Checking π-Calculus Structural Congruence is Graph Isomorphism Complete

Victor Khomenko[1] and Roland Meyer[2]

[1] School of Computing Science, Newcastle University
Newcastle upon Tyne, NE1 7RU, U.K.
e-mail: Victor.Khomenko@ncl.ac.uk
[2] Department of Computing Science, University of Oldenburg
D-26129 Oldenburg, Germany
e-mail: Roland.Meyer@informatik.uni-oldenburg.de

Abstract. We show that the problems of checking π-Calculus structural congruence (πSC) and graph isomorphism (GI) are Karp reducible to each other. The reduction from GI to πSC is given explicitly, and the reduction in the opposite direction proceeds by transforming πSC into an instance of the term equality problem (i.e., the problem of deciding equivalence of two terms in the presence of associative and/or commutative operations and commutative variable-binding quantifiers), which is known to be Karp reducible to GI. Our result is robust in the sense that it holds for several variants of structural congruence and some rather restrictive fragments of π-Calculus.
Furthermore, we address the question of solving πSC in practice, and describe a number of optimisations exploiting specific features of π-Calculus terms, which allow one to significantly reduce the size of the resulting graphs that have to be checked for isomorphism.
Keywords: Structural Congruence, Graph Isomorphism, π-Calculus, Computational Complexity.

1 Introduction

π-Calculus is a well-known formalism in the area of reconfigurable and mobile systems. Its main advantage is that it allows for concise modelling of systems where the interconnect topology between system components changes dynamically. However, even the conventional concurrent systems are notoriously difficult to design correctly because of the complexity of their behaviour, and reconfigurable systems add another layer of complexity due to their dynamical nature. Hence, formal methods, especially computer-aided simulation and verification tools, have to be employed in the design process to ensure correct behaviour.

The Mobility Workbench [VM94], the verification kit Hal [FGMP03], the Spatial Logic Model Checker [Cai04] and Petruchio [SM08] are examples of well-known π-Calculus verification tools. A basic task they have to perform while computing the state space (or the Petri net representation in case of Petruchio) is to check *structural congruence* — an equivalence relation between π-Calculus terms. So to make verification tools efficient, practical algorithms for checking structural congruence are needed. This problem will be denoted by πSC in sequel.[3]

In this paper we show that πSC is equivalent to checking graph isomorphism (GI). More precisely, we provide two polynomial-time reductions, one reducing GI to πSC, and the other

[3] The variant of π-Calculus we consider in this paper does not have the replication operator (also known as bang operator) and uses parameterised recursion instead, and the structural congruence does not expand recursive definitions. We just remark that for π-Calculus with the replication operator checking structural congruence is much more difficult, and for some variants of structural congruence even the decidability is not known. (However, see [EG99, EG04b, EG04a, EG07] for some positive results.)

reducing πSC to the term equality problem (TE), i.e., the problem of deciding if two terms containing otherwise uninterpreted associative, commutative and associative-commutative operations and commutative variable-binding quantifiers are equal. Furthermore, the terms may contain constants, uninterpreted functional symbols and non-commutative variable-binding quantifiers. TE is known to be Karp reducible to GI [Bas94].

GI is trivially in $\mathcal{NP}$; however, it is one of the small number of problems which are neither shown to be $\mathcal{NP}$-complete, nor have a known polynomial-time algorithm.[4] GI is not believed to be $\mathcal{NP}$-complete, as in such a case Stockmeyer's polynomial hierarchy [Sto76] would collapse (which would be a big surprise in the theory of computational complexity). On the other hand, it is plausible that GI is in $\mathcal{P}$, and in such a case our reduction would yield a polynomial-time algorithm for πSC. (See [KST93] for more information about GI.)

Since the exact complexity of GI is not known, the complexity theorists introduced a new complexity class $\mathcal{GI}$, defined as the class of problems Cook reducible (i.e., polynomial-time Turing reducible) to GI. Hence, our main result amounts to saying that πSC is $\mathcal{GI}$-complete. Furthermore, this paper actually yields Karp reductions (i.e., polynomial-time many-one reductions) from πSC to GI and from GI to πSC, which are a well-behaving subclass of Cook reductions.[5]

In spite of the fact that a polynomial-time algorithm for GI has not been invented yet, in practice GI can be solved very efficiently by publicly available solvers [NAU, McK]. With the presented translation, one can use off-the-shelf GI solvers as back-ends in π-Calculus tools to check structural congruence. The approach of reducing πSC to GI and using an established GI solver has a number of advantages compared with implementing a custom solver for πSC:

- Established solvers tend to be very efficient due to the use of advanced heuristics; they are also well debugged.
- All improvements to solvers are inherited in the tools that use them — i.e., such improvements come essentially for free.
- The solvers are used in the black-box fashion, i.e., no knowledge about their implementation is required; this effectively separates the concerns.
- Implementing a reduction to some standard problem like GI or SAT is usually much less effort consuming and error prone than implementing a custom solver for the problem at hand.

For these reasons the development time is reduced and the quality of the resulting tools is improved.

Our reduction of πSC to TE (and ultimately to GI) is intended not only as a theoretical result, but also to be applied for checking π-Calculus structural congruence in our tool Petruchio [SM08]. Hence we show how the reduction of TE to GI described in [Bas94] can be optimised by exploiting specific features of π-Calculus terms. These optimisations not only reduce the size of the graphs that are to be checked for isomorphism, but also improve their structure so that this check becomes easier in many practical cases.

The paper is organised as follows. In Section 2 we introduce π-Calculus and structural congruence, as well as the term equality (TE) problem. In Section 3 we show that πSC and GI are polynomial-time reducible to each other, and then extend this result to some interesting variants of πSC in Section 4. Section 5 describes a number of optimisations which allow for a significant reduction in the sizes of the resulting graphs that have to be checked for isomorphism, and we conclude in Section 6.

[4] Examples of other such problems are Integer Factorisation and Discrete Logarithm.

[5] To show that a problem is in $\mathcal{NP}$ it is enough to give a Karp reduction to some problem in $\mathcal{NP}$, which is generally not the case for Cook reductions: e.g., any problem in *co*-$\mathcal{NP}$ has a Cook reduction to any $\mathcal{NP}$-complete problem, whereas a problem in *co*-$\mathcal{NP}\setminus\mathcal{NP}$ (provided such problems exist) will not have a Karp reduction to any problem in $\mathcal{NP}$. That is, $\mathcal{NP}$ is closed w.r.t. Karp reductions, but is not closed w.r.t. Cook reductions unless $\mathcal{NP} =$*co*-$\mathcal{NP}$.

2 Basic Notions

In this section we recall the basic notions concerning π-Calculus and the term equality (TE) problem.

2.1 The π-Calculus

We use a variant of π-Calculus with parameterised recursion similar to the one proposed in [SW01]. Let the set $\mathcal{N} \stackrel{\text{df}}{=} \{a, b, x, y \ldots\}$ of *names* contain the channels (which are also the possible messages) that occur in communications. During a process execution the *prefixes* π are successively consumed (removed) from the process to communicate with other processes or to perform silent actions:

$$\pi ::= \overline{a}\langle b\rangle \;\mid\; a(x) \;\mid\; \tau.$$

The *output action* $\overline{a}\langle b\rangle$ sends the name b along channel a. The *input action* $a(x)$ receives a name on a that replaces x. The τ prefix stands for a *silent action*.

To denote recursive processes, we use *process identifiers* from the set $PIDS \stackrel{\text{df}}{=} \{K, L, \ldots\}$. A process identifier is defined by an equation $K(\tilde{x}) := P$, where $\tilde{x}$ is a short-hand notation for $x_1, \ldots, x_k$. When the identifier is *called*, $K\lfloor\tilde{a}\rfloor$, it is replaced by the process obtained from P by replacing the names $\tilde{x}$ by $\tilde{a}$. More precisely, a *substitution* $\sigma = \{\tilde{a}/\tilde{x}\}$ is a function that maps the names in $\tilde{x}$ to $\tilde{a}$, and is the identity for all the names not in $\tilde{x}$. The *application of substitution*, $P\sigma$, is defined in the standard way [SW01].

A π-Calculus process is either a *stop* process, a *call* to a process identifier, a *choice* between processes, a *parallel composition* of processes, a prefix followed by a process, or a *restriction* of a name in a process:[6]

$$P ::= \mathbf{0} \;\mid\; K\lfloor\tilde{a}\rfloor \;\mid\; P + P \;\mid\; P \,|\, P \;\mid\; \pi.P \;\mid\; \nu a{:}P.$$

We use the notation $\sum_{i=1}^{n} P_i$ and $\prod_{i=1}^{n} P_i$ for iterated $+$ and $|$, and denote by $\mathcal{PROC}$ the set of all π-Calculus processes. We also abbreviate $\nu a_1{:}\ldots\nu a_k{:}P$ by $\nu\tilde{a}{:}P$, and define $\nu\tilde{a}{:}P \stackrel{\text{df}}{=} P$ if $\tilde{a}$ is empty.

Some definitions of π-Calculus syntax require the choices to be *guarded,* i.e., each choice should have the form $\sum_{i\in I} \pi_i.P_i$. However, we decided to use the above syntax, as it is more permissive, and the requirement of guarded sums does not affect our results (i.e., πSC and GI are Karp reducible to each other even if the choices are guarded, and the reductions are the same as for the general case). Furthermore, the above syntax allows one to consider the interesting extensions of the π-Calculus structural congruence relation discussed in Section 4, where a process with guarded choices can be structurally congruent with one having un-guarded choices.

The input action $a(b)$ and the restriction $\nu c{:}P$ *bind* the names b and c, respectively. The *set of bound names* in a process P is $bn(P)$. A name which is not bound is *free,* and the *set of free names* in P is $fn(P)$. We permit α-conversion of bound names. Therefore, w.l.o.g., we make the following assumptions common in π-Calculus theory, collectively referred to as NoClash in sequel: for any process P,

- a name is bound at most once in P;
- $bn(P) \cap fn(P) = \emptyset$;
- if a substitution $\sigma = \{\tilde{a}/\tilde{x}\}$ is applied to P then $bn(P) \cap (\tilde{a} \cup \tilde{x}) = \emptyset$.

[6] We use the notation $\nu a{:}P$ instead of the standard notation $\nu a.P$ in order to avoid confusion with the prefixing operator '.' as in, e.g., $\tau.\mathbf{0}$.

