occurs infinitely many times in u. We define $F = \{b_{j_1}, \ldots, b_{j_m}\}$. Since $F' \neq \emptyset$, $F \neq \emptyset$. By the construction of F, u is fair in F and $F \subseteq R(F')$. As by construction for each $a_i \in F'$ there exists $b_{j_i} \in F$ with $a_i \ R \ b_{j_i}$, then for every $1 \leq i \leq m$, $a_i \in R^{-1}(b_{j_i})$. Therefore, $F' \subseteq R^{-1}(F)$.

We will also prove that for any F that satisfies $F \subseteq R(F')$ and $F' \subseteq R^{-1}(F)$, the sets F and C are disjoint. Suppose there exists $b \in \Sigma$ such that $b \in F \cap C$. As $b \in C$, by construction, for each a with $a \ R \ b$, we have $a \in C'$. Since $b \in F$ and by construction $F \subseteq R(F')$, there exists $a \in F'$, such that $a \ R \ b$. Therefore, $a \in C' \cap F'$. This is a contradiction with the induction hypothesis according to which F' and C' are disjoint. Therefore, $F \cap C = \emptyset$. $\qquad\square$

- $\delta(P_1 \underset{A}{\parallel} P_2) = \delta(P_1) \vee \delta(P_2)$ and

$$\Phi(P_1 \underset{A}{\parallel} P_2) = \{(F, C) \mid F \cap C = \emptyset \wedge (F_i, C_i) \in \Phi(P_i) \text{ for } i = 1, 2 \wedge F = F_1 \cup F_2 \wedge$$
$$C = (C_1 \cap A) \cup (C_2 \cap A) \cup ((C_1 - A) \cap (C_2 - A))\} \cup$$
$$\{(F, C) \mid (F, C) \in \Phi(P_1) \wedge F \cap A = \emptyset\} \cup$$
$$\{(F, C) \mid (F, C) \in \Phi(P_2) \wedge F \cap A = \emptyset\}$$

Proof. Let $\delta(P_1 \underset{A}{\parallel} P_2) = $ false. By construction, $\delta(P_1) = $ false and $\delta(P_2) = $ false. There-
fore, by induction hypothesis, P_1 and P_2 are livelock-free. Hence, by definition, $P_1 \underset{A}{\parallel} P_2$
is livelock-free.

Let $u \in \mathsf{traces}^\omega(P_1 \underset{A}{\parallel} P_2)$. From Lemma 3.7, there exist $u_1 \in \mathsf{traces}^\infty(P_1)$ and $u_2 \in$
$\mathsf{traces}^\infty(P_2)$, such that $u \in u_1 \underset{A}{\parallel} u_2$ and, $u_1 \in \Sigma^\omega$ or $u_2 \in \Sigma^\omega$. We will consider three
different cases.

