A coinductive equational characterisation of trace inclusion for regular processes ⋆

Matthew Hennessy

Trinity College Dublin
matthew.hennessy@cs.tcd.ie

Abstract. In 1966 Arto Salomaa gave a complete axiomatisation of regular expressions. It can be viewed as a sound and complete proof system for regular processes with respect to the behavioural equivalence called *language equivalence*. This proof system consists of a finite set of axioms together with one inductive proof rule.
We show that the behavioural preorder called *language containment* or *trace inclusion* can be characterised in a similar manner, but using a coinductive rather than an inductive proof rule.

1 Introduction

In 1966 Arto Salomaa gave two complete axiomatisations for regular expressions; see [13, 8]. We concentrate on the first one where the key idea is the uniqueness of the solution of certain regular expression equations. This is recalled in Section 2 within the framework of *regular processes*, from [9]. We use a language for defining recursive processes of the form $\text{REC}x.t$ where the body t can be defined using *prefixing*, $a.u$, nondeterministic choice, $u_1 + u_2$, or a termination event $\mathsf{0}$; of course the body t may also contain further regular processes.

This language, referred to as rCCS, can be given various semantic interpretations, which can be expressed in terms of *behavioural equivalences* between processes. One such behavioural equivalence is called *language equivalence*, where each process p in rCCS is interpreted as a (regular) set of sequences of actions $\mathcal{L}(p)$, intuitively the sequences of actions it can perform. Then two processes are deemed to be *language equivalent*, written $p \equiv_{\mathcal{L}} q$ whenever $\mathcal{L}(p) = \mathcal{L}(q)$. This corresponds to *may equivalence* from [6] or *trace equivalence* from [7].

In this framework Salomaa's result, as formulated for example in [12], is a sound and complete proof system for determining when $p \equiv_{\mathcal{L}} q$. The proof system consists of

– simple proof rules for embodying the principle of *substitution of equals for equals*

⋆ This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero – the Irish Software Research Centre.

– a set of equations (or axiom schemas)
– an inductive proof rule for $\equiv_{\mathcal{L}}$ for regular processes, called *unique fixpoint induction*

Unique fixpoint induction is very intuitive:[1]

$$\frac{t\{x \mapsto q\} = q}{\text{REC}x.t = q} \ (\text{UFI})$$

It states that if a process q satisfies (semantically) the body of a recursive process then it is semantically equal to the recursive process itself.

Many other behavioural equivalences for regular processes can be captured in the same manner, simply by varying the equations. For example *strong bisimulation equivalence* and *weak bisimulation equivalence* are captured in this manner in [10, 11].

However many behavioural theories of processes are expressed in terms of behavioural preorders rather than equivalences. Typical examples include *refusals* [7], *must testing* [6], or the various contract preorders considered in [2]. It is unclear how Salomaa's proof system can be adapted for such behavioural preorders. In particular there is no known complete induction principle to replace unique fixpoint induction.

Here we consider a simple behavioural preorder, *language* or *trace inclusion*. Let $p \leq_{\mathcal{L}} q$ if $\mathcal{L}(p) \subseteq \mathcal{L}(q)$. Of course it is straightforward to establish for a particular pair of processes p, q whether or not $p \leq_{\mathcal{L}} q$ using Salomaa's proof system; it is sufficient to try to establish $p + q =_{\mathcal{L}} q$. But this does not in itself give a sound and complete proof system for the behavioural preorder $\leq_{\mathcal{L}}$ based on the ideas outlined above, namely

– simple proof rules for embodying the principle of *substitution of equals for equals*
– a set of inequations
– some inductive proof rule for $\leq_{\mathcal{L}}$ over regular processes.

This is the purpose of the current short paper. We show that by using a simple *coinductive* proof rule we can give such a sound and complete proof system for regular processes.

We now outline the remainder of the paper. In the next section we define formally the language of regular processes, and their semantics. We then outline the sound and complete proof system for language equivalence, based on an inductive proof rule. In Section 4 we outline our novel proof system, based on a set of standard inequations, together with one coinductive proof rule. Proving the soundness of the proof system is non-trivial, and is given in Section 5. The following section is devoted to completeness. The proof here depends on the fact that the set of *reachable states* of processes, in a novel interpretation as a labeled transition system, is finite. This topic is isolated in the independent Section 7. The paper ends with a short conclusion.

[1] For soundness the variable x in body t should be *guarded*.

$$\frac{}{\mu.p \xrightarrow{\mu} p}\ (\text{A-PRE}) \qquad \frac{}{\text{REC}x.t \xrightarrow{\tau} t\{x \mapsto \text{REC}x.t\}}\ (\text{REC})$$

$$\frac{p \xrightarrow{\mu} p'}{p + q \xrightarrow{\mu} p'}\ (\text{EXT-L}) \qquad \frac{q \xrightarrow{\mu} q'}{p + q \xrightarrow{\mu} q'}\ (\text{EXT-R})$$

Fig. 1. Operational semantics

2 Regular processes and language equivalence

The language of recursive terms is given by the following grammar:

$$\mathsf{rCCS}: \qquad t ::= 0 \mid \mu.t, \mu \in \mathsf{Act}_\tau \mid t_1 + t_2 \mid x \in \mathsf{Var} \mid \text{REC}x.t$$

where Act is a set of actions, ranged over by a, and Act_τ represents $\mathsf{Act} \uplus \{\tau\}$, where τ is a special symbol for an internal action. All occurrences of the variable x in t are *bound* in the term $\text{REC}x.t$, and this leads to the standard notion of *free* and *bound* variables. We are only interested in *closed terms*, those not containing any free variables, which we refer to as *processes*. For the sake of simplicity we will also assume that all terms of the form $\text{REC}x.t$ are *guarded*; that is every occurence of x in the body of the recursion t appears underneath an external prefix $a.-$.

The (standard) operational semantics of processes is given in Figure 1, with judgements for transitions of the form $p \xrightarrow{\mu} q$, where μ ranges over Act_τ. The rule (REC) uses the standard notion of substitution: in general $t\{x \mapsto p\}$ represents the result of substituting all free occurrences of the variable x in the term t by the closed term p. This may be defined by structural induction on t.

The transitions in Figure 1 are generalised to *weak transitions* of the form $p \overset{s}{\Longrightarrow} q$, where s ranges over $\mathsf{Act}^\star$ as follows:

- $p \overset{\varepsilon}{\Longrightarrow} p$
- $p \xrightarrow{a} p'$, $p' \overset{s}{\Longrightarrow} q$ imply $p \overset{as}{\Longrightarrow} q$
- $p \xrightarrow{\tau} p'$, $p' \overset{s}{\Longrightarrow} q$ imply $p \overset{s}{\Longrightarrow} q$

We use $p \overset{s}{\Longrightarrow}$ to indicate that for some q, $p \overset{s}{\Longrightarrow} q$.