Definition 1 (Structural congruence $\equiv$). *The* structural congruence *relation $\equiv$ is the smallest congruence on process terms defined by the following axioms:*

α-conversion of bound names is permitted	(α)
$+$ and $\mid$ are associative and commutative	$(AC^+), (AC^\mid)$
$\mathbf{0}$ *is a neutral element for $+$ and $\mid$*	$(\mathbf{0}^+), (\mathbf{0}^\mid)$
$\nu x{:}P \equiv P \quad$ *if* $x \notin fn(P)$	(P^ν)
$\nu x{:}\nu y{:}P \equiv \nu y{:}\nu x{:}P$	(C^ν)
$\nu x{:}(P \mid Q) \equiv P \mid (\nu x{:}Q) \quad$ *if* $x \notin fn(P)$	$(SE^\mid)$

The last axiom is called scope extrusion. $\Diamond$

Often the axiom $\nu x{:}\mathbf{0} \equiv \mathbf{0}$ is used instead of (P^ν). One can see that this axiom is equivalent to (P^ν), as it can be obtained from it by taking $P = \mathbf{0}$, and (P^ν) can be derived from this axiom as follows:

$$\nu x{:}P \equiv (\nu x{:}P) \mid \mathbf{0} \equiv \nu x{:}(P \mid \mathbf{0}) \equiv P \mid \nu x{:}\mathbf{0} \equiv P \mid \mathbf{0} \equiv P.$$

The variants of structural congruence extending it by scope extrusion axioms for choice and/or prefixing are considered in Section 4. We do not provide the SOS rules defining π-Calculus semantics, as they are not essential for this paper; they can be found in [SW01].

2.2 The term equality problem

Let t, t' be terms built using:

- quantifiers introducing bound names; some of these quantifiers may be commutative, i.e., the terms $\theta x{:}\theta y{:}t''$ and $\theta y{:}\theta x{:}t''$ (where θ is a commutative quantifier) are considered equivalent;
- associative, commutative and associative-commutative binary operators;
- uninterpreted functional symbols and constants (the latter can be considered as uninterpreted functional symbols of arity 0);
- the names bound by the quantifiers.

The term equality problem (TE) consists in deciding if t and t' are equivalent modulo associativity, commutativity and associativity-commutativity axioms for the corresponding operators, the commutativity of corresponding quantifiers, and α-conversion of bound names. This problem is $\mathcal{GI}$-complete [Bas94]; in Section 5.1 we recall the TE to GI reduction.

Moreover, a special case of this problem, with no commutative quantifiers, is in $\mathcal{P}$ [Bas94]. Indeed, in such a case the terms can be converted into canonical form using the function can described below, and then compared for equality. Intuitively, adjacent applications of the same associative operator are compounded into a single application of the operator of greater arity, and the operands of commutative operators are sorted w.r.t. some total order $\prec$ on terms. The following auxiliary objects are used in the definition of can:

- $VAR = \{x_1, x_2, \ldots\}$ is a set of variables that can occur only in terms in canonical form (they will be used to replace the bound names in terms); we define a total order on VAR as $x_i < x_j$ iff $i < j$;
- $\prec$ is any total order on terms in canonical form that can be computed in polynomial time;
- $sort_\prec$ is a function that sorts a list of terms in canonical form according to $\prec$;

– $\mathit{frontier}_f$, where f is a functional symbol or operation, is a function that takes a term and returns a list of terms:

$$\mathit{frontier}_f(t) \stackrel{\text{df}}{=} \begin{cases} \mathit{frontier}_f(a_1) \circ \cdots \circ \mathit{frontier}_f(a_n) & \text{if } t = f(a_1, \ldots, a_n) \\ [t] & \text{otherwise,} \end{cases}$$

where $\circ$ denotes the list concatenation. Intuitively, if $\otimes$ is a binary associative operator then this function gathers all the operands needed for compounding $\otimes$ into an operator of greater arity, e.g., $\mathit{frontier}_{\otimes}((a \otimes b) \otimes (g(c) \otimes d)) = [a, b, g(c), d]$, i.e., the version of $\otimes$ with the arity 4 is applied to the operands $a, b, g(c), d$ in this order.

Let t be a term. Then $\mathsf{can}(t)$ is defined as:

Case 1: If t is a constant c then $\mathsf{can}(t) \stackrel{\text{df}}{=} c$.
Case 2: If t is a variable v then $\mathsf{can}(t) \stackrel{\text{df}}{=} v$.
Case 3: If $t = f(t_1, \ldots, t_n)$ then:

$$\mathsf{can}(t) \stackrel{\text{df}}{=} \begin{cases} f(\mathsf{can}(t_1), \ldots, \mathsf{can}(t_n)) & \text{if } f \text{ is uninterpreted} \\ (f, \mathit{sort}_{\prec}([\mathsf{can}(t_1), \mathsf{can}(t_2)])) & \text{if } f \text{ is comm. but not assoc.} \\ (f, [\mathsf{can}(t'_1), \ldots, \mathsf{can}(t'_k)]) & \text{if } f \text{ is assoc. but not comm.} \\ (f, \mathit{sort}_{\prec}([\mathsf{can}(t'_1), \ldots, \mathsf{can}(t'_k)])) & \text{if } f \text{ is assoc. and comm.,} \end{cases}$$

where $[t'_1, \ldots, t'_k] = \mathit{frontier}(t)$ in the last two cases.

Case 4: If $t = \theta v{:}t'$, where θ is a quantifier, then $\mathsf{can}(t) \stackrel{\text{df}}{=} \theta x{:}(\mathsf{can}(t')\{x/v\})$, where $x \in \mathit{VAR}$ is the smallest variable not occurring in $\mathsf{can}(t')$. Note that the quantifier remains in $\mathsf{can}(t)$ in order to keep the information about its type and position.

Observe that can does not yield a canonical form if some of the quantifiers commute.

3 Reductions

In this section we show that πSC and GI are Karp reducible to each other.

3.1 Reducing GI to πSC

First we show that πSC is $\mathcal{GI}$-hard. More precisely, we describe a polynomial-time computable transformation T which takes a graph and builds a π-Calculus term, such that $T(G) \equiv T(G')$ iff G and G' are isomorphic. This gives a Karp reduction of GI to πSC.

Each graph $G = (V, E)$ can be converted into a digraph $\vec{G} = (V, \vec{E})$ by replacing each edge $\{v, w\}$ by a pair of arcs (v, w) and (w, v). Trivially, two graphs G and G' are isomorphic iff the corresponding digraphs $\vec{G}$ and $\vec{G'}$ are isomorphic. Hence, it remains to show how to reduce the digraph isomorphism problem to πSC.

Given a digraph $\vec{G} = (V, \vec{E})$, we interpret each vertex $v \in V$ as a restricted name, and also represent it by the π-Calculus sub-term $K\lfloor v \rfloor$; moreover, each arc $(v, w) \in \vec{E}$ corresponds to a π-Calculus sub-term $L\lfloor v, w \rfloor$. Here K and L are process identifiers, and their defining equations can be arbitrary, as the π-Calculus structural congruence does not expand calls to process identifiers. Hence, we get the following representation of a digraph as a π-Calculus process:

$$\nu v_1{:}\ldots\nu v_n{:}\left(\prod_{v \in V} K\lfloor v \rfloor \mid \prod_{(v,w) \in \vec{E}} L\lfloor v, w \rfloor\right) .$$

One can see that for this construction the property "$T(\vec{G}) \equiv T(\vec{G'})$ iff $\vec{G}$ and $\vec{G'}$ are isomorphic" holds. Indeed, suppose φ is an isomorphism from $\vec{G}$ to $\vec{G'}$. Then $T(\vec{G})$ contains

sub-terms $K\lfloor v \rfloor$ and $L\lfloor v, w \rfloor$ iff $T(\vec{G'})$ contains sub-terms $K\lfloor \varphi(v) \rfloor$ and $L\lfloor \varphi(v), \varphi(w) \rfloor$. Hence, $T(\vec{G})$ can be transformed into $T(\vec{G'})$ by α-conversion of bound names (with the substitution given by φ) and by using commutativity and associativity of $|$, i.e., $T(\vec{G}) \equiv T(\vec{G'})$. Conversely, if $T(\vec{G}) \equiv T(\vec{G'})$ then $T(\vec{G})$ could be transformed by α-conversion into some term t that can in turn be transformed into $T(\vec{G'})$ by using only associativity and commutativity of $|$. The substitution given by this α-conversion is an isomorphism between $\vec{G}$ and $\vec{G'}$.

Note that a rather restricted fragment of π-Calculus is sufficient to make πSC $\mathcal{GI}$-hard. Indeed, the above construction places all the restrictions in the beginning of the term, and uses neither the choice operator $+$, nor the prefixing operator '.', nor input/output/silent actions, nor public channels. Furthermore, $|$ can be replaced by $+$, and a call to a process identifier $K\lfloor v, w \rfloor$ can be replaced by, e.g., $\overline{v}\langle w \rangle.\mathbf{0}$. To summarise, if the π-Calculus fragment has the restriction operator ν, at least one of $|$ or $+$, and some means of referring to the names bound by ν (these could be input or output prefixes or calls to process identifiers) then the πSC problem for this fragment is $\mathcal{GI}$-hard.

On the other hand, πSC for a π-Calculus fragment not including a restriction operator ν is in $\mathcal{P}$ (see the end of Section 3.2). Moreover, πSC for a π-Calculus fragment including neither $|$ nor $+$, i.e., a term is a sequence of prefixes interspersed with restrictions, finishing with either $\mathbf{0}$ or a call to a process identifier $K\lfloor \tilde{a} \rfloor$, can also be easily seen to be in $\mathcal{P}$. For such terms a canonical form can be obtained by first removing the restrictions whose bound names do not occur in their scopes, and then using a canonical naming scheme for the remaining bound variables based on their last occurrence in the term.

3.2 Reducing πSC to GI

We observe that it is enough to reduce πSC to TE, as the latter is Karp reducible to GI [Bas94] (this reduction is recalled in Section 5.1). Note that in spite of an apparent similarity between πSC and TE, πSC is not directly an instance of TE, as input prefixes are different from quantifiers in TE, the individual prefixes do not directly correspond to constants or variables, and in the definition of π-Calculus structural congruence there are axioms besides α-conversion of bound names, associativity, commutativity, associativity-commutativity and quantifier commutativity, viz. $(\mathbf{0}^+)$, $(\mathbf{0}^|)$, (P^ν) and $(SE^|)$. Hence, some work has to be done to cast πSC in terms of TE.

We assume that NoClash holds (this can be ensured in polynomial time). The first step is to eliminate the need in using the axioms that are outside the scope of TE for showing πSC. To achieve this, we convert the π-Calculus terms to the following normal form:

- use the axioms $(\mathbf{0}^+)$, $(\mathbf{0}^|)$ and (P^ν) to simplify the resulting terms until none of them applies;
- maximise the scope of restrictions using the $(SE^|)$ axiom (in the reverse direction).

Clearly, this conversion can be done in polynomial time. At this point, the structural congruence of the resulting terms can be proven without using the axioms $(SE^|)$, $(\mathbf{0}^+)$, $(\mathbf{0}^|)$ and (P^ν); the formal proof of this fact is rather technical, and is deferred until Section 3.3.