− Let $u_1 \in \Sigma^\omega$ and $u_2 \in \Sigma^{*\checkmark}$. By induction hypothesis for P_1, there exists $(F, C) \in \Phi(P_1)$
  such that u_1 is fair in F and co-fair in C. Suppose $F \cap A \neq \emptyset$. Then, u_1 contains
  infinitely many occurrences of events from A. Since P_1 and P_2 synchronise on the events
  in A, u_2 must also contain infinitely many events from A, which is a contradiction with
  $u_2 \in \Sigma^{*\checkmark}$. Therefore, $F \cap A = \emptyset$ and, by construction, $(F, C) \in \Phi(P_1 \underset{A}{\parallel} P_2)$. Now,
  since u_2 is finite and does not affect fairness and co-fairness, u is fair in F and co-fair
  in C.
− The case where $u_2 \in \Sigma^\omega$ and $u_1 \in \Sigma^{*\checkmark}$ is handled in the same way.
− Let $u_1 \in \Sigma^\omega$ and $u_2 \in \Sigma^\omega$. By induction hypothesis for P_1 and P_2, there exist $(F_1, C_1) \in$
  $\Phi(P_1)$ and $(F_2, C_2) \in \Phi(P_2)$, such that u_1 is fair in F_1 and co-fair in C_1 and u_2 is fair
  in F_2 and co-fair in C_2. We note, that for each $a \in A$, the number of occurrences of a
  in u_1, u_2 and u is the same due to P_1 and P_2 synchronising on a. Therefore, for each
  $a \in A$, u_1 contains infinitely many occurrences of a if and only if u_2 contains infinitely
  many occurrences of a. Hence, $F_1 \cap C_2 \cap A = \emptyset$ and $F_2 \cap C_1 \cap A = \emptyset$.
  Let $F = F_1 \cup F_2$ and $C = (C_1 \cap A) \cup (C_2 \cap A) \cup ((C_1 - A) \cap (C_2 - A))$.
  We will first prove that $F \cap C = \emptyset$. Suppose for the sake of the argument that there
  exists $b \in \Sigma$ such that $b \in F \cap C$. Since $b \in F$, by construction, $b \in F_1$ or $b \in F_2$. Let
  without loss of generality $b \in F_1$. We will consider the cases $b \in A$ and $b \notin A$.
  ∗ Suppose $b \in A$. Since $b \in F_1$, u_1 is fair in b and, therefore, $b \notin C_1$. Since $b \in C$ and
    $b \in A$, $b \in C_1 \cap A$ or $b \in C_2 \cap A$. As $b \notin C_1$, $b \in C_2 \cap A$. Therefore, $b \in F_1 \cap C_2$
    which is a contradiction with $F_1 \cap C_2 \cap A = \emptyset$. Therefore, this case is not possible.
  ∗ Suppose $b \notin A$. Since $b \in C$, $b \in C_1$ and $b \in C_2$. Therefore, $b \in F_1 \cap C_1$ which
    is a contradiction with the induction hypothesis by which F_1 and C_1 are disjoint.
    Therefore, this case is not possible either.
  Therefore, $F \cap C = \emptyset$.
  Now, for any event $b \in \Sigma$, if $b \in F_1$ or $b \in F_2$, i.e., b has infinitely many occurrences
  in u_1 or u_2, then b has infinitely many occurrences in $u \in u_1 \underset{A}{\parallel} u_2$ as well. Therefore,
  u is fair in $F_1 \cup F_2$.
  Let for some $a \in A$, $a \in C_1$ or $a \in C_2$ and let without loss of generality the former
  holds. Then, a occurs only finitely many times in u_1 and, since P_1 and P_2 synchronise
  on a, a occurs only finitely many times in u_2 and u as well. Therefore, u is co-fair

in a and, more generally, in $(C_1 \cap A) \cup (C_2 \cap A)$. Now let $b \in (C_1 \backslash A) \cap (C_2 \backslash A)$. Therefore, $b \notin A$, $b \in C_1$ and $b \in C_2$. Therefore, since b occurs only finitely often in both u_1 and u_2, b occurs only finitely often in u as well. Therefore, u is also co-fair in $(C_1 \backslash A) \cap (C_2 \backslash A)$. Hence, u is co-fair in $(C_1 \cap A) \cup (C_2 \cap A) \cup ((C_1 \backslash A) \cap (C_2 \backslash A))$ and $(F_1 \cup F_2, (C_1 \cap A) \cup (C_2 \cap A) \cup ((C_1 \backslash A) \cap (C_2 \backslash A)) \in \Phi(P_1 \underset{A}{\|} P_2)$ by construction. $\square$

Proposition 6.2. *For any structurally finite-state process P, if $\mathsf{F}(P) \neq \emptyset$ then $\delta(P) = $ false.*

Proof. (Sketch.) One shows by structural induction on the $\overline{\mathsf{SFS}}$ process P the stronger statement that if $\mathsf{F}(P) \neq \emptyset$ then (i) $\delta(P) = $ false, and (ii) for any $(U, V) \in \mathsf{F}(P)$ and any $(F, C) \in \Phi(P)$, it is the case that $F \cap V \neq \emptyset$.

All cases are relatively straightforward. Note that, since $P \in \overline{\mathsf{SFS}}$, recursion does not need to be handled, as it falls within the 'sequential $\overline{\mathsf{SFS}}$' case. It is worth pointing out that, in carrying out the inductive proof, it turns out that it is never necessary to take account of any information regarding either U or C; they can be ignored entirely. $\square$

APPENDIX F. CASE STUDY: AN ABSTRACTED VERSION OF THE ALTERNATING BIT PROTOCOL

In this section, we briefly describe an abstracted version of a network communication protocol called the Alternating Bit Protocol. We use the abstracted version only to illustrate our concepts. For the experimental evaluation in Section 7 we use a modelling of the authentic protocol, the script for which can be found on the website associated with [22].