Definition 1 (Language of a process). *For every $k \geq 0$ let $\mathcal{L}^k(p) = \{ s \in \mathsf{Act}^\star \mid p \overset{s}{\Longrightarrow}, |s| \leq k \}$, and let $\mathcal{L}(p) = \cup_{k \geq 0} \mathcal{L}^k(p)$. $\mathcal{L}(p)$ is refered to as the language of the process p, or it's set of traces.*

We write $p \leq_{\mathcal{L}} q$ if $\mathcal{L}(p) \subseteq \mathcal{L}(q)$, and $p \equiv_{\mathcal{L}} q$ if $\mathcal{L}(p) \subseteq \mathcal{L}(q)$ and $\mathcal{L}(q) \subseteq \mathcal{L}(p)$. □

3 The proof system for language equivalence

The proof system for language equivalence is given in Figure 2, with judgements are of the form $\vdash p = q$ where p, q are processes. A simple side-condition would be required on the rule (UFI), if we did not have our simplifying assumption that all recursive processes are guarded.

The rule (EQ) presupposes a set of equations **Eq** such as those in Figure 3. In general axioms take the form $T = U$ where T, U are words formed from the alphabet $\{ \mathsf{0}, \mu.-,\ -+- \}$ using axiom-variables $X, Y, \ldots$ taken from a set AVar. We say the pair $\langle p, p' \rangle$ is an instance of an equation, written $\langle p, p' \rangle \in \mathsf{Ins}(\mathbf{Eq})$, if there exists some axiom $T = U$ in **Eq** such that $p = \sigma(T)$, $p' = \sigma(U)$ where σ is an instantiation, that is a mapping from AVar to processes.

Let us write $\vdash_{eq} p = q$ if there is a proof of $\vdash p = q$ in the proof system using the equations in Figure 3. Those on the left hand side determine an idempotent commutative monoid; on the right hand side there is an axiom which says that τ transitions are essentially invisible, together with the distribution of prefixing over nondeterministic choice.

This proof system is both sound and complete with respect to language equivalence:

Theorem 1 (Salomaa, Rabinovich). *For all processes, $\vdash_{eq} p = q$ if and only if $p \equiv_{\mathcal{L}} q$.*

Proof. The proof for a corresponding property for regular expressions was given in [13]. This was adapted in [12] for a slight variation on our regular processes, using a proof technique from [10]. □

One could attempt to adapt this proof system to deal with language inclusion, with judgements of the form $\vdash p \leq q$; for example the set of equations could be replaced by *inequations*. However the major issue would be the replacement of the fixpoint rule (UFI) with a fixpoint rule for inequations which is sufficiently powerful to attain completeness.

In the next section we suggest an alternative approach.

4 The proof system for trace inclusion

This proof system has judgements of the form

$$A \vdash p \leq p'$$

where p, p' are processes and A is a *finite set* of assumptions, each of which takes the form $p_1 \leq p_2$. The rules for forming proof trees are given in Figure 4, many of which are straightforward adaptations of corresponding rules from Figure 2. We have (INEQ) for instantiating inequations and the rule (REC) from Figure 2 has been split into two rules, one for unfolding and the other for folding. There are also two obvious rules for managing assumptions, (HYP) and (W). The major

$$\frac{}{\vdash p = p}\ (\textsc{Id}) \qquad \frac{\vdash p_1 = p_2,\ \vdash p_2 = p_3}{\vdash p_1 = p_3}\ (\textsc{Tr})$$

$$\frac{\vdash p = p'}{\vdash p' = p}\ (\textsc{Sym}) \qquad \frac{\langle p, p' \rangle \in \mathsf{Ins}(\mathbf{Eq})}{\vdash p = p'}\ (\textsc{Eq})$$

$$\frac{\vdash p = p'}{\vdash p + q = p' + q}\ (\textsc{Pl}) \qquad \frac{\vdash p = p'}{\vdash a.p = a.p'}\ (\textsc{Pre})$$

$$\frac{}{\vdash \textsc{rec}x.t = t\{x \mapsto \textsc{rec}x.t\}}\ (\textsc{Rec}) \qquad \frac{\vdash t\{x \mapsto q\} = q}{\vdash \textsc{rec}x.t = q}\ (\textsc{Ufi})$$

Fig. 2. The proof system for language equivalence

$$X + X = X \qquad X + Y = Y + X \qquad \tau.X = X$$
$$X + (Y + Z) = (X + Y) + Z \qquad X + 0 = X \qquad a.(X + Y) = a.X + a.Y$$

Fig. 3. The equations for language equivalence

change is the replacement of the structural rule for prefixing, (Pre) in Figure 2, with the rule (coRec). Note that this can be viewed as a generalisation as in the new proof system the rule (Pre) can be derived:

$$\dfrac{\dfrac{A \vdash p \leq p'}{A, a.p \leq a.p' \vdash p \leq p'}\,\mathrm{W}}{A \vdash a.p \leq a.p'}\,\mathrm{CoRec}$$

We call this a *coinductive* rule because the conclusion of the rule is one of it's hypotheses. This of course makes it's soundness problematic; see the discussion in the next section.

Each equation in Figure 3 can be interpreted as two inequations. For example in place of idempotency $X + X = X$ we have the two inequations $X + X \leq X$ and $X \leq X + X$. In addition we need one new inequation:

$$X \leq X + Y \tag{1}$$

Let us write $\vdash_{leq} A \vdash p \leq p'$ to mean that there is a valid proof tree with conclusion $A \vdash p \leq p'$; that is a proof tree constructed using the rules in Figure 4, using the set of inequations just outlined. We abbreviate $\vdash_{leq} \emptyset \vdash p \leq p'$ to $\vdash_{leq} p \leq p'$. We also use $p \leq_{ineq} p'$ to mean that p may be rewritten to p' using this set of inequations. More specifically, in the rewriting all the rules in Figure 4 may be used, except (Hyp),(W) and (coREC).