Now we replace the sub-terms of the form $\overline{x}\langle y \rangle$ by $s(x, y)$, and the sub-terms of the form $x(y).P$ by $\rho y{:}r(x, y).P$, where ρ is a new non-commutative quantifier (in the sense of TE), and $s(\_, \_)$ and $r(\_, \_)$ are new uninterpreted functional symbols. The resulting terms form an instance of TE, where:

- the choice and parallel composition are associative-commutative operators;
- $s(\_, \_)$, $r(\_, \_)$, the prefixing operator '.' and the process identifiers are uninterpreted functional symbols;
- ν is a commutative quantifier and ρ is a non-commutative quantifier;
- public channel names, τ and $\mathbf{0}$ are constants (since all the axioms for $\mathbf{0}$ no longer apply, it can be regarded as uninterpreted);

– the names introduced by the restriction and input prefixes are the names bound by the quantifiers ν and ρ.

Clearly, the resulting terms are equivalent iff the original π-Calculus terms were structurally congruent, which completes our reduction of πSC to TE. For example,

$$\begin{aligned}
\nu x{:}\overline{a}\langle x\rangle.b(z).\overline{z}\langle x\rangle.\mathbf{0} \mid \nu y{:}a(p).\overline{b}\langle y\rangle.\mathbf{0} \mid \nu q{:}\tau.\mathbf{0} \mid \nu t{:}\mathbf{0} &\equiv (P^{\nu}), (\mathbf{0}^{|}) \\
\nu x{:}\overline{a}\langle x\rangle.b(z).\overline{z}\langle x\rangle.\mathbf{0} \mid \nu y{:}a(p).\overline{b}\langle y\rangle.\mathbf{0} \mid \tau.\mathbf{0} &\equiv (SE^{|}) \\
\nu x{:}\nu y{:}(\overline{a}\langle x\rangle.b(z).\overline{z}\langle x\rangle.\mathbf{0} \mid a(p).\overline{b}\langle y\rangle.\mathbf{0} \mid \tau.\mathbf{0}) &\rightsquigarrow \text{(replace the i/o prefixes)} \\
\nu x{:}\nu y{:}(s(a,x).\rho z{:}r(b,z).s(z,x).\mathbf{0} \mid \rho p{:}r(a,p).s(b,y).\mathbf{0} \mid \tau.\mathbf{0}).
\end{aligned}$$

One can observe that if the original π-Calculus terms did not contain the restriction operator then the resulting instance of TE does not contain commutative quantifiers. This special case of TE can be solved in polynomial time by conversion to the canonical form, as described in Section 2.2. Hence, the special case of πSC with terms not containing restrictions is in $\mathcal{P}$.

3.3 The proof

We now present the proof that was deferred in Section 3.2. More precisely, we prove that after the processes have been converted to the normal form explained in Section 3.2, the axioms $(SE^{|})$, $(\mathbf{0}^{+})$, $(\mathbf{0}^{|})$ and (P^{ν}) are no longer needed to prove their structural congruence.

We first formally define this normal form as well as a function nf converting any π-Calculus process into it. Then we define an equivalence $\widehat{\equiv}$ on the processes in normal form that corresponds to $\equiv$ restricted to such processes and not using the above axioms. It turns out that $\widehat{\equiv}$ characterises $\equiv$, i.e., for any π-Calculus processes P and Q, $P \equiv Q$ iff $\mathsf{nf}(P) \mathrel{\widehat{\equiv}} \mathsf{nf}(Q)$. That is, the above axioms are not necessary when proving the structural congruence of the processes in normal form, which completes the flow of the argument.

Definition 2 (Normal form). *A* non-restricted process in normal form $P^{\neg\nu}$ *and a* process in normal form P^{nf} *are defined as follows:*

$$\begin{aligned}
P^{\neg\nu} &::= \pi.\mathbf{0} \;\mid\; K\lfloor\tilde{a}\rfloor \;\mid\; \pi.P^{nf} \;\mid\; P^{nf} + P^{nf} \;\mid\; P^{\neg\nu} \mid P^{\neg\nu} \\
P^{nf} &::= \nu\tilde{a}{:}P^{\neg\nu}\,,
\end{aligned}$$

where each name in $\tilde{a}$ occurs free in the scope of the restriction. We denote by $\mathcal{PROC}^{nf}$ the set of all processes in normal form. ◊

In this definition $\tilde{a}$ can be empty, i.e., $P^{\neg\nu}$ is a special case of P^{nf}: in fact, $P^{\neg\nu}$ is a process in normal form whose top-level operator is not a restriction. Observe that this normal form requires that the operands of $\mid$ do not have the restriction as the top-level operation, and that neither P^{nf} nor $P^{\neg\nu}$ can be $\mathbf{0}$.

The following function translates every process into either $\mathbf{0}$ or a process in normal form.

Definition 3. *The function* $\mathsf{nf} : \mathcal{PROC} \to \mathcal{PROC}^{nf} \cup \{\mathbf{0}\}$ *is defined inductively as shown in Figure 1.* ◊

In the last line of this definition $\tilde{a}_P$ and $\tilde{a}_Q$ can be empty, and the NoClash assumption is essential. Note that this normal form is not invariant under structural congruence, i.e., it is possible that $P \equiv Q$ but $\mathsf{nf}(P) \neq \mathsf{nf}(Q)$.

The following lemma shows that the process $\mathsf{nf}(P)$ is either $\mathbf{0}$ or in normal form (i.e., the range of the function nf is correct); moreover P and $\mathsf{nf}(P)$ are structurally congruent and compatible with α-conversion, as required to for the characterisation of structural congruence in Lemma 2.

$$\mathsf{nf}(\mathbf{0}) \stackrel{\text{df}}{=} \mathbf{0} \qquad \mathsf{nf}(K\lfloor\tilde{a}\rfloor) \stackrel{\text{df}}{=} K\lfloor\tilde{a}\rfloor \qquad \mathsf{nf}(\pi.P) \stackrel{\text{df}}{=} \pi.\mathsf{nf}(P)$$

$$\mathsf{nf}(P+Q) \stackrel{\text{df}}{=} \begin{cases} \mathsf{nf}(P)+\mathsf{nf}(Q) & \text{if } \mathsf{nf}(P)\neq\mathbf{0}\neq\mathsf{nf}(Q) \\ \mathsf{nf}(P) & \text{if } \mathsf{nf}(P)\neq\mathbf{0}=\mathsf{nf}(Q) \\ \mathsf{nf}(Q) & \text{if } \mathsf{nf}(P)=\mathbf{0}\neq\mathsf{nf}(Q) \\ \mathbf{0} & \text{if } \mathsf{nf}(P)=\mathbf{0}=\mathsf{nf}(Q). \end{cases}$$

$$\mathsf{nf}(\nu x{:}P) \stackrel{\text{df}}{=} \begin{cases} \mathsf{nf}(P) & \text{if } x\notin fn\,(P) \\ \nu x{:}\mathsf{nf}(P) & \text{if } x\in fn\,(P) \end{cases}$$

$$\mathsf{nf}(P\,|\,Q) \stackrel{\text{df}}{=} \begin{cases} \mathbf{0} & \text{if } \mathsf{nf}(P)=\mathbf{0}=\mathsf{nf}(Q) \\ \mathsf{nf}(P) & \text{if } \mathsf{nf}(P)\neq\mathbf{0}=\mathsf{nf}(Q) \\ \mathsf{nf}(Q) & \text{if } \mathsf{nf}(P)=\mathbf{0}\neq\mathsf{nf}(Q) \\ \nu\tilde{a}_P{:}\nu\tilde{a}_Q{:}(P^{\neg\nu}\,|\,Q^{\neg\nu}) & \text{if } \mathsf{nf}(P)=\nu\tilde{a}_P{:}P^{\neg\nu}\wedge\mathsf{nf}(Q)=\nu\tilde{a}_Q{:}Q^{\neg\nu}. \end{cases}$$

Fig. 1. Definition of nf.

Lemma 1 (Properties of nf). *For every process $P\in\mathcal{PROC}$, the following hold:*

(a) $\mathsf{nf}(P)\in\mathcal{PROC}^{nf}\cup\{\mathbf{0}\}$;
(b) $P\equiv\mathsf{nf}(P)$;
(c) for any substitution σ, $\mathsf{nf}(P)\sigma=\mathsf{nf}(P\sigma)$.

Proof. Induction on the structure of processes. □

As the function nf is not invariant under structural congruence, we introduce another equivalence relation $\widehat{\equiv}$ on processes in normal form, which characterises $\equiv$. This relation is similar to $\equiv$ but omits the axioms $(SE^{|})$, $(\mathbf{0}^{+})$, $(\mathbf{0}^{|})$ and (P^{ν}).

Definition 4 (Normal form equivalence $\widehat{\equiv}$). *The* normal form equivalence $\widehat{\equiv}$ *is the least equivalence on processes in normal form, i.e.,* $\widehat{\equiv}\subseteq\mathcal{PROC}^{nf}\times\mathcal{PROC}^{nf}\cup\{(\mathbf{0},\mathbf{0})\}$, *such that α-conversion of bound names is allowed, $+$ and $|$ are associative and commutative, the axiom $\nu a{:}\nu b{:}P^{nf}\mathrel{\widehat{\equiv}}\nu b{:}\nu a{:}P^{nf}$ holds, and the following implications are valid:*

$$\begin{aligned} P^{nf}\mathrel{\widehat{\equiv}}Q^{nf} &\Rightarrow \pi.P^{nf}\mathrel{\widehat{\equiv}}\pi.Q^{nf} \\ P^{nf}\mathrel{\widehat{\equiv}}Q^{nf} &\Rightarrow P^{nf}+R^{nf}\mathrel{\widehat{\equiv}}Q^{nf}+R^{nf} \\ P^{nf}\mathrel{\widehat{\equiv}}Q^{nf}\wedge a\in fn\left(P^{nf}\right)\cap fn\left(Q^{nf}\right) &\Rightarrow \nu a{:}P^{nf}\mathrel{\widehat{\equiv}}\nu a{:}Q^{nf} \\ P^{\neg\nu}\mathrel{\widehat{\equiv}}Q^{\neg\nu} &\Rightarrow P^{\neg\nu}\,|\,R^{\neg\nu}\mathrel{\widehat{\equiv}}Q^{\neg\nu}\,|\,R^{\neg\nu}. \end{aligned}$$

◊

The side condition $a\in fn\left(P^{nf}\right)\cap fn\left(Q^{nf}\right)$ ensures that $\nu a{:}P^{nf}$ and $\nu a{:}Q^{nf}$ are processes in normal form. Note that $P^{nf}\equiv Q^{nf}$ implies $fn\left(P^{nf}\right)=fn\left(Q^{nf}\right)$, so to check $a\in fn\left(P^{nf}\right)\cap fn\left(Q^{nf}\right)$ it is sufficient to check whether $a\in fn\left(P^{nf}\right)$ holds.

