COMPUTING SCIENCE

A Polynomial Translation of pi-Calculus (FCP) to Safe Petri Nets

Roland Meyer, Victor Khomenko and Reiner Hüchting

TECHNICAL REPORT SERIES

No. CS-TR-1323 April 2012

TECHNICAL REPORT SERIES

No. CS-TR-1323 April, 2012

A Polynomial Translation of pi-Calculus (FCP) to Safe Petri Nets

Roland Meyer, Victor Khomenko and Reiner Hüchting

Abstract

We develop a polynomial translation from finite control processes (an important fragment of pi-calculus) to safe low-level Petri nets. To our knowledge, this is the first such translation. It is natural (there is a close correspondence between the control flow of the original specification and the resulting Petri net), enjoys a bisimulation result, and is suitable for practical model checking.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

MEYER, R., KHOMENKO, V., HÜCHTING, R.

A Polynomial Translation of pi-Calculus (FCP) to Safe Petri Nets
[By] R. Meyer, V. Khomenko, R. Hüchting

Newcastle upon Tyne: Newcastle University: Computing Science, 2012.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1323)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1323

Abstract

We develop a polynomial translation from finite control processes (an important fragment of pi-calculus) to safe low-level Petri nets. To our knowledge, this is the first such translation. It is natural (there is a close correspondence between the control flow of the original specification and the resulting Petri net), enjoys a bisimulation result, and is suitable for practical model checking.

About the authors

Roland Meyer is a Junior Professor of Theoretical Computer Science and head of the research group on Concurrency Theory at the University of Kaiserslautern. Previously, from 2009 to 2010, he was a CNRS postdoc in the LIAFA research institute affiliated with the University Paris 7. He obtained his PhD in Computer Science (with distinction) from the University of Oldenburg, where he also studied Computer Science and Mathematics (2001 to 2005). Roland is interested in theoretical aspects of concurrency. His ambition is to understand the principles underlying the behaviour of network applications, remote programs, and multi-threaded software, and exploit them in the design of highly efficient system analysis algorithms. Harnessing methods from computability theory, computer-aided verification, and formal languages, he is specialised in inference techniques for infinite state systems, and currently extends state space exploration algorithms towards automatic system correction and optimisation.

Victor Khomenko obtained his MSc with distinction in Computer Science, Applied Mathematics and Teaching of Mathematics and Computer Science in 1998 from Kiev Taras Shevchenko University, and PhD in Computing Science in 2003 from Newcastle University. He was a Program Committee Chair for the International Conference on Application of Concurrency to System Design (ACSD'10). He also organised the Workshop on UnFOlding and partial order techniques (UFO'07) and Workshop on BALSA Re-Synthesis (RESYN'09). In January 2005 Victor became a Lecturer in the School of Computing Science, Newcastle University, and in September 2005 he obtained a Royal Academy of Engineering / EPSRC Post-doctoral Research Fellowship and worked on the Design and Verification of Asynchronous Circuits (DAVAC) project. After the end of this award, in September 2010, he switched back to Lectureship. Victor’s research interests include model checking of Petri nets, Petri net unfolding techniques, verification and synthesis of self-timed (asynchronous) circuits.

Reiner Hüchting is a PhD student in the Concurrency Theory team at the University of Kaiserslautern. He studied Computer Science at the University of Kaiserslautern from 2001 to 2008 and obtained his Master's with a thesis on the expected size of tree structures. Afterwards he moved to a research assistant position where he supervises lectures and seminars. His research interests include foundational aspects of concurrent systems, in particular the impact of reconfiguration on decidability and complexity of verification problems.

Suggested keywords

PI-CALCULUS
FINITE CONTROL PROCESS
FCP
PETRI NET
MODEL CHECKING

A Polynomial Translation of π-Calculus (FCP) to Safe Petri Nets

Roland Meyer[1], Victor Khomenko[2], and Reiner Hüchting[1]

[1] University of Kaiserslautern e-mail: {meyer,huechting}@cs.uni-kl.de
[2] Newcastle University e-mail: Victor.Khomenko@ncl.ac.uk

Abstract. We develop a polynomial translation from finite control processes (an important fragment of π-calculus) to safe low-level Petri nets. To our knowledge, this is the first such translation. It is natural (there is a close correspondence between the control flow of the original specification and the resulting Petri net), enjoys a bisimulation result, and is suitable for practical model checking.

Keywords: π-calculus, finite control process, FCP, Petri net, model checking.

1 Introduction

Many contemporary systems enjoy a number of features that significantly increase their power, usability and flexibility:

Dynamic reconfigurability The overall structure of many existing systems is flexible: nodes in ad-hoc networks can dynamically appear or disappear; individual cores in Networks-on-Chip can be temporarily shut down to save power; resilient systems have to continue to deliver (reduced) functionality even if some of their modules develop faults.

Logical mobility Mobile systems permeate our lives and are becoming ever more important. Ad-hoc networks, where devices like mobile phones and laptops form dynamic connections are common nowadays, and the vision of pervasive (ubiquitous) computing, where several devices are simultaneously engaged in interaction with the user and each other, forming dynamic links, is quickly becoming a reality.

Dynamic allocation of resources It is often the case that a system has several instances of the same resource (e.g. network servers or processor cores in a microchip) that have to be dynamically allocated to tasks depending on the current workload, power mode, priorities of the clients, etc.

The common feature of such systems is the possibility to form *dynamic* logical connections between the individual modules. It is implemented using reference passing: a module can become aware of another module by receiving a reference (e.g. in the form of a network address) to it, which enables subsequent communication between these two modules. This can be thought of as a new (logical) channel dynamically created between these modules. We will refer to such systems as Reference Passing Systems (RPS).

As people are increasingly dependent on the correct functionality of such systems, the cost incurred by design errors in them can be extremely high. However, even the conventional concurrent systems are notoriously difficult to design correctly because of the complexity of their behaviour, and reference passing adds another layer of complexity due to the logical structure of the system becoming dynamical. Hence, computer-aided formal verification has to be employed in the design process to ensure the correct behaviour of RPSs.

While the complexity of systems increases, the time-to-market is reducing. To address this, system design has changed from a holistic to a compositional process: The system is usually composed from pre-existing modules. *This change in the design process has to be mirrored by the change of focus of the formal verification from the level of individual modules to the inter-modular level.*

Nowadays it is reasonable to assume that individual modules are already well-tested or formally verified by their vendors. Moreover, the inter-module communication fabric (e.g. a

computer network) is usually built of standard components and uses standard protocols, and so can be assumed to be correct-by-construction. On the other hand, the interaction between the modules is usually highly complicated. Thus, verification of the inter-module communication is required to ensure that *the system as a whole* provides the desired functionality.