The process *Send* (see Figures 12 and 13) attempts to send messages to itself infinitely often. Those messages, however, have to go through an unreliable network *Medium*, which may do an arbitrary (possibly infinite) number of *error* events before delivering the message back to *Send* in the form of an *out* event. We impose a fairness constraint *Fair* on *Medium*, forcing it to do at most a single error before delivering the message correctly, i.e., we require that every *error* event be immediately followed by an *out* event. We construct the system by putting the mutually-recursive processes *Send* and *Medium* in parallel with the process *Fair*, synchronising on the set of their shared events $\{error, out\}$ and hiding the *error* event at the top. The resulting process *System* is livelock-free and is, in fact, equivalent to the process $B_1 = in \longrightarrow out \longrightarrow B_1$, which implements a single-slot buffer.

$$
\begin{aligned}
&Send = in \longrightarrow Medium \\
&Medium = out \longrightarrow Send \ \square \ error \longrightarrow Medium \\[6pt]
&Fair = out \longrightarrow Fair \ \square \ error \longrightarrow out \longrightarrow Fair \\[4pt]
&Network = Send \underset{\{error, out\}}{\|} Fair \\[4pt]
&System = Network \setminus \{error\}
\end{aligned}
$$

Figure 12: ABP: an abstracted version.
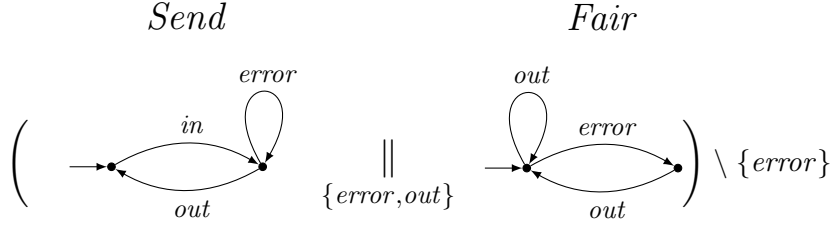
$$Send \qquad\qquad\qquad Fair$$

Figure 13: Abstracted ABP: transition systems.

Using the systems of rules presented in Section 5, we calculate the sets of fair sets of *Send*, *Fair*, *Network* and *System* as follows (where the operator $\uparrow$ denotes upper closure on $\mathcal{P}(\mathcal{P}(\Sigma))$ and $\Sigma = \{in, out, error\}$):

$$\mathsf{F}(Send) \mathrel{\widehat{=}} \uparrow \{\{in, error\}, \{out, error\}\}$$
$$\mathsf{F}(Fair) \mathrel{\widehat{=}} \uparrow \{\{out\}\}$$
$$\mathsf{F}(Network) \mathrel{\widehat{=}} \uparrow \{\{out\}, \{out, error\}, \{in, error\}\}$$
$$\mathsf{F}(System) \mathrel{\widehat{=}} \uparrow \{\{out\}\}$$

Therefore, *System* is livelock-free and any infinite trace of *System* contains infinitely many occurrences of the event *out*.

An interesting weakness of the general framework is that it fails to establish the fact that *System* is also $\{in\}$-fair. Indeed, since *System* is equivalent to the process $B_1 = in \longrightarrow out \longrightarrow B_1$, any infinite trace of *System* should also contain infinitely many occurrences of *in*. Therefore, the process $Network \setminus \{error, out\}$, which is equivalent to the process $IN = in \longrightarrow IN$, is livelock-free and $\{in\}$-fair. However, $\mathsf{F}(Network \setminus \{error, out\}) = \emptyset$ (thanks to the F rule for hiding) and therefore the general framework would mark $Network \setminus \{error, out\}$ as potentially divergent.

Let us now illustrate the precision of the system of rules for $\overline{\mathsf{SFS}}$ processes by trying to establish that the process $System = Network \setminus \{error, out\} = (Send \underset{\{error,out\}}{\parallel} Fair) \setminus \{error, out\}$ is livelock-free.

The processes *Send* and *Fair* depicted in Figure 13 are both sequential $\overline{\mathsf{SFS}}$ processes— for those we apply the algorithms described in Section 6.1 to conclude that $\delta(Send) = \delta(Fair) = $ false and, regarding the set of fair/co-fair pairs,

$$\Phi(Send) = \{ (\{in, out\}, \{error\}), \quad (\{error\}, \{in, out\}), \quad (\{error, in, out\}, \emptyset) \},$$
$$\Phi(Fair) = \{ (\{out\}, \{in, error\}), \quad (\{error, out\}, \{in\}) \}.$$

Now let us consider the process $Network = Send \underset{\{error,out\}}{\parallel} Fair$.

Since both *Send* and *Fair* are livelock-free, there is no way of having a divergence in *Network*, which is confirmed by the rule $\delta(Network) = \delta(Send) \vee \delta(Fair) = $ false.