Intuitively, $\widehat{\equiv}$ is a congruence w.r.t. prefixing, choice and restriction, but the congruence property for parallel compositions holds only in a limited way, as, in general, $P^{nf}\,|\,Q^{nf}$ is not in normal form even if both P^{nf} and Q^{nf} are; hence, only non-restricted processes in normal form can be replaced by equivalent ones if they are under the parallel composition.

Lemma 2 ($\widehat{\equiv}$ characterises $\equiv$). $P\equiv Q$ *iff* $\mathsf{nf}(P)\mathrel{\widehat{\equiv}}\mathsf{nf}(Q)$.

Proof.
($\Longleftarrow$) We observe that all axioms making up $\widehat{\equiv}$ also hold for $\equiv$, i.e., $\mathsf{nf}(P)\mathrel{\widehat{\equiv}}\mathsf{nf}(Q)$ implies $\mathsf{nf}(P)\equiv\mathsf{nf}(Q)$. By Lemma 1(b), $P\equiv\mathsf{nf}(P)$ and $\mathsf{nf}(Q)\equiv Q$, and thus $P\equiv Q$ is obtained by transitivity of structural congruence.

($\Longrightarrow$) We assume $P \equiv Q$, and need to show that $\mathsf{nf}(P) \mathrel{\widehat{\equiv}} \mathsf{nf}(Q)$. We do this by an induction on the set $\equiv \subseteq \mathcal{PROC} \times \mathcal{PROC}$, i.e., on the set of pairs of processes that are structurally congruent. In the base case, we demonstrate that for each axiom $P \equiv Q$ defining $\equiv$ (see Definition 1), $\mathsf{nf}(P) \mathrel{\widehat{\equiv}} \mathsf{nf}(Q)$ holds. In the induction step, we assume that the property holds for congruent terms P and Q, i.e., $P \equiv Q$ implies $\mathsf{nf}(P) \mathrel{\widehat{\equiv}} \mathsf{nf}(Q)$. We then consider the terms $P' = \pi.P$ ($\nu a.P$, $P + R$, $P \mid R$) and $Q' = \pi.Q$ ($\nu a.Q$, $Q + R$, $Q \mid R$), where Q' is obtained from P' by replacing P by Q (and thus $P' \equiv Q'$), and show that $\mathsf{nf}(P') \mathrel{\widehat{\equiv}} \mathsf{nf}(Q')$.

Base case

We consider the axioms defining $\equiv$ (see Definition 1) in turn.

(α) We consider α-conversion of restricted names (the case of names bound by input prefixes can be dealt with analogously), i.e., we have $\nu a{:}P \equiv \nu b{:}(P\{b/a\})$.

- If $a \notin \mathit{fn}(P)$ then $P = P\{b/a\}$, and since $b \notin \mathit{fn}(P)$ due to the NoClash assumption, $b \notin \mathit{fn}(P\{b/a\})$. So, using the definition of nf, we get

$$\mathsf{nf}(\nu a{:}P) = \mathsf{nf}(P) = \mathsf{nf}(P\{b/a\}) = \mathsf{nf}(\nu b{:}P\{b/a\}).$$

- If $a \in \mathit{fn}(P)$ then

$$\begin{aligned}
\mathsf{nf}(\nu a{:}P) &= \text{(Def. of } \mathsf{nf}\text{)} \\
\nu a{:}\mathsf{nf}(P) &\mathrel{\widehat{\equiv}} (\alpha\text{-conversion in } \mathrel{\widehat{\equiv}}) \\
\nu b{:}(\mathsf{nf}(P)\{b/a\}) &= \text{(Lemma 1(c))} \\
\nu b{:}(\mathsf{nf}(P\{b/a\})) &= \text{(Def. of } \mathsf{nf}\text{)} \\
\mathsf{nf}(\nu b{:}(P\{b/a\})).
\end{aligned}$$

(AC^{+}) We first show that $\mathsf{nf}(P + Q) \mathrel{\widehat{\equiv}} \mathsf{nf}(Q + P)$. The case when one of the operands of $+$ is $\mathbf{0}$ is trivial (in fact, even $=$ holds in this case):

$$\mathsf{nf}(P + \mathbf{0}) = \mathsf{nf}(P) = \mathsf{nf}(\mathbf{0} + P).$$

We now assume that both operands differ from $\mathbf{0}$:

$$\begin{aligned}
\mathsf{nf}(P + Q) &= \text{(Def. of } \mathsf{nf}\text{)} \\
\mathsf{nf}(P) + \mathsf{nf}(Q) &\mathrel{\widehat{\equiv}} (\text{Comm. of } + \text{ w.r.t. } \mathrel{\widehat{\equiv}}) \\
\mathsf{nf}(Q) + \mathsf{nf}(P) &= \text{(Def. of } \mathsf{nf}\text{)} \\
\mathsf{nf}(Q + P).
\end{aligned}$$

Now we show that $\mathsf{nf}(P + (Q + R)) \mathrel{\widehat{\equiv}} \mathsf{nf}((P + Q) + R)$. We assume that none of P, Q, R is $\mathbf{0}$ (the other cases are trivial).

$$\begin{aligned}
\mathsf{nf}(P + (Q + R)) &= \text{(Def. of } \mathsf{nf}\text{)} \\
\mathsf{nf}(P) + (\mathsf{nf}(Q) + \mathsf{nf}(R)) &\mathrel{\widehat{\equiv}} (\text{Assoc. of } + \text{ w.r.t. } \mathrel{\widehat{\equiv}}) \\
(\mathsf{nf}(P) + \mathsf{nf}(Q)) + \mathsf{nf}(R) &= \text{(Def. of } \mathsf{nf}\text{)} \\
\mathsf{nf}((P + Q) + R).
\end{aligned}$$

($AC^{|}$) We show that $\mathsf{nf}(P \mid Q) \mathrel{\widehat{\equiv}} \mathsf{nf}(Q \mid P)$ and $\mathsf{nf}(P \mid (Q \mid R)) \mathrel{\widehat{\equiv}} \mathsf{nf}((P \mid Q) \mid R)$. We assume that none of P, Q, R is $\mathbf{0}$ (the other cases are trivial). Let $\mathsf{nf}(P) = \nu\tilde{a}_P{:}P^{\neg\nu}$, $\mathsf{nf}(Q) = \nu\tilde{a}_Q{:}Q^{\neg\nu}$ and $\mathsf{nf}(R) = \nu\tilde{a}_R{:}R^{\neg\nu}$. Then we get:

$$\begin{aligned}
\mathsf{nf}(P \mid Q) &= \text{(Def. of } \mathsf{nf}\text{)} \\
\nu\tilde{a}_P{:}\nu\tilde{a}_Q{:}(P^{\neg\nu} \mid Q^{\neg\nu}) &\mathrel{\widehat{\equiv}} (\nu x{:}\nu y{:}P^{nf} \mathrel{\widehat{\equiv}} \nu y{:}\nu x{:}P^{nf}) \\
\nu\tilde{a}_Q{:}\nu\tilde{a}_P{:}(P^{\neg\nu} \mid Q^{\neg\nu}) &\mathrel{\widehat{\equiv}} (\text{Comm. of } \mid \text{ w.r.t. } \mathrel{\widehat{\equiv}}) \\
\nu\tilde{a}_Q{:}\nu\tilde{a}_P{:}(Q^{\neg\nu} \mid P^{\neg\nu}) &= \text{(Def. of } \mathsf{nf}\text{)} \\
\mathsf{nf}(Q \mid P)
\end{aligned}$$

and

$$\begin{aligned}
\mathsf{nf}(P\,|(Q\,|\,R)) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\nu\tilde{a}_P{:}\nu\tilde{a}_Q{:}\nu\tilde{a}_R{:}(P^{\neg\nu}\,|(Q^{\neg\nu}\,|\,R^{\neg\nu})) &\mathrel{\widehat{\equiv}} \text{(Assoc. of } |\text{ w.r.t. } \mathrel{\widehat{\equiv}}\text{)}\\
\nu\tilde{a}_P{:}\nu\tilde{a}_Q{:}\nu\tilde{a}_R{:}((P^{\neg\nu}\,|\,Q^{\neg\nu})\,|\,R^{\neg\nu}) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\mathsf{nf}((P\,|\,Q)\,|\,R).
\end{aligned}$$

$(\mathbf{0}^+)$, $(\mathbf{0}^|)$, (P^ν) $\mathsf{nf}(P+\mathbf{0}) = \mathsf{nf}(P)$, $\mathsf{nf}(P\,|\,\mathbf{0}) = \mathsf{nf}(P)$ and $\mathsf{nf}(\nu x{:}P) = \mathsf{nf}(P)$ if $x \notin fn\,(P)$ hold by definition of nf.

(C^ν) We need to show that $\mathsf{nf}(\nu x{:}\nu y{:}P) \mathrel{\widehat{\equiv}} \mathsf{nf}(\nu y{:}\nu x{:}P)$.

– If $x \notin fn\,(P)$ then by definition of nf:

$$\mathsf{nf}(\nu x{:}\nu y{:}P) = \mathsf{nf}(\nu y{:}P) = \nu y{:}\mathsf{nf}(P) = \nu y{:}\mathsf{nf}(\nu x{:}P) = \mathsf{nf}(\nu y{:}\nu x{:}P).$$

Similarly, one can show that the equality holds if $y \notin fn\,(P)$.