These considerations can be addressed by abstracting away the low-level communication infrastructure (e.g. network behaviour) and the internal behaviour of the modules. Only the behaviour on the modules' interfaces is modelled, and the model of the overall system is the composition of these interface models. This view has the advantage of *separating the verification concerns.*

Traditionally, such inter-module verification is accomplished using rely/guarantee reasoning and supported by automated theorem provers. However, due to undecidability reasons, theorem proving cannot be fully automated and requires substantial manual intervention to help the tool to discharge some of the proof obligations. Hence, lifting (fully automatic) model checking to the inter-modular level is highly desirable.

There is a number of formalisms that are suitable for specification of RPSs. The main considerations and tradeoffs in choosing an appropriate formalism are its expressiveness and the tractability of the associated verification techniques. Expressive formalisms (like π-calculus [16] and Ambient Calculus [4]) are Turing powerful and so not decidable in general. Fortunately, the ability to pass references *per se* does not lead to undecidability. One can impose restrictions on dimensions like communication [1, 13], control [5, 20] and interconnection shape [12, 13] to recover decidability while retaining a reasonable modelling power.

Finite Control Processes (FCP) [5] are a fragment of π-calculus, where the system is constructed as a parallel composition of a fixed number of sequential entities (threads). The control of each thread can be represented by a finite automaton, and the number of threads is bounded in advance. The threads communicate synchronously via channels, and have the possibility to create new channels dynamically and to send channels via channels. These capabilities are often sufficient for modelling mobile applications and instances of parameterised systems, and the appeal of FCPs is due to combining this modelling power with decidability of verification problems [5, 14].

In this paper, we contribute to FCP verification, following an established approach: we translate the process into a safe low-level Petri net (PN). This translation bridges the gap between expressiveness and verifiability: While π-calculus is suitable for modelling mobile systems but difficult to verify due to the complicated semantics, PNs are a low-level formalism equipped with efficient analysis algorithms. With the translation, all verification techniques and tools that are available for PNs can be applied to analysing the (translated) process.

Technically, our translation relies on three insights: (i) the behaviour of an FCP $\nu a.(S_1 \mid S_2)$ coincides with the behaviour of $(S_1\{n/a\} \mid S_2\{n/a\})$ where the restricted name a has been replaced by a fresh public name n (a set of fresh names that is linear in the size of the FCP will be sufficient); (ii) we have to recycle fresh names, and so implement reference counters for them; and (iii) we hold substitutions explicit and give them a compact representation using decomposition, e.g., $\{a, b/x, y\}$ into $\{a/x\}$ and $\{b/y\}$.

1.1 Complexity-theoretic considerations

There is a large body of literature on π-calculus to PN translations (cf. Section 1.2 for a detailed discussion). Complexity-theoretic considerations, however, suggest that they are all suboptimal for FCPs — either in terms of size [3, 10, 13, 14] or because of a too powerful target formalism [1, 6, 11].

The following shows that a polynomial translation of FCPs into low-level safe PNs must exist. Indeed, it is well-known that a Turing machine with bounded tape can be modelled by such a PN of polynomial (in the size of the control and the tape's length) size, see e.g. [8]. In turn, as the state of an FCP can always be described by a string of length linear in the process size, an FCP can be simulated by a Turing machine with the tape of linear length (in the

FCP's size). Moreover, there is an easy translation from a safe PN to an FCP of linear (in the PN's size) size.[1] That is, the three formalisms can simulate each other with only polynomial overhead. This argument is in fact constructive and shows the PSPACE-completeness of FCP model checking, but the resulting PN would be ugly.

These considerations motivated us to look for a *natural* polynomial translation of FCPs to safe PNs, which is the main contribution of this paper. We stress that our translation is not just a theoretical result, but is also quite practical:

- it is very natural (there is a strong correspondence between the control flow of the FCP being translated and the resulting PNs);
- the transition systems of the FCP and that of its PN representation are bisimilar, which makes the latter suitable for checking temporal properties of the former;
- the resulting PN is compact (polynomial even in the worst case);
- we propose a number of optimisations allowing to significantly reduce the size of the resulting PN in practice;
- we propose several extensions of the translation, in particular to polyadic π-calculus and match/mismatch operators;
- the conducted experiments demonstrate that the translation is suitable for practical model checking.

1.2 Related work

There are two main approaches to verification of FCPs. The first one is to directly generate the reachable state space of the model, e.g. as done on-the-fly by the Mobility Workbench (MWB) [22]. This approach is relatively straightforward, but it has a number of disadvantages, in particular its scalability is poor due to the complexity of the π-calculus semantics restricting the use of heuristics for pruning the state space and the need to perform expensive operations (like computing the canonical form of the term) every time a new state is generated. Furthermore, some efficient model checking techniques like symbolic representation of the state space are very difficult to apply.

The alternative approach, and the one followed in this paper, is to translate a π-calculus term into a simpler formalism, e.g. Petri nets (PN), that is then analysed. This approach has a number of advantages, in particular it does not depend on a concrete verification technique, and can adapt any such technique for PNs. Furthermore, RPSs often are highly concurrent, and so translating them into a true concurrency formalism like PNs has a number of advantages, in particular one can efficiently utilise partial-order reductions for verification, alleviating thus the problem of combinatorial state space explosion (that is, a small specification often has a huge number of reachable states, which is beyond the capability of existing computers).

Although for the π-calculus several translations have been proposed in literature, none of them provides a polynomial translation of FCPs into safe PNs. We discuss several such translations below.

The verification kit HAL [9] translates a model into a *History Dependent Automaton* — a finite automaton where states are labelled by sets of names that represent restrictions — a formalism proposed by Montanari and Pistore [17, 20]. For model checking, these automata are further translated to finite automata [9]. Like in our approach, the idea is to replace restrictions with fresh names. But their translation stores full substitutions, which may yield an exponential blow up of the finite automaton. The translation presented here avoids this blow up by compactly representing substitutions by PN markings. This, however, needs careful substitution manipulation and reference counting.

To handle restrictions, Amadio and Meyssonnier [1] replace unused names by generic free names. Their translation instantiates the substitution, i.e. a π-calculus process like

[1] The idea is to create a thread for each transition and two defining equations for each place that correspond to the presence and absence of a token in that place. Then communication actions are used to simulate the behaviour of the transitions.

$(\overline{x_1}\langle y_1\rangle.\overline{x_2}\langle y_2\rangle)\{a,b,a,b/x_1,y_1,x_2,y_2\}$ is represented by $\overline{a}\langle b\rangle.\overline{a}\langle b\rangle$. This creates an exponential blow up: since the substitutions that are applied change over time, m public names and n variables may yield m^n instantiated terms. Moreover, since the number of processes to be modified by replacement is not bounded in the paper, these authors use PNs with transfer. Their translation handles a subset of π-calculus that is incomparable with FCPs. Moreover, as the results of this paper show, transfer nets are an unnecessarily powerful target formalism — e.g. reachability is undecidable in such nets [7].