Let us now have a look at the Φ rule for parallel composition. Since each of the fair/co-fair pairs (F, C) of *Send* and *Fair* have non-empty intersection with the synchronisation set $A = \{error, out\}$ of the parallel composition, we conclude that:

(1) We can only use the first set-comprehension clause for assembling the fair/co-fair pairs of *Network*.

(2) Both *Send* and *Fair* contribute infinite traces in any infinite trace u of *Network*, i.e.,
$u = u_1 \parallel_A u_2$, where u_1 in $\mathsf{traces}^\omega(Send)$ and $u_2 \in \mathsf{traces}^\omega(Fair)$.

Intuitively, every infinite trace of *Fair*, and in particular u_2, contains infinitely many occurrences of *out*. Since *Send* and *Fair* synchronise on *out*, u_1 also contains infinitely many occurrences of *out*. But in u_1, *out* occurs infinitely often precisely whenever *in* occurs infinitely often. Therefore, u_1, and hence also u, both contain infinitely many occurrences of *in*. Therefore u is fair in *in*.

Formally, since both u_1 and u_2 are infinite, we need to consider every pair $((F_1, C_1), (F_2, C_2))$ in the Cartesian product of $\Phi(Send)$ and $\Phi(Fair)$, decide whether to discard it and, if not, figure out how to merge appropriately the pair of pairs into a single pair (F, C).

One of the crucial observations is the following. For $a \in A = \{error, out\}$, the number of occurrences of a in u_1, u_2 and u is the same. Therefore we can discard all those pairs $((F_1, C_1), (F_2, C_2))$ such that there is $a \in A$ with $a \in F_1 \cap C_2$ or $a \in C_1 \cap F_2$. This leaves us with only two pairs:

(1) $((\{in, out\}, \{error\}), (\{out\}, \{in, error\}))$, and
(2) $((\{error, in, out\}, \emptyset), (\{error, out\}, \{in\}))$.

The important question now is what do we do with the event *in* which does not belong to the synchronisation set A. The reasoning we apply is that u is fair in *in* if at least one of u_1 and u_2 is fair in *in*, and u is co-fair in *in* if both u_1 and u_2 are co-fair in *in*. Then from the first pair we obtain $(F, C) = (\{in, out\}, \{error\})$ and from the second pair we obtain $(F, C) = (\{error, in, out, \}, \emptyset)$. Hence we obtain the following final result for $\Phi(Network)$, which confirms that every infinite trace of *Network* contains infinitely many occurrences of *in*:

$$\Phi(Network) = \{ (\{in, out\}, \{error\}), \quad (\{error, in, out\}, \emptyset) \}$$

Now the only thing that remains is to handle the hiding operator, i.e., analyse $System = Network \setminus \{error, out\}$. Since for all $(F, C) \in \Phi(Network)$, $F - \{error, out\} \neq \emptyset$, $\delta(System)$ = false, i.e., we establish, as required, that *System* is livelock-free. As a nice consequence $\Phi(System) = \{(\{in\}, \{error, out\})\}$ asserts that every infinite trace of u contains infinitely many occurrences of *in* and only finitely many occurrences of *out* and *error*.

Appendix G. Symbolic Encoding

In this section we focus on the details regarding the symbolic part of our frameworks and algorithms.

In general, because we need to encode sets of sets of events, we use one-hot Boolean encoding [11], i.e., for each $a \in \Sigma^{\tau\checkmark}$ we employ a Boolean variable which is also written a. The Boolean formula a then encodes all sets of events $\{A \subseteq \Sigma^{\tau\checkmark} \mid a \in A\}$. For the $\overline{\mathsf{SFS}}$ framework we use a single vector y of $|\Sigma^{\tau\checkmark}|$ Boolean variables, whereas for the general framework—the one described in Section 5—we employ two copies: one vector x for modelling the U component and another y for modelling the V component (see Propositions 5.2, 5.4, and 5.5). In addition, we use auxiliary copies of variables for constructing more complex expressions using quantifiers and substitution. For those we use primed versions of x and y.

It is important to note that SAT techniques enable us to find a *single* fair set or fair/co-fair pair of sets for a process. An advantage of this approach is the efficiency of modern SAT

solvers. However, we need to introduce fresh vectors of variables for each instance of (even the same) subprocess. This is required because it might be necessary to generate different fair or fair/co-fair sets for a given term, depending on the context in which it appears.

Using BDDs [4] enables us to find *all possible* fair or fair/co-fair sets that the system of rules is capable of detecting. Hence we do not need to duplicate subprocess encodings, but we need to take care of variable orderings which can dramatically influence the size of the resulting BDD. We use a variable ordering similar to the ones proposed in [19] and adopted by the probabilistic model checker PRISM [9, 12]. BDDs generally generate more compact representations than SAT encodings due to their canonicity and capacity to capture regularities.