– If $x, y \in fn\,(P)$ then

$$\begin{aligned}
\mathsf{nf}(\nu x{:}\nu y{:}P) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\nu x{:}\nu y{:}\mathsf{nf}(P) &\mathrel{\widehat{\equiv}} \text{(Axiom } \nu x{:}\nu y{:}P^{nf} \mathrel{\widehat{\equiv}} \nu y{:}\nu x{:}P^{nf}\text{)}\\
\nu y{:}\nu x{:}\mathsf{nf}(P) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\mathsf{nf}(\nu y{:}\nu x{:}P).
\end{aligned}$$

$(SE^|)$ We show that $\mathsf{nf}(\nu x{:}(P\,|\,Q)) \mathrel{\widehat{\equiv}} \mathsf{nf}(P\,|\,\nu x{:}Q)$ if $x \notin fn\,(P)$. As the case $x \notin fn\,(Q)$ is trivial, we only consider the case $x \in fn\,(Q)$. Let $\mathsf{nf}(P) = \nu\tilde{a}_P{:}P^{\neg\nu}$ and $\mathsf{nf}(Q) = \nu\tilde{a}_Q{:}Q^{\neg\nu}$. Then we get:

$$\begin{aligned}
\mathsf{nf}(\nu x{:}(P\,|\,Q)) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\nu x{:}\mathsf{nf}(P\,|\,Q) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\nu x{:}\nu\tilde{a}_P{:}\nu\tilde{a}_Q{:}(P^{\neg\nu}\,|\,Q^{\neg\nu}) &\mathrel{\widehat{\equiv}} \text{(Axiom } \nu x{:}\nu y{:}P^{nf} \mathrel{\widehat{\equiv}} \nu y{:}\nu x{:}P^{nf}\text{)}\\
\nu\tilde{a}_P{:}\nu x{:}\nu\tilde{a}_Q{:}(P^{\neg\nu}\,|\,Q^{\neg\nu}) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\mathsf{nf}(P\,|\,\nu x{:}Q).
\end{aligned}$$

Induction step

Our induction hypothesis is $P \equiv Q \Rightarrow \mathsf{nf}(P) \mathrel{\widehat{\equiv}} \mathsf{nf}(Q)$. By the definition of structural congruence, we need to consider $\pi.P \equiv \pi.Q$, $P + R \equiv Q + R$, $\nu a{:}P \equiv \nu a{:}Q$ and $P\,|\,R \equiv Q\,|\,R$, and show the following using the induction hypothesis:

$$\begin{aligned}
\mathsf{nf}(\pi.P) &\mathrel{\widehat{\equiv}} \mathsf{nf}(\pi.Q)\\
\mathsf{nf}(P+R) &\mathrel{\widehat{\equiv}} \mathsf{nf}(Q+R)\\
\mathsf{nf}(\nu a{:}P) &\mathrel{\widehat{\equiv}} \mathsf{nf}(\nu a{:}Q)\\
\mathsf{nf}(P\,|\,R) &\mathrel{\widehat{\equiv}} \mathsf{nf}(Q\,|\,R)\ .
\end{aligned}$$

Prefixing Consider the prefixing $\pi.P \equiv \pi.Q$:

$$\begin{aligned}
\mathsf{nf}(\pi.P) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\pi.\mathsf{nf}(P) &\mathrel{\widehat{\equiv}} \text{(Ind. hyp. and Def. of } \mathrel{\widehat{\equiv}}\text{)}\\
\pi.\mathsf{nf}(Q) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\mathsf{nf}(\pi.Q).
\end{aligned}$$

Choice Consider the choice composition $P + R \equiv Q + R$:

$$\begin{aligned}
\mathsf{nf}(P+R) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\mathsf{nf}(P) + \mathsf{nf}(R) &\mathrel{\widehat{\equiv}} \text{(Ind. hyp. and Def. of } \mathrel{\widehat{\equiv}}\text{)}\\
\mathsf{nf}(Q) + \mathsf{nf}(R) &= \text{(Def. of } \mathsf{nf}\text{)}\\
\mathsf{nf}(Q+R).
\end{aligned}$$

Restriction Consider the restriction $\nu a{:}P \equiv \nu a{:}Q$:

– if $a \notin fn(P) = fn(Q)$ then

$$\begin{aligned} \mathsf{nf}(\nu a{:}P) &= \text{(Def. of } \mathsf{nf}) \\ \mathsf{nf}(P) &\mathrel{\widehat{\equiv}} \text{(Ind. hyp.)} \\ \mathsf{nf}(Q) &= \text{(Def. of } \mathsf{nf}) \\ \mathsf{nf}(\nu a{:}Q). \end{aligned}$$

– if $a \in fn(P) = fn(Q)$ then

$$\begin{aligned} \mathsf{nf}(\nu a{:}P) &= \text{(Def. of } \mathsf{nf}) \\ \nu a{:}\mathsf{nf}(P) &\mathrel{\widehat{\equiv}} (*) \\ \nu a{:}\mathsf{nf}(Q) &= \text{(Def. of } \mathsf{nf}) \\ \mathsf{nf}(\nu a{:}Q). \end{aligned}$$

(*) By Lemma 1(b), $\mathsf{nf}(P) \equiv P$. Since $fn(\_)$ is invariant under $\equiv$, $fn(\mathsf{nf}(P)) = fn(P)$, and so $a \in fn(\mathsf{nf}(P)) = fn(\mathsf{nf}(Q))$. Hence the axiom $P^{nf} \mathrel{\widehat{\equiv}} Q^{nf} \wedge a \in fn\left(P^{nf}\right) \cap fn\left(Q^{nf}\right) \Rightarrow \nu a{:}P^{nf} \mathrel{\widehat{\equiv}} \nu a{:}Q^{nf}$ in the definition of $\mathrel{\widehat{\equiv}}$ can be applied to yield the desired equivalence.

Parallel composition Consider $P \mid R \equiv Q \mid R$. We assume that none of P, Q, R is $\mathbf{0}$ (otherwise the property is trivial). Let $\mathsf{nf}(P) = \nu\tilde{a}_P{:}P^{\neg\nu} \mathrel{\widehat{\equiv}} \nu\tilde{a}_Q{:}Q^{\neg\nu} = \mathsf{nf}(Q)$ and $\mathsf{nf}(R) = \nu\tilde{a}_R{:}R^{\neg\nu}$. The definition of $\mathrel{\widehat{\equiv}}$ ensures that there is a substitution σ that renames $\tilde{a}_P$ to $\tilde{a}_Q$, so that $P^{\neg\nu}\sigma \mathrel{\widehat{\equiv}} Q^{\neg\nu}$.

$$\begin{aligned} \mathsf{nf}(P \mid R) &= \text{(Def. of } \mathsf{nf}) \\ \nu\tilde{a}_P{:}\nu\tilde{a}_R{:}(P^{\neg\nu} \mid R^{\neg\nu}) &\mathrel{\widehat{\equiv}} (\alpha\text{-conversion using } \sigma) \\ \nu\tilde{a}_Q{:}\nu\tilde{a}_R{:}(P^{\neg\nu}\sigma \mid R^{\neg\nu}\sigma) &= (R^{\neg\nu}\sigma = R^{\neg\nu} \text{ due to NoClash}) \\ \nu\tilde{a}_Q{:}\nu\tilde{a}_R{:}(P^{\neg\nu}\sigma \mid R^{\neg\nu}) &\mathrel{\widehat{\equiv}} (P^{\neg\nu}\sigma \mathrel{\widehat{\equiv}} Q^{\neg\nu}, \text{ axiom } P^{\neg\nu} \mathrel{\widehat{\equiv}} Q^{\neg\nu} \Rightarrow P^{\neg\nu} \mid R^{\neg\nu} \mathrel{\widehat{\equiv}} Q^{\neg\nu} \mid R^{\neg\nu}) \\ \nu\tilde{a}_Q{:}\nu\tilde{a}_R{:}(Q^{\neg\nu} \mid R^{\neg\nu}) &= \text{(Def. of } \mathsf{nf}) \\ \mathsf{nf}(Q \mid R). \end{aligned}$$

This concludes the proof. □

4 Variants

When defining the semantics of π-Calculus, one has a choice what to present as SOS rules and what to put as axioms defining the structural congruence. Hence several variants of structural congruence are used in practice, and in this section we show that our result can be extended to some of them.

The variant of structural congruence presented in Section 2.1 was rather strong, i.e., it equates rather few terms. In practice one would often want as weak a structural congruence as possible, as making more terms structurally congruent reduces the size of the reachability graph of a π-Calculus process (where the states are the equivalence classes w.r.t. $\equiv$). Similarly, in the translation of π-Calculus to Petri nets proposed [Mey07] a weaker structural congruence reduces the size of the resulting Petri net.

In this section we show that adding some common axioms to the ones described in Section 2.1 does not affect our result. In particular, we consider the following axioms that extend the scope extrusion to the choice and prefixing operators:

$$\nu x{:}(P + Q) \equiv P + \nu x{:}Q, \text{ if } x \notin fn(P) \qquad (SE^{+})$$

$$\nu x{:}\pi.P \equiv \pi.\nu x{:}P, \text{ if } x \text{ does not occur in } \pi \qquad (SE^{\bullet})$$

The axiom (SE^+) is often used in the literature [Par01]; in fact, one can argue that for the π-Calculus syntax presented in Section 2.1, where choices are not guarded, the structural congruence including this axiom is more natural than the one presented in Section 2.1. Furthermore, adding both (SE^+) and $(SE^\bullet)$ is very useful in practice, as any term P becomes structurally congruent to a term of the form $\nu\tilde{a}{:}P'$, where P' does not contain any restrictions.

One can see that the GI to πSC reduction in Section 3 still works if any subset of the above axioms is added to the definition of structural congruence. The ideas of the πSC to TE reduction in Section 3 can also be largely reused in the presence of (any subset of) the new axioms. However, for the proof in Section 3.3 to proceed, the following changes have to be made to the definitions of the normal form and the function nf, as well as to the definition of the normal form equivalence $\widehat{\equiv}$:

Definition of normal form In Definition 2, $P^{nf} + P^{nf}$ should be replaced by $P^{\neg\nu} + P^{\neg\nu}$ if (SE^+) is used, and $\pi.P^{nf}$ should be replaced by $\pi.P^{\neg\nu}$ if $(SE^\bullet)$ is used.

Definition of nf The rule for sums should be replaced by

$$\mathsf{nf}(P+Q) \stackrel{\text{df}}{=} \begin{cases} \mathbf{0} & \text{if } \mathsf{nf}(P) = \mathbf{0} = \mathsf{nf}(Q) \\ \mathsf{nf}(P) & \text{if } \mathsf{nf}(P) \neq \mathbf{0} = \mathsf{nf}(Q) \\ \mathsf{nf}(Q) & \text{if } \mathsf{nf}(P) = \mathbf{0} \neq \mathsf{nf}(Q) \\ \nu\tilde{a}_P{:}\nu\tilde{a}_Q{:}(P^{\neg\nu} + Q^{\neg\nu}) & \text{if } \mathsf{nf}(P) = \nu\tilde{a}_P{:}P^{\neg\nu} \wedge \mathsf{nf}(Q) = \nu\tilde{a}_Q{:}Q^{\neg\nu} \end{cases}$$

if (SE^+) is used, and the rule for prefixing should be replaced by

$$\mathsf{nf}(\pi.P) \stackrel{\text{df}}{=} \nu\tilde{a}{:}\pi.P^{\neg\nu} \quad \text{if } \mathsf{nf}(P) = \nu\tilde{a}{:}P^{\neg\nu}$$

if $(SE^\bullet)$ is used. (Note that in the above definitions $\tilde{a}_P$, $\tilde{a}_Q$ and $\tilde{a}$ can be empty; moreover, in the latter definition there is no need to check whether the names in $\tilde{a}$ occur in π due to the NoClash assumption.)