Busi and Gorrieri study non-interleaving and causal semantics for the π-calculus and provide decidability results for model checking [3]. The work fails to prove bisimilarity, which is recovered in [10]. The translations may be exponential for FCPs due to the instantiation of substitutions.

Devillers, Klaudel and Koutny [6] achieve a bisimilar translation of π-calculus into high-level Petri nets, thus using a Turing complete formalism where automatic analyses are necessarily incomplete. The main contribution is compositionality: for every π-calculus operator there is a corresponding net operator, and in many cases the size of the net is linear in the size of the process. However, the target formalism is too powerful and the paper provides no experimental evaluation.

Khomenko, Koutny and Niaouris [11] translate the recursion-free fragment of π-calculus into high-level PNs and verify the latter using an unfolding based technique for high-level PNs. The approach can express only finite runs, and so its practical applicability is limited. Besides, the target formalism is unnecessarily powerful.

The approach in [14] translates FCPs into safe low-level PNs, which are then verified using PN unfoldings. The experiments in that paper indicate that this technique is much more scalable than the ones above, and it has the advantage of generating low-level rather than high-level PNs. However, in the worst case the resulting PN is exponential in the size of the original FCP.

Peschanski, Klaudel and Devillers [19] translate π-graphs (a graphical variant of π-calculus) into high-level PNs. The technique works on a fragment that is equivalent to FCPs. However, the target formalism is unnecessarily powerful, and the paper provides no experimental evaluation.

2 Basic notions

In this section we recall the basic notions concerning Petri nets and FCPs.

2.1 Petri nets

A *Petri net* (PN) is a tuple $N \stackrel{\text{df}}{=} (P, T, F, M_0)$ such that P and T are disjoint sets of *places* and *transitions*, $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*, and M_0 is the *initial marking* of N, where a *marking* $M : P \to \mathbb{N} \stackrel{\text{df}}{=} \{0, 1, 2, \ldots\}$ of N is a multiset of places. We draw PNs in the standard way: places are represented as circles, transitions as boxes, the flow relation by arcs, and a marking by tokens within circles. The *size* of N is $\|N\| \stackrel{\text{df}}{=} |P| + |T| + |F| + |M_0|$.

We denote by ${}^\bullet z \stackrel{\text{df}}{=} \{y \mid (y,z) \in F\}$ and $z^\bullet \stackrel{\text{df}}{=} \{y \mid (z,y) \in F\}$ the *preset* and *postset* of $z \in P \cup T$, respectively. A transition t is *enabled at marking* M, denoted by $M \xrightarrow{t}$, if $M(p) \geq 0$ for every $p \in {}^\bullet t$. Such a transition can be *fired*, leading to the marking M' with $M'(p) \stackrel{\text{df}}{=} M(p) - F(p,t) + F(t,p)$ for every $p \in P$. We denote the firing relation by $M \xrightarrow{t} M'$ or by $M \to M'$ if the identity of the transition is irrelevant. The set of *reachable markings of* N is denoted by $\mathcal{R}(N)$.

A PN N is *k-bounded* if $M(p) \leq k$ for every $M \in \mathcal{R}(N)$ and every place $p \in P$, and *safe* if it is 1-bounded. We will focus on safe PNs.

Several places in a Petri net are called *mutually exclusive* if at each reachable marking at most one of them contains tokens. In a safe Petri net, a place p is a *complement* of a set Q of

mutually exclusive places if at any reachable marking p contains a token iff none of the places in Q contains a token. If $Q = \{q\}$ is a singleton, the places p and q are complements of each other.

2.2 Finite control processes

In π-calculus [15,21], threads communicate via synchronous message exchange. The key idea in the model is that messages and the channels they are sent on have the same type: they are just *names* from some set $\Phi \stackrel{\text{df}}{=} \{a, b, x, y, i, f, r, \ldots\}$. This means a name that has been received as message in one communication may serve as channel in a later interaction. To communicate, processes consume *prefixes* π of the form

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

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

Threads, also called *sequential processes*, are constructed as follows. A *choice process* $\sum_{i\in I} \pi_i.S_i$ over a finite set of indices I executes a prefix π_i and then behaves like S_i. The special case of choices over an empty index set $I = \emptyset$ is denoted by $\mathbf{0}$ — such a process has no behaviour. A *restriction* $\nu r.S$ generates a name r that is different from all other names in the system. We denote a (perhaps empty) sequence of restrictions $\nu r_1 \ldots \nu r_k$ by $\nu\tilde{r}$ with $\tilde{r} = r_1 \ldots r_k$. To implement parameterised recursion, we use *calls to process identifiers* $K\lfloor\tilde{a}\rfloor$. We defer the explanation of this construct for a moment. To sum up, threads take the form

$$S ::= K\lfloor\tilde{a}\rfloor \;\mid\; \textstyle\sum_{i\in I} \pi_i.S_i \;\mid\; \nu r.S.$$

We use $\mathcal{S}$ to refer to the set of all threads. A *finite control process (FCP)* F is a parallel composition of a fixed number of threads:

$$F ::= \nu\tilde{a}.(S_{init,1} \mid \ldots \mid S_{init,n}).$$

Our presentation of parameterised recursion using calls $K\lfloor\tilde{a}\rfloor$ follows [21]. Process identifiers K are taken from some set $\Psi \stackrel{\text{df}}{=} \{H, K, L, \ldots\}$ and have a *defining equation* $K(\tilde{f}) := S$. Thread S can be understood as the implementation of identifier K. The process has a list of *formal parameters* $\tilde{f} = f_1, \ldots, f_k$ that are replaced by *factual parameters* $\tilde{a} = a_1, \ldots, a_k$ when a call $K\lfloor\tilde{a}\rfloor$ is executed. Note that both lists $\tilde{a}$ and $\tilde{f}$ have the same length. When we talk about an *FCP specification* F, we mean process F with all its defining equations.

To implement the replacement of $\tilde{f}$ by $\tilde{a}$ in calls to process identifiers, we use *substitutions.* A substitution is a function $\sigma : \Phi \to \Phi$ that maps names to names. If we make domain and codomain explicit, $\sigma : A \to B$ with $A, B \subseteq \Phi$, we require $\sigma(a) \in B$ for all $a \in A$ and $\sigma(x) = x$ for all $x \in \Phi \setminus A$. We use $\{\tilde{a}/\tilde{f}\}$ to denote the substitution $\sigma : \tilde{f} \to \tilde{a}$ with $\sigma(f_i) \stackrel{\text{df}}{=} a_i$ for $i \in \{1, \ldots, k\}$. The *application of substitution* σ *to* S is denoted by $S\sigma$ and defined in the standard way [21].