Example 1. Let r_1, r_2 denote $\textsc{rec}x.a.x$, $\textsc{rec}x.a.a.x$ respectively. The following is a valid proof tree:

$$\frac{}{\vdash p \leq p}\ (\textsc{Id}) \qquad \frac{A \vdash p_1 \leq p_2,\ A \vdash p_2 \leq p_3}{A \vdash p_1 \leq p_3}\ (\textsc{Tr})$$

$$\frac{\langle p, p' \rangle \in \mathsf{Ins}(\mathbf{InEq})}{A \vdash p \leq p'}\ (\textsc{inEq})$$

$$\frac{A \vdash p \leq p'}{A \vdash p + q \leq p' + q}\ (\textsc{Pl}) \qquad \frac{A, a.p \leq a.p' \vdash p \leq p'}{A \vdash a.p \leq a.p'}\ (\textsc{coRec})$$

$$\frac{}{\vdash \textsc{rec}x.t \leq t\{x \mapsto \textsc{rec}x.t\}}\ (\textsc{Ufd}) \qquad \frac{}{\vdash t\{x \mapsto \textsc{rec}x.t\} \leq \textsc{rec}x.t}\ (\textsc{Fld})$$

$$\frac{}{p \leq p' \vdash p \leq p'}\ (\textsc{Hyp}) \qquad \frac{B \vdash p \leq p',\ A \subseteq B}{AS \vdash p \leq p'}\ (\textsc{W})$$

Fig. 4. The proof system

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a.r_1 \leq a.a.r_2,\ a.r_1 \leq a.r_2 \vdash a.r_1 \leq a.a.r_2}\,(\textsc{Hyp,W})}{a.r_1 \leq a.a.r_2,\ a.r_1 \leq a.r_2 \vdash r_1 \leq r_2}\,(\textsc{Tr,Fld,Ufd})}{a.r_1 \leq a.a.r_2 \vdash a.r_1 \leq a.r_2}\,(\textsc{coRec}) \quad \dfrac{}{\vdash r_1 \leq a.r_1}\,(\textsc{Ufd})}{a.r_1 \leq a.a.r_2 \vdash r_1 \leq a.r_2}\,(\textsc{Tr}) \quad \dfrac{}{\vdash a.a.r_2 \leq r_2}\,(\textsc{Ufd})}{\vdash a.r_1 \leq a.a.r_2}\,(\textsc{coRec}) \quad \dfrac{}{\vdash r_1 \leq a.r_1}\,(\textsc{Ufd})}{\vdash r_1 \leq r_2}\,\textsc{Tr}$$

This means that $\vdash_{leq} \textsc{rec}x.a.x \leq \textsc{rec}x.a.a.x$. □

5 Soundness

To prove soundness of the proof system we need a semantic interpretation of the judgements $A \vdash p \leq p'$ which is preserved by all instances of the proof rules. There is an obvious choice, which is however unsound.

Example 2. Let us write

$$p_1 \leq p_1', \ldots p_k \leq p_k' \models^w p \leq p', \text{ for } k \geq 0,$$

if $p_1 \leq_{\mathcal{L}} p_1'$, $\ldots p_k \leq_{\mathcal{L}} p_k'$ implies $p \leq_{\mathcal{L}} p'$.

Unfortunately this is not preserved by the rule (coRec). An instance of this rule is

$$\frac{a.b.\,0 \leq a.\,0 \vdash b.\,0 \leq 0}{\vdash a.b.\,0 \leq a.\,0}$$

Note that the premise is (vacuously) semantically valid, $a.b.\,0 \leq a.\,0 \vDash^w b.\,0 \leq 0$, because $a.b.\,0 \not\leq_{\mathcal{L}} a.\,0$. However the conclusion is not semantically valid, $\not\vDash^w a.b.\,0 \leq a.\,0$, because $a.b.\,0 \not\leq_{\mathcal{L}} a.\,0$. □

Instead, as in [3], we base our semantic interpretation on a *stratified* characterisation of language inclusion.

Definition 2 (Semantic interpretation). *For $n \geq 0$ write*

$$p_1 \leq p_1', \ldots, p_k \leq p_k' \vDash_n p \leq p'$$

if $\mathcal{L}^n(p_1) \subseteq \mathcal{L}^n(p_1'), \ldots, \mathcal{L}^n(p_k) \subseteq \mathcal{L}^n(p_k')$ implies $\mathcal{L}^n(p) \subseteq \mathcal{L}^n(p')$.

We use $A \vDash p \leq p'$ to mean that $A \vDash_n p \leq p'$ for every $n \geq 0$. □

The counterexample given above no longer works for this stratified semantic interpretation. This is because

$$a.b.\,0 \leq a.\,0 \not\vDash b.\,0 \leq 0$$

In particular $a.b.\,0 \leq a.\,0 \not\vDash_1 b.\,0 \leq 0$ because $\mathcal{L}^1(a.b.\,0) \subseteq \mathcal{L}^1(a.\,0)$ but $\mathcal{L}^1(b.\,0)$ is not a subset of $\mathcal{L}^1(0)$.

Theorem 2 (Soundness). *$\vdash_{leq} A \vdash p \leq p'$ implies $A \vDash p \leq p'$.*

Proof. It suffices to show that each of the proof rules in Figure 4 preserves the semantics. The only non-trivial case is the rule (coRec).

So suppose $A, a.p \leq a.p' \vDash p \leq p'$; that is

$$A, a.p \leq a.p' \vDash_k p \leq p' \qquad \text{for all } k \geq 0 \tag{2}$$

We have to show that from this hypothesis, which we refer to as the *outer hypothesis*, the conclusion $A \vDash a.p \leq a.p'$ follows. In particular we show that $A \vDash_n a.p \leq a.p'$, for every $n \geq 0$, by induction on n.

The base case, when $n = 0$, is straightforward, as $\mathcal{L}^0(r) = \{\varepsilon\}$ for any process r.

In the inductive case we let $n = (m+1)$, and we can assume

$$A \vDash_m a.p \leq a.p' \tag{3}$$

which we refer to as the *inner hypothesis*. We have to deduce $A \vDash_{(m+1)} a.p \leq a.p'$.

To this end suppose $\mathcal{L}^{(m+1)}(q) \subseteq \mathcal{L}^{(m+1)}(q')$ for every $q \leq q' \in A$. We have to show $\mathcal{L}^{(m+1)}(a.p) \subseteq \mathcal{L}^{(m+1)}(a.p')$.

First we apply the inner hypothesis (3): this is possible since $\mathcal{L}^{(m+1)}(q) \subseteq \mathcal{L}^{(m+1)}(q')$ implies $\mathcal{L}^m(q) \subseteq \mathcal{L}^m(q')$. So we obtain $\mathcal{L}^m(a.p) \subseteq \mathcal{L}^m(a.p')$.