G.1. The $\overline{\mathsf{SFS}}$ Framework.

G.1.1. *Computing Fair/Co-Fair Sets for Sequential $\overline{\mathsf{SFS}}$ Processes.* Let P be a sequential $\overline{\mathsf{SFS}}$ process and let us suppose that we have already established that $\delta(P) = $ false, i.e., that P is livelock-free. As described in Section 6.1, we then generate a collection of fair/co-fair pairs of disjoint sets $\Phi(P) = \{(F_1, C_1), \ldots, (F_k, C_k)\} \subseteq \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma)$ such that for every $1 \leq i \leq k$,

$$(F_i, C_i) \in \Phi(P) \longleftrightarrow \exists\, u \in \mathsf{traces}^\omega(P) \,.\, u \text{ is fair in } F_i \text{ and co-fair in } C_i. \qquad \text{(G.1)}$$

The computation of $\Phi(P)$ is carried out directly on the labelled transition system M_P associated with P (in which unreachable states have been excised). Let us fix $M_P = \langle S, \mathsf{init}, \Sigma_P, \longrightarrow \rangle$ and let us suppose that P is a subcomponent of a system with alphabet Σ.

Algorithm 1 Computing $\Phi(P)$

1: $\Phi(P) = \emptyset$
2: **for every** non-empty set $L \subseteq \Sigma_P$ **do**
3:     construct a labelled graph G_L from P's LTS (having pruned unreachable states) by deleting all $(\Sigma - L)$-labelled transitions
4:     **if** G_L contains an SCC which comprises *every* event in L **then**
5:         include $(L, \Sigma - L)$ in $\Phi(P)$
6:     **end if**
7: **end for**
8: return $\Phi(P)$

For a particular non-empty $L \subseteq \Sigma_P$, deciding whether or not to include $(L, \Sigma - L)$ in $\Phi(P)$ (lines 3–4, Algorithm 1) can be carried out in PTIME. More specifically, after obtaining G_L, we can check whether there exists $s \in S$ such that for every $a \in L$, there exists a transition $src \xrightarrow{a} dest$, such that there are paths from s to src and from $dest$ back to s, as illustrated in Figure 14 for $L = \{in, out, error\}$. Note that such paths necessarily consist entirely of events in $L \cup \{\tau\}$.

In fact, we can encode this symbolically for all possible subsets of Σ_P via the following Boolean formula:
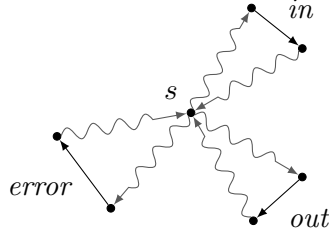
Figure 14: Calculating fair/co-fair sets for sequential $\overline{\mathsf{SFS}}$ processes.

$$MaxSCC = \bigvee_{s \in S} \left\{ \bigwedge_{a \in \Sigma_P} \left[\neg a \vee \bigvee_{src \xrightarrow{a} dest} \left(\mathsf{Path}(s, src) \wedge \mathsf{Path}(dest, s) \right) \right] \right\}, \qquad \text{(G.2)}$$

where:

(1) For all $s, t \in S$, $\mathsf{Path}(s, t)$ encodes all symbolic traces over Σ_P from s to t of length at most $|S|$, i.e., all symbolic traces of length at most the longest simple path in M_P. In order to compute $\mathsf{Path}(s, t)$ for all $s, t \in S$ simultaneously, we extend standard algorithms for computing the transitive closure of the adjacency matrix of the transition relation of M_P, such as Floyd-Warshall, iterative squaring, or successive adjacency-matrix multiplications. Since the order of events on those traces is irrelevant to fairness and co-fairness, we do not employ symbolic state variables and use just a single copy of event variables to carry out the computation, as illustrated in Figure 15. We note that in those algorithms we do not check whether we reach a fixed point in the computation. As a consequence, if using a SAT encoding, the resulting formulas may contain redundancies.

(2) The Boolean formula (G.2) contains an implicit iterator over all possible subsets L of $\Sigma_P \cup \{\tau\}$. In order to exclude the options of $L = \emptyset$ and $L = \{\tau\}$, we conjoin the formula with the restriction $\bigvee_{a \in \Sigma_P} a$.

(3) We need to also declare all infinite traces of P as co-fair in $\Sigma - \Sigma_P$. To do so, we add another Boolean conjunct $\bigwedge_{a \in (\Sigma - \Sigma_P)} \neg a$.