Definition of $\widehat{\equiv}$ The axiom for sums should be replaced by

$$P^{\neg\nu} \mathrel{\widehat{\equiv}} Q^{\neg\nu} \Rightarrow P^{\neg\nu} + R^{\neg\nu} \mathrel{\widehat{\equiv}} Q^{\neg\nu} + R^{\neg\nu}$$

if (SE^+) is used, and the axiom for prefixing should be replaced by

$$P^{\neg\nu} \mathrel{\widehat{\equiv}} Q^{\neg\nu} \Rightarrow \pi.P^{\neg\nu} \mathrel{\widehat{\equiv}} \pi.Q^{\neg\nu}$$

if $(SE^\bullet)$ is used.

In the proof of Lemma 2 the new axioms have to be considered in the 'Axioms' section; the proof for (SE^+) is very similar to the proof for $(SE^|)$, and the proof for $(SE^\bullet)$ is as follows.

We show that $\mathsf{nf}(\nu x{:}\pi.P) \mathrel{\widehat{\equiv}} \mathsf{nf}(\pi.\nu x{:}P)$ if x does not occur in π. Let $\mathsf{nf}(P) = \nu\tilde{a}{:}P^{\neg\nu}$. Then we get:

- If $x \notin fn(P)$ then $x \notin fn(\pi.P)$, which validates the first of the following equalities. For the second one we use the adapted definition of nf, and the last one holds with the observation $\mathsf{nf}(\nu x{:}P) = \mathsf{nf}(P) = \nu\tilde{a}{:}P^{\neg\nu}$. So we have:

$$\mathsf{nf}(\nu x{:}\pi.P) = \mathsf{nf}(\pi.P) = \nu\tilde{a}{:}\pi.P^{\neg\nu} = \mathsf{nf}(\pi.\nu x{:}P).$$

- If $x \in fn(P)$ then by definition of nf:

$$\mathsf{nf}(\nu x{:}\pi.P) = \nu x{:}\mathsf{nf}(\pi.P) = \nu x{:}\nu\tilde{a}{:}\pi.P^{\neg\nu} = \mathsf{nf}(\pi.\nu x{:}P).$$

In the induction step, if (SE^+) is used then the proof for the case of choice becomes very similar to the one for the case of parallel composition, and if $(SE^\bullet)$ is used then the proof for the case of prefixing changes as follows. Consider the prefixing $\pi.P \equiv \pi.Q$. Let $\mathsf{nf}(P) = \nu\tilde{a}_P{:}P^{\neg\nu}$

and $\mathsf{nf}(Q) = \nu\tilde{a}_Q{:}Q^{\neg\nu}$. The definition of $\widehat{\equiv}$ ensures that there is a substitution σ that renames $\tilde{a}_P$ to $\tilde{a}_Q$, so that $P^{\neg\nu}\sigma \mathrel{\widehat{\equiv}} Q^{\neg\nu}$. So we have:

$$\begin{aligned}
\mathsf{nf}(\pi.P) &= \text{(Def. of } \mathsf{nf}\text{)} \\
\nu\tilde{a}_P{:}\pi.P^{\neg\nu} &\mathrel{\widehat{\equiv}} (\alpha\text{-conversion using } \sigma) \\
\nu\tilde{a}_Q{:}\pi\sigma.P^{\neg\nu}\sigma &= (\pi\sigma = \pi \text{ by NoClash assumption}) \\
\nu\tilde{a}_Q{:}\pi.P^{\neg\nu}\sigma &\mathrel{\widehat{\equiv}} (P^{\neg\nu}\sigma \mathrel{\widehat{\equiv}} Q^{\neg\nu}, \text{ axiom } P^{\neg\nu} \mathrel{\widehat{\equiv}} Q^{\neg\nu} \Rightarrow \pi.P^{\neg\nu} \mathrel{\widehat{\equiv}} \pi.Q^{\neg\nu}) \\
\nu\tilde{a}_Q{:}\pi.Q^{\neg\nu} &= \text{(Def. of } \mathsf{nf}\text{)} \\
\mathsf{nf}(\pi.Q).
\end{aligned}$$

5 Optimisation

Our reduction of πSC to TE (and ultimately to GI) is intended not only as a theoretical result, but also to be applied for checking π-Calculus structural congruence in our tool Petruchio [SM08]. In this section we recall the original reduction of [Bas94] from TE to GI, and show how it can be optimised, by exploiting specific features of π-Calculus terms. After the optimisations are performed, an off-the-shelf GI solver can be used to decide the isomorphism.

5.1 Reducing TE to LDGI

In [Bas94] a Karp reduction from TE to label-preserving digraph isomorphism (LDGI)[7] is given, and the latter problem is known to be Karp reducible to GI [BC79]. It works by building an augmented parse tree of the term, compounding the vertices corresponding to binary associative and associative-commutative operations into vertices with larger out-degree, and linking the vertices corresponding to the quantifiers with those corresponding to the variables bound by these quantifiers; then the labels of the latter vertices are erased, to reflect the fact that the precise names of bound variables are not important (as they can always be changed by α-conversion). We formally explain the construction and give an example in Figure 2.

The following recursive procedure is applied to the term t, yielding a rooted labelled acyclic digraph G_t. It uses the auxiliary function $\mathit{frontier}_f$ defined in Section 2.2, which gathers all the operands needed for compounding a binary associative operator f into an operator of greater arity.

Case 1: If t is a constant c then G_t consists of a single vertex labelled by c.
Case 2: If t is a variable x then G_t consists of a single vertex labelled by x.
Case 3: If $t = f(t_1, \ldots, t_n)$ then:

- *Case 3a:* If f is neither associative nor commutative then G_t is rooted at a new vertex labelled by f, with the arcs from it to the roots of $G_{t_1}, \ldots, G_{t_n}$, and the arc pointing to the root G_{t_i} is labelled by i, for each $i \in \{1, \ldots, n\}$.
- *Case 3b:* If f is commutative and not associative ($n = 2$) then the construction is the same as in Case 3a, but the arcs are not labelled.
- *Case 3c:* If f is associative and not commutative ($n = 2$) then let $[t'_1, \ldots, t'_k] = \mathit{frontier}_f(t)$. Then G_t is rooted at a new vertex labelled by f, with the arcs from it to the roots of $G_{t'_1}, \ldots, G_{t'_k}$, and the arc pointing to the root of $G_{t'_i}$ is labelled by i, for each $i \in \{1, \ldots, k\}$.
- *Case 3d:* If f is both associative and commutative ($n = 2$) then the construction is the same as in Case 3c, but the arcs are not labelled.

[7] The labels can be attached to both vertices and arcs, and some vertices and/or arcs may be unlabelled.

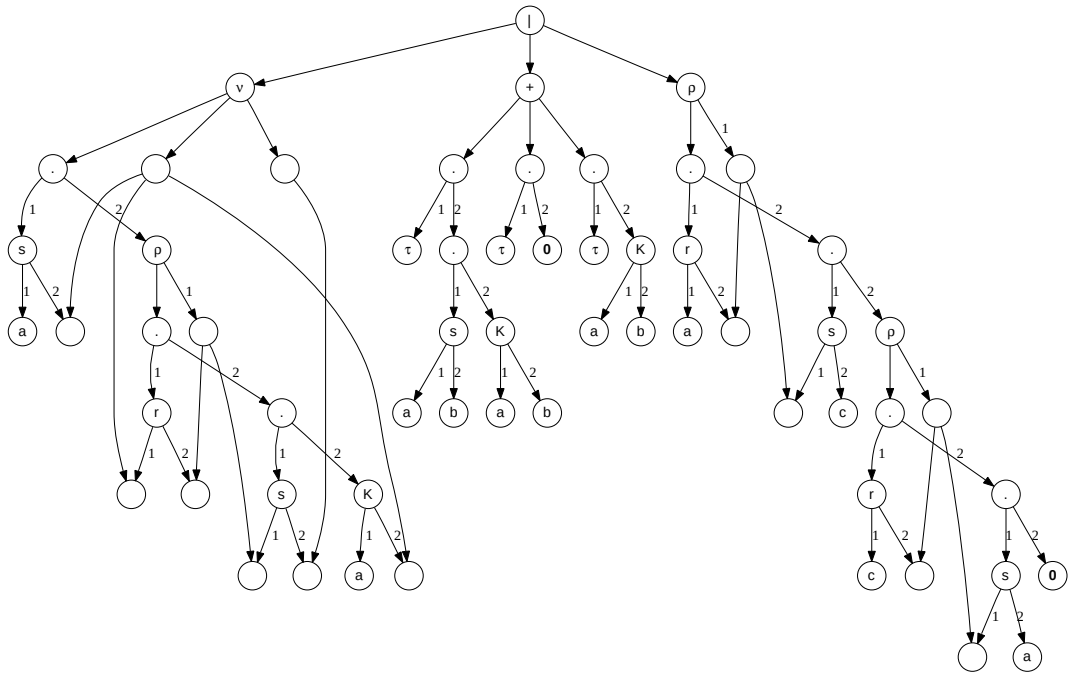


Fig. 2. Reduction from TE to LDGI.

Case 4: If $t = \theta x_1 : \ldots \theta x_n : t'$, where θ is a quantifier, then:

Case 4a: If θ is not commutative then G_t is rooted at a new vertex labelled by θ, with the arcs from it to the root of $G_{t'}$ and to the new unlabelled vertices $w_1, \ldots, w_n$, with the arc pointing to w_i labelled by i, for each $i \in \{1, \ldots, n\}$. Moreover, arcs are added from each w_i to each vertex in $G_{t'}$ labelled by x_i, and the latter 'x_i'-labels are erased.

Case 4b: If θ is commutative then the construction is the same as in Case 4a, but the arcs pointing to w_i are not labelled.

One can easily show that the size of the resulting digraph is linear in the size of the term.

The construction is illustrated in Figure 2 for the term

$$\nu x{:}\nu y{:}\overline{a}\langle x\rangle.x(z).\overline{z}\langle y\rangle.K\lfloor a, x\rfloor \mid \tau.\overline{a}\langle b\rangle.K\lfloor a, b\rfloor + \tau.\mathbf{0} + \tau.K\lfloor a, b\rfloor \mid a(p).\overline{p}\langle c\rangle.c(q).\overline{q}\langle a\rangle.\mathbf{0}\,,$$

which is re-written into

$$\nu x{:}\nu y{:}s(a, x).\rho z{:}r(x, z).s(z, y).K(a, x) \mid \tau.s(a, b).K(a, b) + \tau.\mathbf{0} + \tau.K(a, b) \mid$$
$$\rho p{:}r(a, p).s(p, c).\rho q{:}r(c, q).s(q, a).\mathbf{0}$$

as described in Section 3.2. Note that in this translation of π-Calculus terms, non-commutative ρ-quantifiers represent the binding in input prefixes, viz. the binding of y in $x(y).P$ is represented by $\rho y{:}r(x, y).P$. Hence in Case 4a in Basin's construction there will never be a series of two or more ρ-quantifiers, i.e., $n = 1$. Also note that since the binary prefixing operator '.' is not associative, its adjacent occurrences are not compounded into an operator of greater arity. However, in the following section we show that Basin's construction can be optimised in the case of π-Calculus terms by fusing adjacent '.'-labelled vertices in the digraph, i.e., essentially by treating prefixing as an associative but not commutative operator.