Input prefix $a(i)$ and restriction νr *bind* the names i and r, respectively. The *set of bound names* in a process $P = S$ or $P = F$ 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 and collectively referred to as *no clash* **(NC)** henceforth. For every FCP specification F, we require that:

- a name is bound at most once;
- a name is used at most once in formal parameter lists;
- the sets of bound names, free names and formal parameters are pairwise disjoint;
- if a substitution $\sigma = \{\tilde{a}/\tilde{x}\}$ is applied to S then $bn(S)$ and $\tilde{a} \cup \tilde{x}$ are disjoint.

Assuming **(NC)**, the names occurring in an FCP specification F can be partitioned into the following sets:

$\mathcal{P}$ public names that are free in F;
$\mathcal{R}$ names bound by restriction operators;
$\mathcal{I}$ names bound by input prefixes;
$\mathcal{F}$ names used as formal parameters in defining equations.

We are interested in the relation between the size of an FCP specification and the size of its Petri net representation. The *size* of an FCP specification is defined as the size of its initial term plus the sizes of the defining equations. The corresponding function $\|\cdot\|$ measures the number of channel names, process identifiers, the lengths of parameter lists, and the number of operators in use:

$$\|\mathbf{0}\| \stackrel{\text{df}}{=} 1 \qquad \|K\lfloor\tilde{a}\rfloor\| \stackrel{\text{df}}{=} 1 + |\tilde{a}|$$
$$\|\textstyle\sum_{i\in I} \pi_i.S_i\| \stackrel{\text{df}}{=} 3|I| - 1 + \sum_{i\in I} \|S_i\| \qquad \|S_{init,1} \mid \ldots \mid S_{init,n}\| \stackrel{\text{df}}{=} n - 1 + \textstyle\sum_{i=1}^{n} \|S_{init,i}\|$$
$$\|\nu r.P\| \stackrel{\text{df}}{=} 1 + \|P\| \qquad \|K(\tilde{f}) := S\| \stackrel{\text{df}}{=} 1 + |\tilde{f}| + \|S\|$$

To define the behaviour of a process, we rely on *structural congruence* $\equiv$. It is the smallest congruence where α-conversion of bound names is allowed, $+$ and $\mid$ are commutative and associative with $\mathbf{0}$ as the neutral element, and the following laws for restriction hold:

$$\nu x.\mathbf{0} \equiv \mathbf{0} \qquad \nu x.\nu y.P \equiv \nu y.\nu x.P \qquad \nu x.(P \mid Q) \equiv P \mid (\nu x.Q) \text{ if } x \notin fn(P).$$

The behaviour of π-calculus processes is determined by the *reaction relation* $\rightarrow$ [15, 21]:

$$\text{(Tau) } \tau.S + M \rightarrow S \qquad \text{(React) } (x(y).S + M) \mid (\overline{x}\langle z\rangle.S' + N) \rightarrow S\{z/y\} \mid S'$$

$$\text{(Res) } \frac{P \rightarrow P'}{\nu a.P \rightarrow \nu a.P'} \qquad \text{(Struct) } \frac{P \rightarrow P'}{Q \rightarrow Q'} \quad \text{if } P \equiv Q \text{ and } P' \equiv Q'$$

$$\text{(Par) } \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q} \qquad \text{(Const) } K\lfloor\tilde{a}\rfloor \rightarrow S\{\tilde{a}/\tilde{f}\} \quad \text{if } K(\tilde{f}) := S$$

The rule (Tau) is an axiom for silent steps. (React) describes the communication of two parallel threads, consuming their send and receive actions respectively and continuing as a process, where the name y is substituted by z in the receiving thread S. (Const) describes identifier calls, likewise using a substitution. The remaining rules define $\rightarrow$ to be closed under structural congruence, parallel composition and restriction. By $\mathcal{R}(F)$ we denote the set of all processes reachable from F. The *transition system* of FCP F factorises the reachable processes along structural congruence, $\mathcal{T}(F) \stackrel{\text{df}}{=} (\mathcal{R}(F)/_{\equiv}, \hookrightarrow, \underline{F})$ where $\underline{F_1} \hookrightarrow \underline{F_2}$ if $F_1 \rightarrow F_2$.

Normal form assumptions To ease the definition of the Petri net translation and the corresponding correctness proofs, we make assumptions about the shape of the FCP specification. These assumptions are not restrictive, as any FCP can be translated into the required form.

We require that the sets of identifiers called (both directly from F and indirectly from defining equations) by different threads are disjoint. This restriction corresponds to the notion of a *safe* FCP [14] and can be achieved by replicating some defining equations. The resulting specification F' is bisimilar with F and has size $O(n\|F\|) = O(\|F\|^2)$. We illustrate the construction on the following example of an FCP specification F (left) together with its replicated version F' (right):

$$K(f_1, f_2) := \tau.L(f_1, f_2) \qquad K^1(f_1^1, f_2^1) := \tau.L^1(f_1^1, f_2^1)$$
$$L(f_3, f_4) := \tau.K(f_3, f_4) \qquad L^1(f_3^1, f_4^1) := \tau.K^1(f_3^1, f_4^1)$$
$$K^2(f_1^2, f_2^2) := \tau.L^2(f_1^2, f_2^2)$$
$$L^2(f_3^2, f_4^2) := \tau.K^2(f_3^2, f_4^2)$$
$$K^3(f_1^3, f_2^3) := \tau.L^3(f_1^3, f_2^3)$$
$$L^3(f_3^3, f_4^3) := \tau.K^3(f_3^3, f_4^3)$$
$$K\lfloor a, b\rfloor \mid K\lfloor b, c\rfloor \mid L\lfloor a, c\rfloor \qquad K^1\lfloor a, b\rfloor \mid K^2\lfloor b, c\rfloor \mid L^3\lfloor a, c\rfloor$$

Intuitively, in the resulting FCP specification each thread has its own set of defining equations. In our translation, this requirement ensures that the control flow nets of different threads are disjoint.

We also can ensure that defining equations do not call themselves, i.e. that the body of $K(\tilde{f}) := S$ does not contain any calls of the form $K\lfloor\tilde{a}\rfloor$. Indeed, we can replace any such call with $K'\lfloor\tilde{a}\rfloor$, with a new defining equation $K'(\tilde{f'}) := K(\tilde{f'})$. This increases the size of the FCP only linearly, and ensures we do not have to re-map parts of $\tilde{f}$ to $\tilde{f}$ when passing the parameters of a call, which simplifies the translation of such calls.

3 Translation of π-calculus to safe Petri nets

In this section we informally explain the proposed polynomial translation of FCPs to safe Petri nets.