With this we can apply the outer hypothesis (2) with $k = m$. We obtain $\mathcal{L}^m(p) \subseteq \mathcal{L}^m(p')$, from which the required $\mathcal{L}^{(m+1)}(a.p) \subseteq \mathcal{L}^{(m+1)}(a.p')$ follows. □

In particular this soundness result means that if we can construct a valid proof tree for the judgement $\vdash p \leq p'$ then $p \leq_{\mathcal{L}} p'$:

Corollary 1. *$\vdash_{leq} p \leq p'$ implies $p \leq_{\mathcal{L}} p'$.*

Proof. Suppose $\vdash_{leq} p \leq q$, that is $\emptyset \vdash p \leq q$. By Theorem 2 we have that $\mathcal{L}^n(p) \subseteq \mathcal{L}^n(q)$ for all $n \geq 0$. This means that $\mathcal{L}(p) \subseteq \mathcal{L}(q)$ and therefore by definition $p \leq_{\mathcal{L}} q$. □

6 Completeness

The proof of completeness is constructive; we design an algorithm for constructing valid proof trees. To describe the algorithm we need to introduce some notation.

Definition 3 (Head normal forms). *A process of the form* $\sum_{a\in A} a.p_a$*, where* A *is a finite subset of* Act *is said to be a* head normal form*, abbreviated to* hnf. □

Proposition 1. *For every process* p *there exists some head normal form,* $\textsc{hnf}(p)$*, such that* $p =_{ineq} \textsc{hnf}(p)$*.*

Proof. See the appendix. The proof relies on the fact that all proceses are *guarded*. □

It will also be convenient at some point to work with processes up to the equivalence generated by three axioms from Figure 3; that is the commutativity and associativity of $+$ together with idempotency. Let $[p]$ denote the equivalence class of all processes equivalent to p. However rather than manipulating these sets of processes we will use particular representatives. We use $(p)_r$ to refer to any actual process in the set $[p]$, for which the idempotency axiom $X+X = X$ cannot be applied to it from left to right. Thus it will take the form $s_1 + s_2 \ldots + s_n$ where each of the processes s_i are syntactically different. We call such processes *reduced*.

The algorithm also uses the three following derived proof rules:

$$\frac{A \vdash p_1 \leq q,\ A \vdash p_2 \leq q}{A \vdash p_1 + p_2 \leq q}\ (\textsc{PlusL}) \qquad \frac{A \vdash p \leq q_1}{A \vdash p \leq q_1 + q}\ (\textsc{PlusR}q)$$

$$\frac{}{A \vdash \mathsf{0} \leq q}\ (\textsc{Zero}q)$$

We leave the reader to show that these can be derived from the rules in Figure 4. All use the transitivity rule (Tr). The derivation of (PlusL) uses two applications of (Pl), and an application of the inequation $X+X \leq X$. That of $(\textsc{Plus}q)$ uses an application of the new inequation (1) above; this is also required in the derivation of $(\textsc{Zero}q)$, in addition to the inequation $\mathsf{0} \leq \mathsf{0} + X$.

The pseudo-code for the algorithm $C(A, p, q)$ is given in Figure 5. It takes as parameters A, a finite set of premises of the form $p_i \leq q_i$, and a pair of processes p, q. It returns with

- **FAIL**, indicating that $p \not\leq_{\mathcal{L}} q$,
- or a proof tree T, which is a valid proof tree for the judgement $A \vdash p \leq q$.

The code is executed by matching the actual parameters sequentially against the patterns on the left hand side in Figure 5; each of the possible five patterns may be considered as *rules* for matching the actual parameters. The first call transforms the parameters p, q into head normal forms. The remainder can be

```
1  C(A,p,q) ⇒ if (p or q not in hnf)
2                 then
3                    let T =  C(A,hnf(p),hnf(q))
4                    in return  (T;(HNF))
5                 else
6  C(A,0,q) ⇒ return Zeroq
7  C(A, p_1 + p_2,q) ⇒ let T_1 = C(A,p_1,q)
8                                  let T_2  = C(A,p_2,q)
9                               in
10                              return (T_1,T_2);(PlusL)
11 C(A,a.p, a.q + r)  ⇒ if a.p ≤a.q in A then return (HYP;PlusRr)
12                       else let B = {A, a.p<a.q}
13 corec                        let T = C(B,(p)_r,(q)_r)
14                              in
15                              T;(coRec);(PlusRr)
16 S(A,a.p,q)            ⇒ return FAIL
```

Fig. 5. The algorithm

considered as a case analysis on the structure of p, which when line 6 is reached is guaranteed to be in head normal form. Note that if the final rule, on line 16, is ever fired then we know that q, which is a hnf, does not have an a transition, and therefore we can conclude the $a.p \not\leq_{\mathcal{L}} q$.

The non-trivial rule is on line 11. Here both the processes being analysed have a transitions. Moreover because they are hnfs we know r does not have an a transition. If the assumption $a.p \leq a.q$ is already available in A then the required proof tree is readily constructed. Otherwise this assumption is added to A to get the set of assumptions B, and a proof tree T is constructed for the judgement $B \vdash (p)_r \leq (q)_r$. The returned proof tree for the judgement $A \vdash p \leq q$ is then constructed using T, with an instance of the coinductive rule (coRec), together with the derived rule (PlusRr) We elide the transformation of $(p)_r, (q)_r$ into the original parameters p, q respectively, but the use of these reduced processes will be important in showing that the algorithm terminates. For the purposes of later discussions we label this recursive call which constructs the proof tree T with *corec*.

Note that in order to simplify the pseudo-code we have assumed that occurrences of **FAIL** are percolated upwards through the code. For example on line 8 if the innner call to $C(A, p_2, q)$ returns **FAIL** then **FAIL** is also returned by the outer call $C(A, p_1 + p_2, q)$.

Execution of the code for given parameters, $C(A, p, q)$ consists of a sequence of recursive calls $C(A_i, p_i, q_i)$ until at some point a base case, such as on lines 6, or 11, or 16, is reached. In order to analyse the behaviour of the algorithm we introduce some notation for describing these sequences.