The Boolean encoding of $\Phi(P)$ is then as follows:

$$\Phi(P) = \left(\bigvee_{a \in \Sigma_P} a \right) \wedge \left(\bigwedge_{a \in (\Sigma - \Sigma_P)} \neg a \right) \wedge MaxSCC. \qquad \text{(G.3)}$$

The Key: PTIME Algorithms and Circuits. As stated earlier, given a process P and a non-empty set of events $L \subseteq \Sigma_P$, deciding whether or not to include $(L, \Sigma - L)$ in $\Phi(P)$ can be carried out in PTIME. Therefore, for the particular P and L, there exists a polynomial-size variable-free Boolean circuit that outputs true if and only if the pair $(L, \Sigma - L)$ is a fair/co-fair pair for P[15].

Let us fix P and let us observe, moreover, that the construction of the variable-free circuit does not depend on the particular choice of L (see (G.2) and (G.3)). Therefore, we can leave the Σ_P input gates of the circuit as Boolean variables [18]. What we obtain is

---

[15]This follows from the PTIME-hardness of CIRCUIT VALUE.

$$A = \begin{bmatrix} \text{false} & in \\ out & error \end{bmatrix}$$

$$\mathsf{Path} = \begin{bmatrix} (in \wedge out) & (in) \vee (in \wedge error) \\ (out) \vee (error \wedge out) & (error) \vee (out \wedge in) \vee (error \wedge error) \end{bmatrix}$$
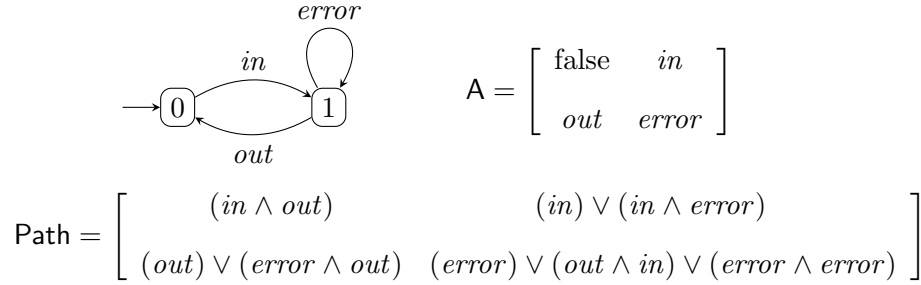
Figure 15: A symbolic representation of the adjacency matrix and the path matrix of the process *Send*. The path matrix is computed using successive matrix multiplications.

a compact circuit of size polynomial in the syntax of P that encodes the computation of $\Phi(P)$ once and for all possible inputs, i.e., for all exponentially many subsets of Σ_P. We remark that the size of the circuit is polynomial in the size of P's LTS, which in turn is polynomial in P's syntactic description, since we are dealing with sequential $\overline{\mathsf{SFS}}$ processes.

Since the circuit is of polynomial size, it can be turned into a polynomial-size (equi-satisfiable) Boolean formula using, e.g., Tseitin's encoding [27]. The circuit can be also turned into a BDD, in which case the size of the BDD could potentially blow up; however this is usually not the case in practice. Consequently, the Boolean formula or the BDD encoding of $\Phi(P)$ can be plugged into our compositional rules and be queried on demand when necessary, which fits very nicely into our symbolic framework.

G.1.2. *Encoding Compositional Rules.* The encoding of the rules for computing the livelock flag $\delta(P)$ and the collections of fair/co-fair pairs $\Phi(P)$ of a compound $\overline{\mathsf{SFS}}$ process P (see Theorem 6.1) are given in Figures 16 and 17 for Boolean formulas (i.e., for SAT) and in Figures 18 and 19 for BDDs.

$$\Phi(a \longrightarrow P)(y) \mathrel{\widehat{=}} \Phi(P)(y)$$

$$\Phi(P_1 \oplus P_2)(y) \mathrel{\widehat{=}} \Phi(P_1)(y') \wedge \Phi(P_2)(y'') \wedge \Big[\bigwedge_{a \in \Sigma} a(y) \leftrightarrow a(y') \vee \bigwedge_{a \in \Sigma} a(y) \leftrightarrow a(y'') \Big] \ \text{ if } \oplus \in \{\sqcap, \square, \fatsemi\}$$

$$\Phi(P_1 \underset{A}{\|} P_2)(y) \mathrel{\widehat{=}} \Phi(P_1)(y') \wedge \Phi(P_2)(y'') \wedge$$