The first step is to model the substitution $\sigma : \mathcal{R} \cup \mathcal{I} \cup \mathcal{F} \to \mathcal{P} \cup \mathcal{N}$ (mapping the bound names and formal parameters occurring in the FCP and active at the current state to their values) as a set of PN places. This step is detailed in Sect. 3.1.

The second step is to translate the control of each thread into a Petri net. This is done in a straightforward way, in particular:

- The communication prefixes are modelled by stubs at this point (i.e. no synchronisation between the threads is performed yet).
- The translation of the overall FCP is the net obtained by placing the translations of its threads side-by-side (recall that the threads in an FCP never share any defining equations).
- Each subterm t of a thread is translated into a subnet with a unique entry place p_t (different occurrences of the same subterm are distinguished and yield different subnets). This place is initially marked if t correspond to some thread's initial expression in the main term of the FCP.

The translation of the control is performed as follows, depending on the structure of t (note that each thread is a sequential process, so $|$ cannot occur in them, and that the prefix operator is a special case of the sum):

Stop process 0 The corresponding subnet is comprised of the entry place, optionally followed by a subnet unmapping in the substitution all the bound names and formal parameters in whose scope this **0** process reside.

Call $K\lfloor\tilde{a}\rfloor$ Let $K(\tilde{f}) := S$ be the defining equation for K. The entry place is followed by a subnet that amends the substitution in such a way that $\tilde{f}$ become mapped to the values of $\tilde{a}$ and all the other bound names and formal parameters in whose scope this call resides become unmapped; then the control is transferred to the entry place of the translation of S.

Restriction If the term has the form $\nu r.S$ then the corresponding entry place is followed by a subnet that amends the substitution so that r becomes mapped to some $n \in \mathcal{N}$ to which no other name is currently mapped, which in turn is followed by the translation of S.

Sum We assume that the sums are guarded, i.e. have the form $\sum_{i=1}^{n} \pi_i.S_i$. The entry place is connected to the entry places of the translations of S_is by transitions labelled π_i. If $\pi_i \neq \tau$ (i.e. the prefix is a communication action) then the corresponding transition is a stub, i.e. it will be eliminated at the end of the translation, after the synchronisation of the nets corresponding to the threads is completed.

The third step of the translation is to synchronise the subnets corresponding to different threads on communication actions. Let t and t'' be two stub transitions labelled by actions $\overline{a}\langle b\rangle$ and $x(y)$, respectively, belonging to different FCP threads. They are synchronised by adding a set of transitions implementing the communication. At most one of these transitions can be enabled (depending on the values of a, b and x), and each of these transitions consumes tokens

from the input places of t and t'', produces tokens at their output places, checks by read arcs the appropriate places of the substitution to verify that the values of a and x are the same (and so the communication is possible), and amends the substitution by mapping y to the value of b. If the communication is impossible, none of these transitions is enabled. After all synchronisations are performed, the stub transitions are removed from the net.

We now explain each element of the translation in more detail. Below, we use the notation $dom(x)$ to denote the *domain* of each name, i.e. any fixed overapproximation of the set of possible values of a name x. The domains of all names in the proposed translation are finite, and can be computed by static analysis (see Sect. 6); however, even the following rough overapproximation is sufficient to prove the polynomiality of the translation:

$$dom(x) \stackrel{\text{df}}{=} \begin{cases} \{x\} & \text{if } x \in \mathcal{P} \\ \mathcal{N} & \text{if } x \in \mathcal{R} \\ \mathcal{P} \cup \mathcal{N} & \text{if } x \in \mathcal{I} \cup \mathcal{F} \end{cases}$$

3.1 Petri net representation of the name substitution

The substitution $\sigma : \mathcal{R} \cup \mathcal{I} \cup \mathcal{F} \to \mathcal{P} \cup \mathcal{N}$ maps the bound names and formal parameters occurring in the FCP and active at the current state to their values.

The efficient Petri net representation of this substitution is the key element of the proposed translation. In particular, the following operations must be efficiently supported:

Initialisation of a restricted name It should be possible to find a value $val \in \mathcal{N}$ to which no bound name or formal parameter is currently mapped, and map a given restricted name r to val.

Remapping A given input name i may be mapped to $\sigma(v)$, where v is the communicated name in the corresponding communication action; alternatively, a given formal parameter f can be mapped to $\sigma(v)$ by a process call that uses v as a factual parameter. In the latter case, it is also possible that v occurs in the list of factual parameters several times, so it is convenient to be able to map several formal parameters to $\sigma(v)$ in one step (i.e. by one PN transition).

Unmapping When a bound name or formal parameters b goes out of scope, its mapping should be removed. During a process call, it often happens that $\sigma(b)$ is assigned to one or more formal parameters, and simultaneously b goes out of scope, so it is convenient to be able remap and unmap b in one step.

Independence The three kinds of operations described above should not interfere with each other when applied to different names (note that due to **(NC)**, the bound names and formal parameters in distinct threads are always different), so that they can be performed concurrently, without the need for synchronisation. This prevents the introduction of arbitration, and so has beneficial effect on the clarity of the translation and the performance of some model checking methods (e.g. unfoldings and those using partial order reductions).

In what follows, we describe a representation of σ as a safe PN that satisfies all the formulated requirements. The PN places modelling σ are shown in Fig. 1. The places $[var = val]$, when marked, represent the fact that $var \in \mathcal{R} \cup \mathcal{I} \cup \mathcal{F}$ is mapped to $val \in \mathcal{P} \cup \mathcal{N}$. Places named $[var \neq val]$, where $var \in \mathcal{R} \cup \mathcal{I} \cup \mathcal{F}$ and $val \in \mathcal{N}$, are compliments of the corresponding places $[var = val]$, and places $[r_* \neq val]$, where $val \in \mathcal{N}$, are complements of the corresponding sets of places $\{[r_1 = val], \ldots, [r_{n_r} = val]\}$ (note that the places in these sets are mutually exclusive).