Definition 4 (Call trees). *Let us write*

$$C(A,p,q) \mapsto C(A',p',q')$$

if executing $C(A,p,q)$ leads directly to a recursive call to $C(A',p',q')$. The call tree *of $C(A,p,q)$ is defined to be the tree with root labelled by $C(A,p,q)$ with sub-trees consisting of all the call trees of the recursive calls $C(A',p',q')$ such that $C(A,p,q) \mapsto C(A',p',q')$. Note that in these trees the out-degree of each node is at most 2. A recursive call matching line 7 generates two sub-nodes; all other recursive calls generates at most one.*

A call path *for $C(A,p,q)$ is a path (finite or infinite) in the call tree of $C(A,p,q)$ starting with the root.* □

$$\frac{}{\mu.t \xrightarrow{\mu} t}\ (\textsc{a-Pre}) \qquad \frac{}{\textsc{rec}x.t \xrightarrow{\tau} t\{x \mapsto \textsc{rec}x.t\}}\ (\textsc{rec})$$

$$\frac{t \xrightarrow{\tau} t'}{t+u \xrightarrow{\tau} t'+u}\ (\textsc{tau-l}) \qquad \frac{u \xrightarrow{\tau} u'}{t+u \xrightarrow{\tau} t+u'}\ (\textsc{tau-r})$$

$$\frac{t \xrightarrow{a} t',\ u \not\xrightarrow{a},\ u \not\xrightarrow{\tau}}{t+u \xrightarrow{a} t'}\ (\textsc{ext-l}) \qquad \frac{u \xrightarrow{a} u',\ t \not\xrightarrow{a},\ t \not\xrightarrow{a}}{t+u \xrightarrow{\tau} u'}\ (\textsc{ext-r})$$

$$\frac{t \xrightarrow{a} t',\ u \xrightarrow{a} u'}{t+u \xrightarrow{a} t'+u'}\ (\textsc{ext})$$

Fig. 6. Towards hnfs

Proposition 2 (Algorithmic correctness). *Suppose $C(A,p,q)$ terminates.*

(i) If it returns **FAIL** *then $p \not\leq_{\mathcal{L}} q$.*
(ii) If it returns a proof tree, then this is a valid proof tree for the judgement $A \vdash p \leq q$.

Proof. In each case the proof is by induction on the number of recursive calls to $C(-,-,-)$.

(i) **FAIL** can be returned on any one of the lines 3,7,8,13, or 16.
If it is the last then p has the form $a.p'$ and moreover, because there was no match on line 11, we also know that q, which is a hnf, does not have an a derivative. Consequently $a.p' \not\leq_{\mathcal{L}} q$.
Suppose it is on line 13, because the recursive call $C(B,(p')_r,(q')_r)$ returns **FAIL**, in which case p,q have the form $a.p', a.q'+r$. By induction we know that $(p')_r \not\leq_{\mathcal{L}} (q')_r$, that is $p' \not\leq_{\mathcal{L}} q'$. Since $a.q'+r$ is a hnf we know that r does not have an a derivative, and therefore it follows that $a.p' \not\leq_{\mathcal{L}} a.q'+r$.
The other cases are handled in a similar manner.

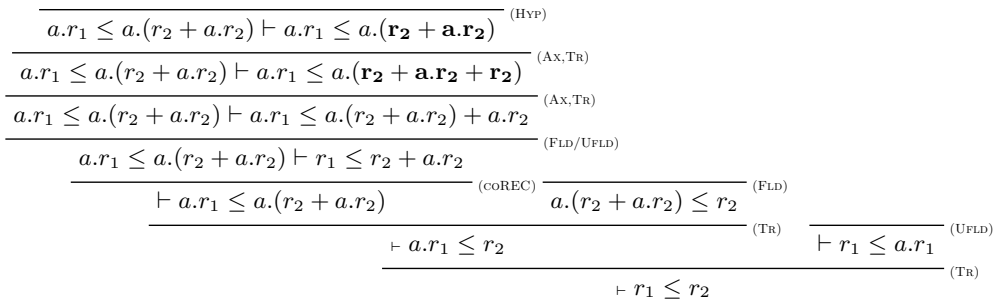

Fig. 7. $r_1 = \text{REC}x.a.x,\ r_2 = \text{REC}x.a.(x + a.x)$

(ii) A proof schema can be returned on any of the lines 4, 6, 10, 11, or 15. In each case the proof consists in checking that the returned schema is indeed a valid proof of the judgement $A \vdash p \leq q$, if necessary by invoking induction. □

The main difficulty in proving that the algorithm always terminates is to characterise the parameters which can be used in a call path from $C(A, p, q)$. This characterisation is complicated by the use of head normal forms in the code. We can capture their use via a relation $t \xrightarrow{\mu} t'$ defined by the rules in Figure 6. Note that for reasons which will be come apparent presently this relation is defined over arbitrary process terms, rather than simply closed terms, as used in Figure 1. So in the rule (Rec) we assume the standard notion of general substitution of (open) terms for variables, which may involve applications of α-conversion in order to avoid free variables being captured.

Proposition 3. *Suppose* $\text{HNF}(p) \xrightarrow{a} q$. *Then* $p \xrightarrow{\tau} \!\!\!\!\rightarrow^{\star} \xrightarrow{a} \!\!\!\!\rightarrow q$.

Proof. See the appendix. □

Let $\mathsf{Reach}(t) = \{\, u \mid t \xrightarrow{s} \!\!\!\!\rightarrow u,\ s \in \mathsf{Act}^{\star} \,\}$. In general $\mathsf{Reach}(t)$ is not finite.

Example 3. Consider $r_2 = \text{REC}x.a.(x + a.x)$. Then $\mathsf{Reach}(r_2)$ contains all processes of the form $r_2 + \sum_{1 \leq i \leq n} u_i$ where each u_i is the process $r_2 + a.r_2$; therefore $\mathsf{Reach}(r_2)$ is infinite.

This explains the use of the function $(-)_r$ in line 13 of the algorithm in Figure 5. Without the application of this function one can check that a call to $C(\emptyset, r_1, r_2)$, where r_1 denotes $\text{REC}x.a.x$, would not terminate. However with the use of $(-)_r$ one can check that $C(\emptyset, r_1, r_2)$ terminates after six recursive calls.