5.2 Reducing the size of digraph

We now describe how to optimise the reduction from TE to LDGI described above by exploiting specific properties of π-Calculus terms. While doing so, one has to be careful that such

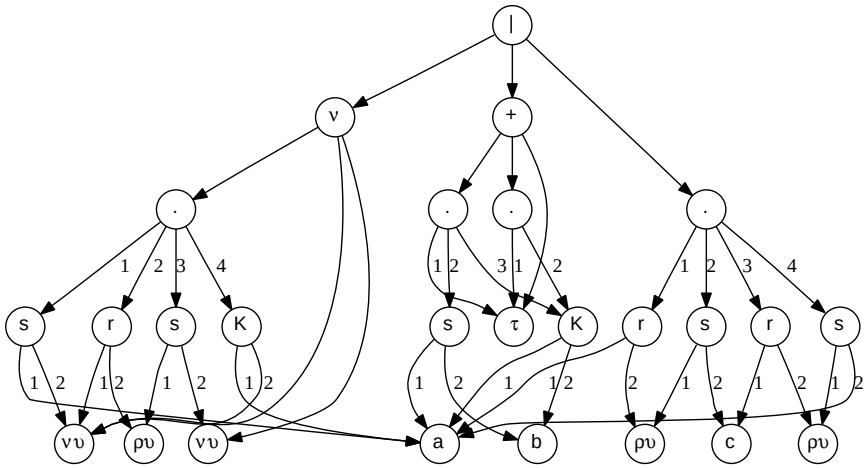


Fig. 3. Optimised reduction.

optimisations are applied consistently to both terms that are to be compared, i.e., the following correctness requirement holds [BC79]:

> $T(G)$ and $T(G')$ are isomorphic iff G and G' are isomorphic, where T is a transformation.

In our case, it is convenient to replace this requirement by the following sufficient condition [BC79]:

> G and $T(G)$ can be recovered from each other up to isomorphism. (*)

Furthermore, the transformation T should be computable in polynomial time.

Below we list a number of transformations allowing one to significantly reduce the size of LDGI in practice. Figure 3 illustrates the result of applying the optimisations described below. One can see that the overall effect of applying them is the reduction in the size of the digraph from 60/63 down to 26/38 vertices/arcs.

Sub-terms sharing The first idea comes from the area of term unification [BS01]: the digraph constructed in Section 5.1 is essentially an augmented parse tree of the term, and so one can identify the sub-terms that occur more than once and remove the duplicates by sharing the corresponding subgraphs. For example, there are three vertices with the in-degree one labelled by a constant τ in the digraph in Figure 2; their fusion yields a single τ-labelled vertex with the in-degree three, as illustrated in Figure 3.

However, there is a number of issues arising when performing such a transformation. First, in the presence of commutative quantifiers it is computationally expensive to check if two sub-terms are equivalent (indeed, we get another instance of TE). Since such checks have to be performed multiple times to identify all the shared sub-terms, the approach becomes impractical.

A possible solution is to limit term sharing only to certain sub-terms, for which such a check is easy. When doing this, one should be careful not to violate the correctness requirement (*), i.e., one has to ensure that the transformations are applied consistently to the two digraphs forming the instance of LDGI at hand, i.e., if the two digraphs are isomorphic then the same sub-terms should be shared in them, so that the resulting optimised digraphs are also isomorphic. This can be achieved for sub-terms not containing commutative quantifiers, as for such sub-terms one can use the efficiently computable canonical form described in Section 2.2; moreover, the correctness requirement (*) will be satisfied, since such a sub-term will never be

equivalent to any sub-term containing a commutative quantifier. Alternatively, one could settle for an even simpler approach of sharing only trivial sub-terms, i.e., fusing only the vertices corresponding to the same constants and to the same variables (since the labels of the vertices corresponding to variables are erased, the algorithm has to remember to which variables they used to correspond). Since most of the vertices of a tree are leaves, sharing trivial sub-terms gives good results in practice, as usually few non-trivial sub-terms can be shared; e.g., in our example the only non-trivial sub-term that can be shared is $K\lfloor a, b\rfloor$.

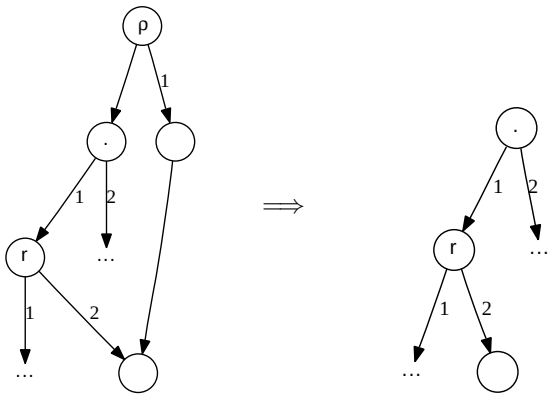
Another issue is that fusing vertices can create parallel arcs, turning a digraph into a multigraph. However, this problem can be easily solved as follows:

- The unlabelled arcs from each of the vertices w_i to ones corresponding to some variable, created in Case 4 of the construction in Section 5.1, will become parallel after vertices corresponding to the same variable are fused. These parallel arcs can then be replaced by a single unlabelled arc.
- Other parallel arcs can be replaced by a single arc labelled by a multiset of labels of the parallel arcs (we assume that all the unlabelled arcs actually have a special label '$\square$'). For example, if there are three labelled arcs from v to w with the labels a, a, b plus two unlabelled arcs, we replace them by a single arc with the label $\{a, a, b, \square, \square\}$.

Note that these transformations do not lose any information, i.e., the original multigraph can be recovered from the transformed one (up to isomorphism), and so the correctness condition (*) is fulfilled.

Sharing sub-terms is advantageous not only because it reduces the size of the digraph, but also because it reduces a number of possible vertex matches that a GI solver has to consider. Indeed, in Figure 2 there are seven vertices labelled by a, which have the same in- and out-degree; when trying to find an isomorphism between two such digraphs, a GI solver will have to consider 49 potential matches for these vertices. On the other hand, after the common sub-terms are shared, these digraphs will have only a single a-labelled vertex each, which gives an unambiguous match.

Removal of ρ-labelled vertices One can observe that the position of ρ-labelled vertices can always be recovered from the corresponding r-labelled vertices, and so ρ-labelled vertices, as well as the auxiliary vertices created in Case 4 of the construction in Section 5.1, can be removed as shown in the picture below:



Note that the vertex corresponding to the variable bound by the quantifier is also the second son of the r-labelled vertex, and so no information is lost.

This transformation not only decreases the size of the digraph, but also increases the efficiency of the optimisation compounding adjacent '.'-labelled vertices described below, as ρ-labelled vertices often separate '.'-labelled ones.

Contraction of the auxiliary vertices for ν quantifiers One can see that the above transformation does not generally work for ν-labelled vertices; however, a partial optimisation can still be achieved. Indeed, after the common sub-terms have been shared and the parallel arcs have been removed, the auxiliary vertices w_i of ν quantifiers, created in Case 4b of the construction in Section 5.1, have the in- and out-degree one, and can be contracted without loss of information. That is, the auxiliary vertices are removed, and the ν-labelled vertex becomes connected to all the vertices corresponding to the variables bound by the series of ν quantifiers represented by this ν-labelled vertex.

Furthermore, if the digraph is such that it has a single ν-labelled vertex that is positioned at the root (and this is always the case if (SE^{+}) and $(SE^{\bullet})$ are added to the definition of structural congruence) then this vertex can be removed (together with all the outgoing arcs). Indeed, it is connected only to the vertices corresponding to the restricted variables and to the vertex v representing the process in the scope of the restriction, and v becomes the new root of the digraph.

Compounding adjacent '.'-labelled vertices The first son of a vertex corresponding to the prefixing operator '.' is always labelled by either s or r or τ. In particular, it is never another '.'-labelled vertex. Hence, without loss of information, one can iteratively replace adjacent '.'-labelled vertices by a single vertex with a larger out-degree. This is essentially equivalent to treating '.' as an associative but not commutative operation in Case 3 of the construction in Section 5.1, i.e., for a term t with the top-level operation '.', a '.'-labelled vertex is created, with the numbered arcs from it to the roots of the digraphs corresponding to the terms in $\mathit{frontier}_{\bullet}(t)$.

Note that all the '.'-labelled vertices in the digraph in Figure 2 have the in-degree one and out-degree two, and so a GI solver will have to consider a lot of potential matches between such vertices in the two digraphs comprising an instance of LDGI. Hence, compounding adjacent '.'-labelled vertices is advantageous not only because it reduces the size of the digraph, but also because it creates 'hairy' vertices of varying out-degree, reducing the number of matches that the GI solver has to consider, as only the vertices with equal in- and out-degrees are considered for matching.

Eliminating the 0-labelled vertex One can observe that the **0**-labelled vertex $v_{\mathbf{0}}$ can be removed from the digraph without loss of information, together with all the incoming arcs. Indeed, since the original π-Calculus process was in normal form, all the parents of $v_{\mathbf{0}}$ are (perhaps, compounded) '.'-labelled vertices, and $v_{\mathbf{0}}$ is the last son of each of these vertices. Hence, to restore the original digraph from the one where $v_{\mathbf{0}}$ was deleted, it is enough to look at the last son of each '.'-labelled vertex w, and if its label is s, r or τ then the arc $(w, v_{\mathbf{0}})$ is added to the digraph and labelled so that $v_{\mathbf{0}}$ becomes the last child of w.

Furthermore, if after removal of $v_{\mathbf{0}}$ some of its parents have a single outgoing arc then they can be contracted without loss of information.

5.3 Eliminating arc labels

Current tools (e.g., NAUTY [NAU, McK]) can check isomorphism of graphs and digraphs with labelled vertices. Hence, in practice it does not make sense to reduce LDGI all way down to GI[8] — it is sufficient to get rid of arc labels. In fact, vertex labels and directed arcs are often beneficial for isomorphism checking, as they reduce the number of potential matches that have to be considered by a GI solver. Note that the optimisations described above generally increase the colours/vertices ratio, which has a positive effect on the performance of a GI solver. To further increase this ratio, one can observe that unlabelled vertices corresponding to the variables should never be matched if the corresponding variables were bound by quantifiers of different types. Hence, in Figure 3 we label such vertices by either $\rho\upsilon$ (if the corresponding

[8] As this comes at the price of increasing the size of the graph.

variable was bound by ρ) or $\nu\upsilon$ (if the corresponding variable was bound by ν), so that no unlabelled vertices remain in the digraph.