$$\Big[\{ \bigwedge_{a \in A} \neg a(y') \wedge \bigwedge_{a \in \Sigma} a(y) \leftrightarrow a(y') \} \vee$$

$$\{ \bigwedge_{a \in A} \neg a(y'') \wedge \bigwedge_{a \in \Sigma} a(y) \leftrightarrow a(y'') \} \vee$$

$$\{ \bigwedge_{a \in \Sigma} a(y) \leftrightarrow (a(y') \vee a(y'')) \wedge \bigwedge_{a \in A} \neg a(y) \leftrightarrow (\neg a(y') \vee \neg a(y''))$$

$$\wedge \bigwedge_{a \in \Sigma \setminus A} \neg a(y) \leftrightarrow (\neg a(y') \wedge \neg a(y'')) \} \Big]$$

$$\Phi(P \setminus A)(y) \mathrel{\widehat{=}} \Phi(P)(y') \wedge \bigwedge_{a \in \Sigma \setminus A} a(y) \leftrightarrow a(y') \wedge \bigwedge_{a \in A} \neg a(y)$$

$$\Phi(P[R])(y) \mathrel{\widehat{=}} \Phi(P)(y') \wedge \bigwedge_{a \in \Sigma} [a(y') \rightarrow (\bigvee_{a\,R\,b} b(y))] \wedge \bigwedge_{b \in \Sigma} [(\bigwedge_{c\,R\,b} \neg c(y')) \rightarrow \neg b(y)]$$

Figure 16: SAT encoding of $\Phi(P)$.

$$\delta(P \setminus A) \mathrel{\widehat{=}} \delta(P) \vee \left(\neg \big[\Phi(P)(y) \rightarrow (\bigvee_{b \in \Sigma \setminus A} b(y)) \big] \text{ is SAT} \right)$$

Figure 17: SAT encoding of $\delta(P)$.

G.2. **The General Framework.** The BDD and SAT encodings of the rules for computing the nonexpansive, guard, contractive, and fair sets of CSP terms (see Theorems 5.2, 5.3, 5.4, and 5.5) are formalised similarly to the ones for the structurally finite-state processes. We illustrate the scheme and the use of two vectors of event variables by providing the BDD encoding of the rules for computing nonexpansive sets in Figure 20.

In the encoding, the vectors of Boolean variables x and y model, respectively, the U and V components of the pairs of sets of events. To understand the meaning of the encoding operators $\mathsf{UClosure}$, $\mathsf{DClosure}$, and $\mathsf{UDClosure}$, suppose the formula $\varphi(x, y)$ encodes the set of pairs of sets of events $A = \{(U, V) \mid \dots \}$ and the formula $\psi(y)$ encodes the set of sets of events $B = \{V \mid \dots \}$. Then the formulas $\mathsf{UClosure}(A)(x, y)$, $\mathsf{UDClosure}(A)(x, y)$, $\mathsf{UClosure}(B)(y)$, and $\mathsf{DClosure}(B)(y)$ encode, respectively, the sets $\{(U, V) \mid (U, V') \in A \wedge V' \subseteq V\}$, $\{(U, V) \mid (U', V') \in A \wedge U \subseteq U' \wedge V' \subseteq V\}$, $\{V \mid V' \in B \wedge V' \subseteq V\}$ and

$$\Phi(a \longrightarrow P)(y) \mathrel{\hat{=}} \Phi(P)(y)$$

$$\Phi(P_1 \oplus P_2)(y) \mathrel{\hat{=}} \Phi(P_1)(y) \ \wedge \ \Phi(P_2)(y) \ \text{ if } \oplus \in \{\sqcap, \square, \mathbin{\raise.3ex\hbox{$\scriptscriptstyle\circ$}\kern-.5ex\raise-.3ex\hbox{$\scriptscriptstyle\circ$}}\}$$

$$\Phi(P_1 \underset{A}{\|} P_2)(y) \mathrel{\hat{=}} \exists\, y' \, \exists\, y'' \boldsymbol{\cdot} \Phi(P_1)(y') \ \wedge \ \Phi(P_2)(y'') \ \wedge$$

$$\Big[\{ \bigwedge_{a \in A} \neg a(y') \ \wedge \ \bigwedge_{a \in \Sigma} a(y) \leftrightarrow a(y') \} \vee$$

$$\{ \bigwedge_{a \in A} \neg a(y'') \ \wedge \ \bigwedge_{a \in \Sigma} a(y) \leftrightarrow a(y'') \} \vee$$