Moreover in Figure 7 we have constructed a valid proof tree for the judgement $\vdash r_1 \leq r_2$, although some abbreviations are used. We have indicated in bold font an essential use of the idempotency axiom $X = X + X$. □

Definition 5. *Let* $t \stackrel{\mu}{\Longrightarrow} u$ *if* $t \stackrel{\mu}{\Rightarrow} u'$ *for some* u' *such that* $u = (u')_r$. *Thus if* $t \stackrel{\mu}{\Longrightarrow} u$ *the rules in Figure 6 are used to find a* u' *such that* $t \stackrel{\mu}{\Rightarrow} u'$, *and then* u' *is reduced to* u. *Let* $\mathsf{rReach}(t) = \{\, u \mid t \stackrel{s}{\Longrightarrow} u,\, s \in \mathsf{Act}^\star \,\}$. □

It is easy to check that $\mathsf{rReach}(r_2)$, where r_2 is defined in Example 3, is the finite set $\{\, r_2,\, r_2 + a.r_2,\, a(r_2 + a.r_2),\, a.(r_2 + a.r_2) + a.r_2 \,\}$. This is a particular instance of a general phenomenon:

Theorem 3. *For every term* t, *the set* $\mathsf{rReach}(t)$ *is finite.*

Proof. See the next section. □

In the sequel we use $\mathsf{Act}(p)$ to denote the (finite) set of actions from Act which appear in the process p.

Proposition 4. *Suppose*

$$C(A_0, p_0, q_0) \mapsto C(A_1, p_1, q_1) \mapsto \ldots \mapsto C(A_k, p_k, q_k), k \geq 0$$

is an arbitrary call path. Then

(1) $A_i \subseteq A_{i+1}$, $\mathsf{Act}(p_n) \subseteq \mathsf{Act}(p_0)$
(2) If none of the recursive calls $C(A_k, p_k, q_k)$ *triggers the rule labelled corec, on line 13 in Figure 5, then there exists some bound* K *such that* $k \leq K$.
(3) If $p \leq q \in A_k$ *then either* $p \leq q \in A_0$ *or* p, q *have the form* $a.p', a.q'$ *respectively, where* $a \in \mathsf{Act}(p)$ *and* $p' \in \mathsf{rReach}(p_0)$, $q' \in \mathsf{rReach}(q_0)$.

Proof. The statement (1) follows by an analysis of the pseudo-code in Figure 5. First note that nowhere is the set of assumptions A_k decreased. Only in one place, line 12, is it changed; it is augmented. Secondly note that $\mathsf{Act}(\textsc{hnf}(p)) = \mathsf{Act}(p)$, and therefore by code one can check that $\mathsf{Act}(p_{n+1}) \subseteq \mathsf{Act}(p_n)$.

Similarly (2) follows by an analysis of the code.

Part (3) is proved by induction on the number of i, $0 \leq i \leq k$ such that $A_{i+1} \neq A_i$. We look at the first step, the least i such that $C(A_{i-1}, p_{i-1}, q_{i-1}) \mapsto C(A_i, p_1, q_1)$ where $A_i \neq A_0$.

This call must be as a result of matching the rule labelled *corec* on line 13 in Figure 5. So p_{i-1}, q_{i-1} must have the form $a.p'$, $a.q' + r$, and p_i, q_i the form p', q', and A_i must be $A_0 \uplus \{\, a.p' \leq a.q' \,\}$.

From part (1) we immediately have that $a \in \mathsf{Act}(p_0)$. Moreover all preceeding recursive calls must have either matched line 3, transforming p_0, q_0 to hnfs, or successive matches to line 7. Therefore $hnf(p)$ has the form $a.p' + \ldots$ and $hnf(q)$ has the form $a.q' + r$. It now follows from Proposition 3 that $p' \in \mathsf{rReach}(p_0)$ and $q' \in \mathsf{rReach}(q_0)$, as required. □

Theorem 4 (Termination). *The recursive procedure* $C(A, p, q)$ *terminates for all parameters* A, p, q.

Proof. Suppose

$$C(A,p,q) = C(A_0,p_0,q_0) \mapsto \ldots \mapsto C(A_k,p_k,q_k) \mapsto \ldots \tag{4}$$

is an arbitrary call path, finite or infinite.

First consider any step $C(A_n,p_n,q_n) \mapsto C(A_{n+1},p_{n+1},q_{n+1})$ resulting from a successful match to line 13 in the algorithm, which we have labelled *corec*. We know that p_n, q_n have the form $a.p'$, $a.q'+r$ respectively and p_{n+1}, q_{n+1} are p', q'. Because the test on line 11 failed we have that $A_{n+1} = A_n \uplus \{\, a.p' \leq a.q' \,\}$. By Proposition 4(3) $p' \in \mathsf{rReach}(p_0)$, $q' \in \mathsf{rReach}(q_0)$ and $a \in \mathsf{Act}(p)$. Obviously $\mathsf{Act}(p)$ is a finite set, as are $\mathsf{rReach}(p_0)$, $\mathsf{rReach}(q_0)$ from Theorem 3. Therefore there exists some k such that for all $i \geq k$ $A_i = A_k$.

It follows that in the sequence (4) above the rule labelled *corec* on line 13 can only be called a finite number of times. By part (2) of Proposition 4 we have that the sequence (4) can only be finite. □

We can now conclude with the main result of the paper:

Corollary 2 (Soundness and Completeness). *$\vdash_{leq} p \leq q$ if and only if $p \leq_{\mathcal{L}} q$.*

Proof. One direction, Soundness, follows from Corollary 1.

Conversely suppose $p \leq_{\mathcal{L}} q$. We know from Theorem 4 that the algorithm $C(\emptyset,p,q)$ terminates. By design this algorithm either returns **FAIL** or a proof tree. By algorithmic correctness, Proposition 2, the former is not possible; the same proposition ensures that the returned proof tree is a valid proof tree for $\emptyset \vdash p \leq q$. That is $\vdash_{leq} p \leq q$. □

7 Finite Reachability

We prove Theorem 3 by giving an over-approximation to the set of terms reachable from an arbitrary term t. The definition is by structural induction on t, and by construction the resulting set is obviously finite.

Definition 6 (Over-approximation). *For every term t the set of* approximants *$t^\star$ is defined as follows:*

(i) $\mathsf{0}^\star = \{\, \mathsf{0} \,\}$, $x^\star = \{\, x \,\}$
(ii) $(\mu.t)^\star = \{\, \mu.t \,\} \cup \{\, \mu.t' \mid t' \in t'^\star \,\}$
(iii) $(t_1 + t_2)^\star = t_1^\star \cup t_2^\star \cup \{\, t_1' + t_2' \mid t_i \in t_i^\star \,\}$
(iv) $(\mathrm{REC}x.t')^\star = \{\, \mathrm{REC}x.t' \,\} \cup \{\, \Sigma(S) \mid S \subseteq T \,\}$,
where $T = \{\, t''\{x \mapsto \mathrm{REC}x.t'\} \mid t'' \in t'^\star$ *or* $t'' + x \in t'^\star \,\}$ *and for any set of terms* $S = \{\, s_1, s_2, \ldots s_n \,\}$, $\Sigma(S)$ *denotes the term* $s_1 + s_2 + \ldots + s_n$. □

Lemma 1. *For every t, the set $t^\star$ is finite.*

Proof. By structural induction on t. □

The proof that $\mathsf{rReach}(t) \subseteq t^*$ is also by structural induction on t and most of the cases are straightforward. For example the case when t has the form $t_1 + t_2$ is handled by the following lemma. Here, and in subsequent proofs we ignore the sequence of actions performed by terms, writing $t \twoheadrightarrow^\star t'$ in place of $t \xrightarrow{s}\!\!\!\!\twoheadrightarrow t'$, or sometimes $t \twoheadrightarrow^k t'$ when we know that there are k steps in the derivation. We also use some standard notion of the size of such a derivation.