Now the arc labels can be eliminated as follows (these transformations are not shown in Figure 3):

- If the destination of a labelled arc has the in-degree one then the arc label is paired with the vertex label, i.e., the arc label is erased and the vertex label becomes (l, l'), where l was the arc label and l' was the vertex label.
- Let v be a compound '.'-labelled vertex whose last son was not the removed **0**-labelled vertex. Then the last son of v can always be distinguished from its other sons, as it will correspond to a process whereas the other sons will correspond to prefixes (i.e., their labels will be either s or r or τ). Hence, the label on the arc pointing to this last son can be erased. Moreover, if v has the out-degree two (i.e., it is not a result of compounding of several '.'-labelled vertices) then the label on the other out-arc can also be erased.
- In all other cases a labelled arc is subdivided by an additional vertex, and the label is transferred to this vertex. Note that since all the arc labels are integers, whereas no vertex is labelled by an integer, a GI solver will never attempt matching the vertices created by arc subdivision with other vertices.

6 Conclusion

In this paper we have shown that πSC is a $\mathcal{GI}$-complete problem. This have been achieved by providing Karp reductions from GI to πSC and from πSC to TE, with TE being known to be Karp reducible to GI. Furthermore, this result is rather robust — as it also holds for alternative definitions of π-Calculus structural congruence, and also for some rather restrictive fragments of π-Calculus. We also identified fragments of π-Calculus for which πSC is in $\mathcal{P}$, viz. terms without restriction or without both + and |.

On the practical side, we have proposed a number of optimisations that allow one to significantly reduce the size of the resulting digraphs that have to be checked for isomorphism. The approach for solving πSC presented in this paper will be used in the Petruchio tool [SM08] for π-Calculus verification. It first translates a π-Calculus process into a Petri net [Mey07], and then verifies the latter using an efficient model checker [MKS08]. πSC has to be solved often during the translation process, and the results of this paper allow one to use efficient off-the-shelf graph isomorphism checkers to optimise this task.

The only related work on computing the structural congruence relation of the π-Calculus has been done by Engelfriet and Gelsema [EG99, EG04b, EG04a, EG07]. While we investigate the problem of efficiently checking the structural congruence that does not expand calls to process identifiers, these authors deal with the classical problem of whether Milner's structural congruence [Mil99] for π-Calculus with the replication operator '!' (and the axiom $!P \equiv P\,|!P$) is decidable. In [EG99, EG04b] decidable variants of Milner's structural congruence are proposed, while in [EG04a, EG07] the decidability of Milner's structural congruence is established for restricted fragments of π-Calculus. The question whether Milner's structural congruence is decidable for the full π-Calculus with replication is to the best of our knowledge still open.

As a future work we would like to investigate how adding some other axioms to the definition of the structural congruence affects our result. In particular, the following axioms look interesting:

$$\nu x{:}\pi.P \equiv \mathbf{0} \quad \text{if } \pi \text{ has the form } \overline{x}\langle\cdot\rangle \text{ or } x(\cdot) \qquad (P^{\nu\pi})$$

$$\nu x{:}(P+Q) \equiv \nu x{:}P + \nu x{:}Q \qquad (D^{\nu+})$$

The former axiom allows for replacing sequential processes waiting for communication on a hidden channel that is unknown to the environment by **0**, as such processes have no behaviour in any context. (It would also be interesting to generalise $(P^{\nu\pi})$ to an axiom replacing any process that has no behaviour in any context by **0**.) The latter axiom allows the restriction

to distribute over +; observe that (SE^+) is just a special case of this axiom, i.e., (SE^+) can be derived from $(D^{\nu+})$ and (P^ν). (Note that applying this axiom can violate the NoClash assumption, as x becomes bound twice; to prevent this, one can replace $\nu x{:}Q$ by $\nu y{:}Q\{y/x\}$ in it.)

One can see that $\mathcal{GI}$-hardness of πSC still holds with any subset of these axioms (and optionally the axioms in Section 4), with the reduction from GI to πSC described in Section 3.1; the only subtlety is that in the presence of $(D^{\nu+})$, the parallel composition can no longer be replaced by choice in the resulting term.

However, the membership of the resulting πSC problems in $\mathcal{GI}$ is still an open issue. We are confident that the argument developed in Section 3.3 can be generalised so that $(P^{\nu\pi})$ is included into the set of allowed axioms without leaving $\mathcal{GI}$. The main idea is to minimise the scope of each restriction in turn using the scope extrusion axiom(s) and (C^ν), and apply $(P^{\nu\pi})$ if possible; this should eliminate the need to use $(P^{\nu\pi})$ for proving the structural congruence of two terms modified in this way, and so the technique described in Section 3.3 (with the amendments described in Section 4 if (SE^+) and/or $(SE^\bullet)$ is used) can be subsequently applied.

The situation with $(D^{\nu+})$ is less clear, but there is a chance that it can also be included into the set of allowed axioms without leaving $\mathcal{GI}$. One can first minimise the scope of all restrictions and then use a normal form function similar to the one described in Section 3.3 (with the amendments described in Section 4 if $(SE^\bullet)$ is used — note that (SE^+) is superfluous in the presence of $(D^{\nu+})$). The definition of nf needs to be modified as follows, to re-use restricted names:

$$\mathsf{nf}(P+Q) \stackrel{\text{df}}{=} \begin{cases} \mathbf{0} & \text{if } \mathsf{nf}(P) = \mathbf{0} = \mathsf{nf}(Q) \\ \mathsf{nf}(P) & \text{if } \mathsf{nf}(P) \neq \mathbf{0} = \mathsf{nf}(Q) \\ \mathsf{nf}(Q) & \text{if } \mathsf{nf}(P) = \mathbf{0} \neq \mathsf{nf}(Q) \\ \nu\tilde{a}_P{:}(P^{\neg\nu} + Q^{\neg\nu}\sigma_P) & \text{if } \mathsf{nf}(P) = \nu\tilde{a}_P{:}P^{\neg\nu} \wedge \mathsf{nf}(Q) = \nu\tilde{a}_Q{:}Q^{\neg\nu} \wedge |\tilde{a}_P| \geq |\tilde{a}_Q| \\ \nu\tilde{a}_Q{:}(P^{\neg\nu}\sigma_Q + Q^{\neg\nu}) & \text{if } \mathsf{nf}(P) = \nu\tilde{a}_P{:}P^{\neg\nu} \wedge \mathsf{nf}(Q) = \nu\tilde{a}_Q{:}Q^{\neg\nu} \wedge |\tilde{a}_P| < |\tilde{a}_Q|, \end{cases}$$

where σ_P substitutes the names in $\tilde{a}_Q$ by the last $|\tilde{a}_Q|$ names in $\tilde{a}_P$, and σ_Q is defined similarly. Intuitively, in the last two cases in this definition we try to re-use the restricted names as much as possible. The resulting process will have $\max(|\tilde{a}_P|, |\tilde{a}_Q|)$ restrictions at the front (which yields the $|\tilde{a}_P| \geq (<)|\tilde{a}_Q|$ distinction), and in the sub-process with fewer restricted names at the front, these names are renamed. For example, the normal form of $\nu x{:}K\lfloor x\rfloor + \nu y{:}L\lfloor y\rfloor$ becomes $\nu x{:}(K\lfloor x\rfloor + L\lfloor x\rfloor)$ rather than $\nu x{:}\nu y{:}(K\lfloor x\rfloor + L\lfloor y\rfloor)$ obtained with the weaker axiom (SE^+).

Acknowledgements This research was supported by the Royal Academy of Engineering/ Epsrc post-doctoral research fellowship EP/C53400X/1 (Davac) and the German Research Council (DFG) as a part of the Graduate School GRK 1076/1 (TrustSoft).

References

[Bas94] D.A. Basin. A term equality problem equivalent to graph isomorphism. *Information Processing Letters*, 51(2):61–66, 1994.

[BC79] K.S. Booth and C.J. Colbourn. Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-04, University of Waterloo, 1979.

[BS01] F. Baader and W. Snyder. Unification theory. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, pages 447–533. Elsevier Science Publishers, 2001. URL: `http://lat.inf.tu-dresden.de/research/papers/2001/BaaderSnyderHandbook.ps.gz`.

[Cai04] L. Caires. Behavioural and spatial observations in a logic for the π-Calculus. In *Proc. FOSSACS'04*, volume 2987 of *Lecture Notes in Computer Science*, pages 72–89. Springer-Verlag, 2004.

[EG99] J. Engelfriet and T. Gelsema. Multisets and structural congruence of the π-Calculus with replication. *Theoretical Computer Science*, 211(1-2):311–337, 1999.

[EG04a] J. Engelfriet and T. Gelsema. The decidability of structural congruence for replication restricted π-Calculus processes. Technical report, Leiden Institute of Advanced Computer Science, 2004. Revised 2005.

[EG04b] J. Engelfriet and T. Gelsema. A new natural structural congruence in the π-Calculus with replication. *Acta Informatica*, 40(6):385–430, 2004.

[EG07] J. Engelfriet and T. Gelsema. An exercise in structural congruence. *Information Processing Letters*, 101(1):1–5, 2007.

[FGMP03] G.-L. Ferrari, S. Gnesi, U. Montanari, and M. Pistore. A model-checking verification environment for mobile processes. *ACM Transactions on Software Engineering and Methodology*, 12(4):440–473, 2003.

[KST93] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhauser Verlag, 1993.

[McK] B.D. McKay. NAUTY *Users Guide (Version 2.2)*. Computer Science Department, Australian National University.

[Mey07] R. Meyer. A theory of structural stationarity in the π-Calculus. *Under revision*, 2007.

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[MKS08] R. Meyer, V. Khomenko, and T. Strazny. A practical approach to verification of mobile systems using net unfoldings. In R. Valk and K. van Hee, editors, *Proc. ATPN'08*, Lecture Notes in Computer Science. Springer-Verlag, 2008. To appear.

[NAU] The NAUTY page. URL: `http://cs.anu.edu.au/~bdm/nauty/`.

[Par01] J. Parrow. An introduction to the π-Calculus. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier Science Publishers, 2001.

[SM08] T. Strazny and R. Meyer. PETRUCHIO page, 2008. URL: `http://petruchio.informatik.uni-oldenburg.de`.

[Sto76] L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[SW01] D. Sangiorgi and D. Walker. *The π-Calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[VM94] B. Victor and F. Moller. The MOBILITY WORKBENCH: A tool for the π-Calculus. In *Proc. CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.