The following important invariants are maintained by all the PN transitions:

- for each $var \in \mathcal{I} \cup \mathcal{F}$ and $val \in \mathcal{N}$, $[var \neq val]$ is complementary to $[var = val]$;
- for each $val \in \mathcal{N}$, the places $[r_1 = val], \ldots, [r_{n_r} = val]$ are mutually exclusive (i.e. no two restricted names can be mapped to the same value), and $[r_* \neq val]$ is complementary to these places;

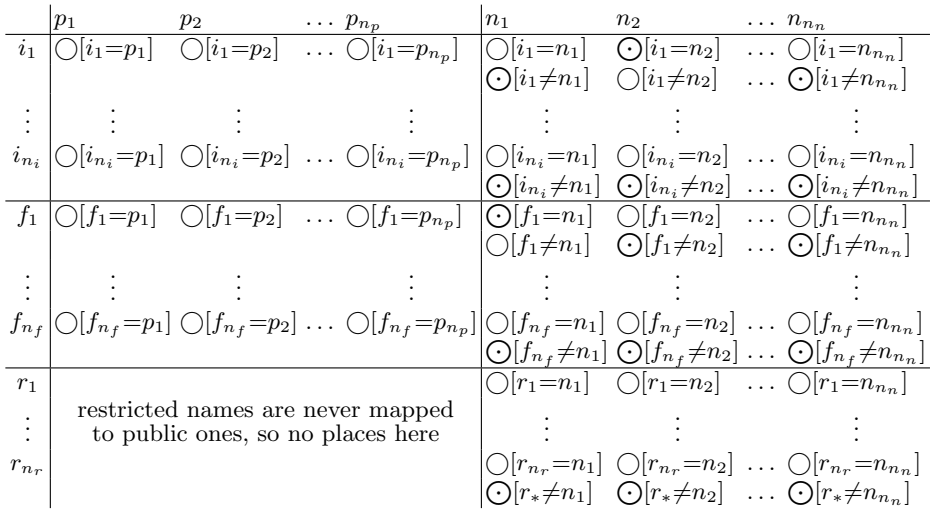

Fig. 1. Illustration of N_{Subst} with a substitution marking that corresponds to $\sigma : \{i_1, f_1\} \to \tilde{a} \cup \mathcal{P}$ where $\sigma(i_1) = a_1$ and $\sigma(f_1) = a_2$ with $a_1 \neq a_2$. The marking represents a_1 by n_2 and a_2 by n_1.

– for each $var \in \mathcal{R} \cup \mathcal{I} \cup \mathcal{F}$, the places $[var = val]$, where val runs through $dom(var)$, are mutually exclusive (i.e. a name can be mapped to at most one value).

The choice of the cardinality of the set $\mathcal{N}$ (n_n in Fig. 1) is of crucial importance: it should be sufficiently big to guarantee that there will always be a name that can be used to initialise a restricted name when necessary, but taking an unnecessary big value for this parameter increases the size of the generated PN as well as the number of its reachable states. The following rough overapproximation is sufficient to prove the polynomiality of the translation:

$$|\mathcal{N}| \stackrel{\text{df}}{=} |\mathcal{R}| + |\mathcal{I}| + |\mathcal{F}|.$$

The rationale is that there should be enough values in $\mathcal{N}$ to assign a unique value to each bound variable and formal parameter. A better overapproximation can be obtained using static analysis, see Sect. 6.

The operations on the substitution are implemented as follows.

Initialisation of a restricted name To find a value $val \in \mathcal{N}$ to which no bound name or formal parameter is currently mapped, and map a given restricted name r_k to val, the PN transition performing the initialisation has to:

– test by read arcs that the places $[i_1 \neq val], \ldots, [i_{n_i} \neq val]$ and $[f_1 \neq val], \ldots, [f_{n_f} \neq val]$ have tokens (i.e. no input or formal parameter name is currently mapped to val);
– consume the token from $[r_* \neq val]$ (checking thus that val is not currently assigned to any restricted name);
– produce a token at $[r_k = val]$ (performing thus the initialisation of r_k with the value val).

Remapping Whenever the value of a name v becomes bound to some input name i_k by a communication action, or to a formal parameter f_k by a process call, the corresponding transition has to consume the token from $[i_k \neq \sigma(v)]$ (resp. $[f_k \neq \sigma(v)]$) and produce a token in $[i_k = \sigma(v)]$ (resp. $[f_k = \sigma(v)]$). Note that in general it is possible that v occurs several times in the list of factual parameters of the call, and so several formal parameters $f_{k_1}, \ldots, f_{k_l}$ have to be bound to $\sigma(v)$. This situation can still be handled by a single PN transition consuming the tokens from $[f_{k_*} \neq \sigma(v)]$ and producing tokens in $[f_{k_*} = \sigma(v)]$; furthermore, this can be combined with unmapping v (see below), and still be done by a single PN transition.

Unmapping When a bound name or formal parameter *var* mapped to *val* goes out of scope, its mapping should be removed. This can be modelled by a PN transition consuming the token from $[var = val]$ and, if $val \in \mathcal{N}$, producing a token in $[var \neq val]$.

3.2 Initialisation of restricted names

Suppose the process has the form $\nu r.S$. The corresponding subnet is constructed as follows, cf. Fig. 2.

For each $n \in \mathcal{N}$, we create a transition t_n^r consuming a token from the entry place, producing a token in the entry place of the subnet implementing S, and performing the initialisation of the restricted name r with the value n as explained in Sect. 3.1.

Note that the transitions t_n^r arbitrate between the names in $\mathcal{N}$, allowing any of the currently unused names to be selected for the initialisation of r. If such an arbitration is undesirable,[2] separate pools of values can be used for each thread, as described Sect. 6.

3.3 Handling 0 process

Executing a **0** process terminates a thread. One option for implementing it would be to unmap all the bound names and formal parameters in whose scope this **0** resides, and then stop. However, one can observe that this unmapping is not necessary, as the used resources (in particular, the names from $\mathcal{N}$ to which these bound variables are mapped) will not be needed. Hence, the subnet implementing a **0** process can consist of its entry place only.

3.4 Handling calls

At the point of a call, all the active bound names and formal parameters of the process go out of scope (recall that we assume that the body of a definition of a K cannot call K). Hence, the substitution has to be appropriately amended.

Let $K\lfloor\tilde{a}\rfloor$ be a call with the factual parameters $\tilde{a}$, B be the set of bound names and formal parameters within whose scope this call is performed, and $K(\tilde{f}) := S$ be the equation defining K. Due to **(NC)**, all the names in $\tilde{f}$ are different (and so $\tilde{f}$ will be treated as a set below), whereas names in $\tilde{a}$ can be repeated; moreover, $\tilde{f}$ does not overlap with $\tilde{a}$ and B. The call is then performed by:

- Modifying the substitution so that the names in $\tilde{f}$ are mapped to the corresponding values of $\tilde{a}$, and the names in B become unmapped.
- Transferring the control to the entry place of the subnet implementing S.

The latter is trivial, and we now describe the former in more detail. Let A be the set of names occurring in $\tilde{a}$ (perhaps, multiple times). Then the required change in the substitution can be modelled by the assignments

$$X_i \leftarrow a, \text{ for each } a \in A \qquad \text{and} \qquad \emptyset \leftarrow a, \text{ for each } a \in B \setminus A,$$

where X_i is the set of formal parameters to which the value of the factual parameter a is assigned by the call, i.e. X_i's are disjoint non-empty sets whose union is $\tilde{f}$. Intuitively, an assignment $X \leftarrow a$, where $X \subseteq \tilde{f}$ can be empty, simultaneously maps all the variables in X to $\sigma(a)$ and, if $a \in B$, unmaps a. Since no two assignments reference the same name, they cannot interfere and thus can be executed in any order or concurrently.