Lemma 2. *Suppose $t_1 + t_2 \twoheadrightarrow^\star u$, with a derivation of size n. Then*

1. *$t_1 \twoheadrightarrow^\star u$, with a derivation of size less then n*
2. *$t_2 \twoheadrightarrow^\star u$, with a derivation of size less then n*
3. *or $u = u_1 + u_2$ where $t_i \twoheadrightarrow^\star u_i$, each also having a derivation size less than n.*

Proof. A straightforward induction on the length of the derivation $t_1 + t_2 \twoheadrightarrow^\star u$ and a case analysis of why $t_1 + t_2 \twoheadrightarrow u$. □

The most difficult case of $\mathsf{rReach}(t) \subseteq t^*$ to treat is when t has the form $\textsc{rec}x.u$. In general a sequence of transitions takes the form

$$\textsc{rec}x.u \twoheadrightarrow u\{x \mapsto \textsc{rec}x.u\} \twoheadrightarrow \ldots \twoheadrightarrow t'$$

Therefore in order to understand the forms that t' can take we need to characterise the derivatives of $u\{x \mapsto r\}$ in terms of those of u and r.

Definition 7. *We define the predicate $t \downarrow x$ by structural induction on t, as follows:*

(i) $\textsc{rec}y.u \downarrow x$, *for all x and y*
(ii) $t_1 \downarrow x$, $t_2 \downarrow x$ *implies* $t_1 + t_2 \downarrow x$
(iii) $\mu.u \downarrow x$ *for every* $\mu \in \mathsf{Act}_\tau$.

We use $t \uparrow$ to mean that $t \downarrow$ is not true. □

Intuitively $t \downarrow x$ means that r does not play any role in any transition from $t\{x \mapsto r\}$. This is captured in the first part of the following proposition.

Proposition 5. *Suppose $t\{x \mapsto r\} \xrightarrow{\mu}\!\!\!\!\twoheadrightarrow u$, where t is reduced. Then if t is different from x, one of the following holds:*

(i) $t \downarrow x$ *and* $u = t'\{x \mapsto r\}$ *where* $t \xrightarrow{\mu}\!\!\!\!\twoheadrightarrow t'$
(ii) $t \uparrow x$, $t = t_1 + x$, t_1 *is reduced, and*
  (a) $u = t_1\{x \mapsto r\} + r'$ *where* $r \xrightarrow{\mu}\!\!\!\!\twoheadrightarrow r'$
  (b) $u = t_1'\{x \mapsto r\} + r$ *where* $t_1 \xrightarrow{\mu}\!\!\!\!\twoheadrightarrow t_1'$
  (c) $u = t_1'\{x \mapsto r\} + r'$ *where* $t_1 \xrightarrow{\mu}\!\!\!\!\twoheadrightarrow t_1'$, $r \xrightarrow{\mu}\!\!\!\!\twoheadrightarrow r'$

Proof. By structural induction on t, with an intricate case analysis. □

Proposition 6. *Suppose $t\{x \mapsto r\} \twoheadrightarrow^\star (u)_r$, with a derivation of size k. Then u has one of the following forms:*

(i) $t'\{x \mapsto r\}$, *where* $t \twoheadrightarrow^\star t'$ *has a derivation of size less then* k
(ii) $\sum_{1 \le i \le n} r'_i$, *where each* $r \twoheadrightarrow^\star r'_1$ *has a derivation of size less than* k
(iii) $t'\{x \mapsto r\} + \sum_{1 \le i \le n} r'_i$, *where the derivations* $t \twoheadrightarrow^\star t' + x$ *and* $r \twoheadrightarrow^\star r'_i$ *again have smaller size.*

Proof. By induction on the size of the derivation k. In the general case the transitions have the form

$$t\{x \mapsto r\} \twoheadrightarrow^\star u' \xrightarrow{\mu} (u)_r$$

where u' is reduced. Induction can be applied to the derivation $t\{x \mapsto r\} \twoheadrightarrow^\star u''$, to give three possibilities for the structure of u'', (i), (ii), (iii) above. In case (i) we apply Proposition 5 to the transition $u' \xrightarrow{\mu} u$. The case (ii) is a straightforward argument, while (iii) is a combination of the first two cases. □

Theorem 5. *For every term* t, $\mathsf{rReach}(t) \subseteq t^\star$.

Proof. By structural induction on t. We show that if $t \twoheadrightarrow^\star u$ then $u \in t^\star$.

The cases when t has one of the forms x, $\mathsf{0}$ are trivial, while when it is of the form $\mu.t'$ a very simple inductive argument suffices. When it has the form $t_1 + t_2$ an inductive argument is also used, supported by Lemma 2. We look briefly at the final and most difficult, case when it has the form $\textsc{rec}x.t_1$.

Here we use an inner induction on the size of the derivation $\textsc{rec}x.t_1 \twoheadrightarrow^\star u$. If u is $\textsc{rec}x.t_1$, that is the length of the derivation is zero, then the result is immediate as by definition $\textsc{rec}x.t_1 \in (\textsc{rec}x.t_1)^\star$. Otherwise we have

$$\textsc{rec}x.t_1 \xrightarrow{\tau} t_1\{x \mapsto r\} \twoheadrightarrow^\star u$$

where r denotes $\textsc{rec}x.t_1$, and we can read off the possible structure of u from Proposition 6. There are three possibilities, and we examine the third when u has the form

$$t'\{x \mapsto r\} + \sum_{1 \le i \le n} r'_i$$

where $t \twoheadrightarrow^\star t' + x$, $r \twoheadrightarrow^\star r'_i$, and each of these derivations being smaller in size than the original one.