$$\{ \bigwedge_{a \in \Sigma} a(y) \leftrightarrow (a(y') \vee a(y'')) \ \wedge \ \bigwedge_{a \in A} \neg a(y) \leftrightarrow (\neg a(y') \vee \neg a(y''))$$

$$\wedge \bigwedge_{a \in \Sigma \setminus A} \neg a(y) \leftrightarrow (\neg a(y') \wedge \neg a(y'')) \} \Big]$$

$$\Phi(P \setminus A)(y) \mathrel{\hat{=}} \big[\, \exists\, y_A \boldsymbol{\cdot} \Phi(P)(y) \big] \ \wedge \bigwedge_{a \in A} \neg a(y)$$

$$\Phi(P[R])(y) \mathrel{\hat{=}} \exists\, y' \boldsymbol{\cdot} \Phi(P)(y') \ \wedge \bigwedge_{a \in \Sigma} [a(y') \to (\bigvee_{a\,R\,b} b(y))] \ \wedge \bigwedge_{b \in \Sigma} [(\bigwedge_{c\,R\,b} \neg c(y')) \to \neg b(y)]$$

Figure 18: BDD encoding of $\Phi(P)$.

$$\delta(P \setminus A) \mathrel{\hat{=}} \delta(P) \ \vee \ \Big(\big[\Phi(P)(y) \to (\bigvee_{b \in \Sigma \setminus A} b(y))\big] \text{ is not valid} \Big)$$

Figure 19: BDD encoding of $\delta(P)$.

$\{V \mid V' \in B \wedge V \subseteq V'\}$:

$$\mathsf{UClosure}(A)(x,y) \mathrel{\hat{=}} \exists\, y' \boldsymbol{\cdot} \varphi(x,y') \wedge \bigwedge_i (y'_i \to y_i)$$

$$\mathsf{UDClosure}(A)(x,y) \mathrel{\hat{=}} \exists\, x'y' \boldsymbol{\cdot} \varphi(x',y') \wedge \bigwedge_i (x_i \to x'_i) \wedge \bigwedge_i (y'_i \to y_i)$$

$$\mathsf{UClosure}(B)(y) \mathrel{\hat{=}} \exists\, y' \boldsymbol{\cdot} \psi(y') \wedge \bigwedge_i (y'_i \to y_i)$$

$$\mathsf{DClosure}(B)(y) \mathrel{\hat{=}} \exists\, y' \boldsymbol{\cdot} \psi(y') \wedge \bigwedge_i (y_i \to y'_i)$$

$$\mathsf{N}_X(P)(x,y) \mathrel{\widehat{=}} \text{true} \quad \textbf{whenever } X \textbf{ is not free in } P\textbf{; otherwise:}$$

$$\mathsf{N}_X(a \longrightarrow P)(x,y) \mathrel{\widehat{=}} \mathsf{N}_X(P)(x,y)$$

$$\mathsf{N}_X(P_1 \oplus P_2)(x,y) \mathrel{\widehat{=}} \mathsf{N}_X(P_1)(x,y) \wedge \mathsf{N}_X(P_2)(x,y) \quad \textbf{if } \oplus \in \{\sqcap, \square, \mathbin{;}, \parallel_A\}$$

$$\mathsf{N}_X(P \setminus A)(x,y) \mathrel{\widehat{=}} \mathsf{UClosure}(\mathsf{N}_X(P)(x,y) \wedge \chi(\{V \mid V \subseteq \Sigma - A\})(y))$$

$$\mathrel{\widehat{=}} \mathsf{UClosure}(\mathsf{N}_X(P)(x,y) \wedge \mathsf{DClosure}(\Sigma - A)(y))$$

$$\mathsf{N}_X(P[R])(x,y) \mathrel{\widehat{=}} \mathsf{UClosure}(\exists y' \mathbin{\textbf{.}} \mathsf{N}_X(P)(x,y') \wedge \rho(y',y))$$

$$\mathsf{N}_X(X)(x,y) \mathrel{\widehat{=}} \bigwedge_i (x_i \rightarrow y_i)$$

$$\mathsf{N}_X(\mu Y \mathbin{\textbf{.}} P)(x,y) \mathrel{\widehat{=}} \mathsf{UDClosure}(\mathsf{N}_X(P)(x,y) \wedge \exists x' \mathbin{\textbf{.}} (\mathsf{N}_Y(P)(x',y) \wedge \bigwedge_i (x_i' \leftrightarrow y_i))) \quad \textbf{if } Y \neq X$$

Figure 20: BDD encoding of $\mathsf{N}_X(P)$.