The subnet implementing an assignment $X \leftarrow a$ has one entry and one exit place and is constructed as follows. For each $val \in dom(a)$ we create a transition t_{val} which:

[2] E.g. due to its negative impact on some of the model checking techniques. Note however that using symmetries mitigates the negative impact of this arbitration on model checking, as all the states that can be reached after performing this arbitration become equivalent.

- consumes a token from the entry place and produces a token on the exit place;
- for each $f \in X$, consumes a token from $[f \neq val]$ (provided this place exists, i.e. $val \in \mathcal{N}$) and produces a token on $[f = val]$;
- if $a \in B$, consumes a token from the place $[a = val]$, and, in case $val \in \mathcal{N}$, produces a token on the place $[a \neq val]$ (or in the place $[r_* \neq val]$ if a is a restricted name).

Such subnets can be combined in either sequential or parallel manner (in the later case additional fork and join transitions are needed).

3.5 Handling communication

Recall that, given stub transitions t' and t'' labelled by actions $\overline{a}\langle b\rangle$ and $x(y)$ belonging to different FCP threads, the synchronisation adds a set of transitions implementing the communication. We now show how these transitions are constructed, cf. Fig. 3.

If static analysis (see Sect. 6) shows that $\overline{a}\langle b\rangle$ and $x(y)$ are potentially synchronisable, we create for each $i \in dom(a) \cap dom(x)$ and $j \in dom(b) \cap dom(y)$ a transition t_{ij} which:

- consumes tokens from the input places of the stubs t' and t'' and produces tokens on their output places;
- checks by read arcs that $[a = i]$ and $[x = i]$ are marked (i.e. the substitution maps a and x to the same value i and thus the synchronisation is possible);
- checks by a read arc that $[b = j]$ is marked, consumes a token from $[y \neq j]$ (if this place exists) and produces a token on $[y = j]$ (mapping thus in the substitution y to j, i.e. to the value of b).

If the synchronisation is possible in the current state of the system, exactly one of these transitions is enabled (depending on the values of a, b and x); else none of these transitions is enabled.

After all such synchronisations are performed, the stub transitions are removed from the net.

3.6 Size of the translation

The size of the PN generated by our translation is dominated by the number of transitions modelling communication — in fact, they determine the degree of the polynomial giving the asymptotic worst-case size of the PN. In the worst case, the numbers of sending and receiving actions are $O(\|F\|)$ and almost all pairs of send/receive actions can synchronise; thus the total number of such synchronisations is $O(\|F\|^2)$. Recall that for a pair of actions $\overline{x_1}\langle y_1\rangle$ and $x_2(y_2)$, a separate transition is generated for each pair of names in $\mathcal{P} \cup \mathcal{N}$. In the worst case $|\mathcal{P} \cup \mathcal{N}| = O(\|F\|)$, and thus the total number of transitions implementing communication, as well as the size of the resulting PN, are $O(\|F\|^4)$. However, the 'communication splitting' optimisation described in Sect. 6 reduces this size down to $O(\|F\|^3)$.

4 Definition of the translation

In this section we formally describe the proposed translation. To do that, we add some further assumptions on the form of the FCP. These assumptions are not restrictive: any FCP can be transformed into an FCP of the required form. However, the assumptions significantly simplify the correctness proofs by reducing the number of cases that have to be considered.

4.1 Additional normal form assumptions

We augment the **(NC)** assumptions as follows: in a defining equation $K(\tilde{f}) := S$, $fn(S) = \tilde{f}$. That is, we do not allow public names in defining equations. Note that this assumption can easily be enforced by passing all the required public names as parameters.

Note that our translation interprets the names in $\mathcal{I} \cup \mathcal{F} \cup \mathcal{R}$ as variables that substitutions $\sigma : \mathcal{I} \cup \mathcal{F} \cup \mathcal{R} \to \mathcal{P} \cup \mathcal{N}$ assign constants to. Here, $\mathcal{P}$ is the finite set of public names and $\mathcal{N}$ is a set of fresh names which are assigned to restricted names. The following assumption helps us avoid case distinctions for the initial process. We assume there are artificial defining equations $K_{Init,i}(\tilde{f}_{Init,i}) := S_{Init,i}$, with $fn(S_{Init,i}) = \tilde{f}_{Init,i} \subseteq \mathcal{F}$, that are called by a virtual initialisation step. Their purpose is to guarantee that the processes $S_{Init,i}$ have the free names $\tilde{f}_{Init,i}$. We then apply substitutions to assign the expected values to these parameters. This means we can write the given FCP as

$$F = \nu\tilde{a}.(S_{Init,1}\sigma_1 \mid \ldots \mid S_{Init,n}\sigma_n),$$

where each substitution σ_i maps $\tilde{f}_{Init,i}$ into $\tilde{a}$ and $\mathcal{P}$, $\sigma_i : \tilde{f}_{Init,i} \to \tilde{a} \cup \mathcal{P}$. We additionally assume that the $S_{Init,i}$ are choices or calls.

Finally, if we have an input $x(y).S$ then we assume $y \in fn(S)$, which can be achieved by adding an artificial parameter to the call at the end of the process. Similarly, for a restriction $\nu r.S$ we assume $r \in fn(S)$. Restrictions not satisfying this requirement can be dropped using structural congruence.

4.2 Outline of the translation

Recall that the main idea is to replace restricted names by fresh public ones. Indeed, the behaviour of

$$F = \nu\tilde{a}.(S_{Init,1}\sigma_1 \mid \ldots \mid S_{Init,n}\sigma_n)$$

coincides with that of

$$S_{Init,1}\sigma'_1 \mid \ldots \mid S_{Init,n}\sigma'_n$$

with $\sigma'_i \stackrel{\text{df}}{=} \sigma_i\{\tilde{n}/\tilde{a}\}$, provided the names $\tilde{n}$ are fresh. These new names are picked from a set $\mathcal{N}$, and since for an FCP specification there is a bound on the number of restricted names in all processes reachable from F, a finite $\mathcal{N}$ suffices. But how to support name creation and deletion with a constant number of free names? The trick is to reuse the names: $n \in \mathcal{N}$ may first represent a restricted name r_1 and later a different restricted name r_2. To implement this recycling of names, we keep track of whether or not $n \in \mathcal{N}$ is currently used in the process. This can be understood as reference counting.