Using the inner induction we have $r_i \in (\textsc{rec}x.t_1)^\star$ for each $1 \le i \le n$. Using the other (structural) induction we have $t' + x \in t_1^\star$, and therefore by definition $t'\{x \mapsto r\} \in (\textsc{rec}x.t_1)^\star$.

It follows that $u \in (\textsc{rec}x.t_1)^\star$, since this is defined so that $u_i \in (\textsc{rec}x.t_1)^\star$, $1 \le i \le n$, implies $u_1 + \ldots + u_n \in (\textsc{rec}x.t_1)^\star$. □

8 Conclusions

We have given a novel sound and complete proof system for trace inclusion of regular processes. The novel rule of the proof system is co-inductive in nature,

in that the conclusion of the rule is already one of it's hypotheses. Proof of soundness is based on a technique used in [3] for a proof system for recursive types, while completeness is demonstrated constructively; an algorithm is given which constructs a proof for every semantically valid judgement. Intuitively the algorithm works by *on the fly* determinising the processes, and systematically comparing their a-derivatives, for each action a from Act. The proof that the algorithm actually terminates is conceptually straightforward, but syntactically intricate. It relies on the fact the set of reachable states from a given process is finite, modulo a structural equivalence. A similar result is proved in [5] for the language of regular expressions, where the equivalence used, between regular expressions, is *semantic identity*. An alternative approach to proving termination of our algorithm might be based on defining a relation between our semantics for regular processes, and the derivatives of regular expressions given in [5].

We believe that our proof system can be adapted to a range of semantic preorders between regular processes, such as the testing preorders of [6]. Of particular interest are the contract preorders from [4, 2], and variations thereof. Such preorders often have alternative characterisations, often expressed in terms of intricate behavioural properties of processes; as an example see Definition 6 of [1]. It would be instructive to instead characterise these preorders over regular processes using variations on our proof system; the rules, including (coRec), would remain but the set of inequations used would depend on the particular contact preorder in mind.

Acknowledgements: The author would like to thank the anonymous referees, and Giovanni Bernardi, for their useful comments on a previous draft.

A Some proofs

Guarded terms: A variable x is *guarded* in the term t if each free occurrence of x in t occurs underneath a prefix $a.-$. A recursion $\text{REC}x.t$ is *guarded* if x is guarded in t. Finally a term u is a *guarded term* if every sub-term of the form $\text{REC}x.t$ is guarded.

It will be convenient to have an inductive principle for guarded processes, that is closed terms which are guarded.

Definition 8. *Let $\Downarrow$ be the least predicate over processes which satisfies the following rules:*

(a) $\mathsf{0}\Downarrow, a.p\Downarrow$
(b) $p\Downarrow, q\Downarrow$ *implies* $\tau.p\Downarrow$ *and* $(p+q)\Downarrow$
(c) $t\{x\mapsto \text{REC}x.t\}\Downarrow$ *implies* $\text{REC}x.t\Downarrow$.

Lemma 3. *Suppose x is guarded in t for every $x \in \text{FV}(t)$. Then $t\rho\Downarrow$, for any substitution ρ such that $\text{DOM}(\rho) \subseteq \text{FV}(t)$.*

Proof. By structural induction on t. □

Proposition 7. *If p is guarded then $p \Downarrow$.*

Proof. By structural induction on p. The only non-trivial case is when it has the form $\text{REC}x.t$, where we know that x is guarded in t. By the previous lemma this means that $t\{x \mapsto \text{REC}x.t\} \Downarrow$, and therefore employing rule (c) from Definition 8 we can conclude that $\text{REC}x.t \Downarrow$. □

In the remainder of this appendix we will assume that all processes are guarded; this assumption is also used throughout the paper.

Proposition 8 (Proposition 1). *For every process p there exists a head normal form $\text{HNF}(p)$ such that $p =_{ineq} \text{HNF}(p)$.*

Proof. By induction on $p \Downarrow$. We proceed by an analysis of the structure of p.

- If p has the form $a.q$, or $\mathsf{0}$ then it is already a hnf.
- If p is $\text{REC}x.t$ then by induction on $\Downarrow$ we know that there is some hnf h such that $t\{x \mapsto \text{REC}x.t\} =_{ineq} h$. Using the (UFD) and (FLD) rules we obtain $\text{REC}x.t =_{ineq} h$.
- If p is $\tau.q$ again the result follows by induction, using the axiom $\tau.X = X$.
- Finally suppose p has the form $p_1 + p_2$. By induction $p_i =_{ineq} h_i$ for some hnfs h_1, h_2. Suppose these have the form $\sum_{a \in A_i} a.p_a^i$, for $i = 1, 2$. Then one can show that

$$p =_{ineq} \sum_{a \in (A_1 - A_2)} a.p_a^1 + \sum_{a \in (A_2 - A_1)} a.p_a^2 + \sum_{a \in (A_2 \cap A_1)} a.(p_a^1 + p_a^2)$$

  which is in hnf. □

Corollary 3 (Proposition 3). *If $\text{HNF}(p) \xrightarrow{a} q$ then $p \stackrel{\tau}{\twoheadrightarrow}^{\star} \stackrel{a}{\twoheadrightarrow} q$.*

Proof. The proof proceeds by induction on $p \Downarrow$ and a case analysis on the construction of $\text{HNF}(p)$ as outlined in the previous proposition. □

References

1. Giovanni Bernardi and Adrian Francalanza. Full-abstraction for must testing preorders (extended abstract), 2017. to appear.
2. Giovanni Bernardi and Matthew Hennessy. Mutually testing processes. *Logical Methods in Computer Science*, 11(2), 2015.
3. Michael Brandt and Fritz Henglein. Coinductive axiomatization of recursive type equality and subtyping. *Fundam. Inform.*, 33(4):309–338, 1998.
4. Mario Bravetti and Gianluigi Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundam. Inform.*, 89(4):451–478, 2008.
5. Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, October 1964.
6. Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984.

7. C. A. R. Hoare. Communicating sequential processes (reprint). *Commun. ACM*, 26(1):100–106, 1983.
8. Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994.
9. R. Milner. *A Calculus of Communicating Systems*. Springer, 1982.
10. Robin Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984.
11. Robin Milner. A complete axiomatisation for observational congruence of finite-state behaviors. *Inf. Comput.*, 81(2):227–247, 1989.
12. Alexander Moshe Rabinovich. A complete axiomatisation for trace congruence of finite state behaviors. In Stephen D. Brookes, Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 9th International Conference, New Orleans, LA, USA, April 7-10, 1993, Proceedings*, volume 802 of *Lecture Notes in Computer Science*, pages 530–543. Springer, 1993.
13. Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966.