The translation takes the finite set of names $\mathcal{N}$ as a parameter. The resulting PN is a composition

$$N(F) \stackrel{\text{df}}{=} N_{Subst} \triangleleft H(N(S_{Init,1}) \parallel \ldots \parallel N(S_{Init,n})).$$

Each PN $N(S_{Init,i})$ is a finite automaton (its transitions have one incoming and one outgoing arc and the initial marking has one token) that reflects the control flow of thread $S_{Init,i}$. It operates on a low-level of abstraction and explicitly handles the introduction and removal of name bindings. The transitions of $N(S_{Init,i})$ are annotated with synchronisation actions and sets of commands. Transitions with complementary synchronisation actions are appropriately merged by the parallel composition $\parallel$. Hiding H then removes the original transitions. Commands are handled by the *implementation operator* $\triangleleft$, which connects the control flow to N_{Subst} — a net that compactly represents the substitutions in a process and implements reference counting.

4.3 Construction of N_{Subst}

The key idea of our representation of a substitution is to partition it into elementary substitutions, e.g. $\{a, b/x, y\}$ is represented as $\{a/x\} \cup \{b/y\}$. The substitution net has corresponding places $[x{=}a]$ and $[y{=}b]$ for each component that may occur in such a decomposition. Moreover, there is a second set of places $[x{\neq}n]$ and $[r_*{\neq}n]$. They are needed to keep track of whether an input, a formal parameter, or a restriction is bound to $n \in \mathcal{N}$. Note that these places complement the corresponding substitution places, in particular $[r_*{\neq}n]$ indicates that no restricted name is bound to n. (Since at most one restriction can be bound to n, this one complement place is sufficient.) N_{Subst} has no transitions and hence no arcs, and we defer the explanation of its initial marking. Formally, $N_{Subst} \stackrel{\text{df}}{=} (P_{Subst} \cup P_{Ref}, \emptyset, \emptyset, M_0)$, where

$$P_{Subst} \stackrel{\text{df}}{=} ((\mathcal{I} \cup \mathcal{F}) \times \{=\} \times \mathcal{P}) \ \cup \ ((\mathcal{I} \cup \mathcal{F} \cup \mathcal{R}) \times \{=\} \times \mathcal{N})$$
$$P_{Ref} \stackrel{\text{df}}{=} (\mathcal{I} \cup \mathcal{F} \cup \{r_*\}) \times \{\neq\} \times \mathcal{N}.$$

Substitution Markings and Correspondence A marking M of N_{Subst} is called a *substitution marking* if it satisfies the following constraints:

$$\textbf{(SM1)} \quad M([r_*{\neq}n]) + \sum_{r \in \mathcal{R}} M([r{=}n]) = 1 \qquad \sum_{a \in \mathcal{P} \cup \mathcal{N}} M([x{=}a]) \leq 1 \quad \textbf{(SM2)}$$
$$M([x{=}n]) = 1 \quad \text{iff} \quad M([x{\neq}n]) = 0. \quad \textbf{(SM3)}$$

The first equation holds for every fresh name $n \in \mathcal{N}$. It states that at most one restricted name is bound to n. We find a token on $[r_*{\neq}n]$ iff there is no such binding. The second constraint states that every name $x \in \mathcal{I} \cup \mathcal{F} \cup \mathcal{R}$ is bound to at most one $a \in \mathcal{P} \cup \mathcal{N}$. The reference counter has to keep track of whether a name $x \in \mathcal{I} \cup \mathcal{F}$ maps to a fresh name $n \in \mathcal{N}$, which motivates the third equivalence.

Consider now a substitution $\sigma : (\mathcal{I}' \cup \mathcal{F}' \to \mathcal{P} \cup \tilde{a}) \cup (\mathcal{R}' \to \tilde{a})$ with domain $\mathcal{I}' \subseteq \mathcal{I}$, $\mathcal{F}' \subseteq \mathcal{F}$, $\mathcal{R}' \subseteq \mathcal{R}$, codomain $\mathcal{P}$ and some set of names $\tilde{a}$, and where the second component $\mathcal{R} \to \tilde{a}$ is injective. A substitution marking M of N_{Subst} is said to *correspond to* σ if the following hold:

(COR1) For all $x \in \mathcal{I} \cup \mathcal{F} \cup \mathcal{R} \setminus dom(\sigma)$ and $a \in \mathcal{N} \cup \mathcal{P}$, $M([x{=}a]) = 0$.
(COR2) For all $x \in dom(\sigma)$ with $\sigma(x) \in \mathcal{P}$, $M([x{=}\sigma(x)]) = 1$.
(COR3) For all $x \in dom(\sigma)$ with $\sigma(x) \in \tilde{a}$, there is $n \in \mathcal{N}$ s.t. $M([x{=}n]) = 1$.
(COR4) The choice of n preserves the equality of names as required by σ, i.e. for all $x, y \in dom(\sigma)$ with $\sigma(x), \sigma(y) \in \tilde{a}$ and all $n \in \mathcal{N}$, we have

$$\sigma(x) = \sigma(y) \quad \text{iff} \quad M([x{=}n]) = M([y{=}n]).$$

Recall that we translate the specification $F = \nu\tilde{a}.(S_{Init,1}\sigma_1 \mid \ldots \mid S_{Init,n}\sigma_n)$. As *initial marking* of N_{Subst}, we fix some substitution marking that corresponds to $\sigma_1 \cup \ldots \cup \sigma_n$. As we shall see, every choice of fresh names $\tilde{n}$ for $\tilde{a}$ indeed yields bisimilar behaviour. Note that **(NC)** ensures that the union of substitutions is again a function. Fig. 1 illustrates N_{Subst} and the concepts of substitution markings and correspondence.

4.4 Construction of $N(S_{Init})$

Petri net $N(S_{Init})$ reflects the control flow of thread S_{Init}. To synchronise send and receive prefixes in different threads, we annotate its transitions with labels from $\mathcal{L} \stackrel{\text{df}}{=} \{\tau, send(a, b), rec(a, b) \mid a, b \in \mathcal{P} \cup \mathcal{N}\}$. To capture the effect that reactions have on substitutions, transitions also carry a set of commands from

$$\mathcal{C} \stackrel{\text{df}}{=} \{map(x, b), unmap(x, b), test([x = b]) \mid x \in \mathcal{I} \cup \mathcal{F} \cup \mathcal{R} \text{ and } b \in \mathcal{P} \cup \mathcal{N}\}.$$

These commands maintain the name binding in the overall net. Formally, a *control flow net* is a tuple (P, T, F, M_0, l, c) where (P, T, F, M_0) is a Petri net and $l : T \to \mathcal{L}$ and $c : T \to \mathbb{P}(\mathcal{C})$ are the labellings.

As S_{Init} is a sequential process, transitions in $N(S_{Init})$ will always have a single input and a single output place. This allows us to understand $N(S_{Init})$ as a finite automaton, and hence define it implicitly via a new labelled transition system for S_{Init}. Recall that $\mathcal{S}$ is the set of sequential processes. We extend them by sequences of names: $\mathcal{S} \times (\mathcal{I} \cup \mathcal{F} \cup \mathcal{R})^*$. These lists will carry the names that have been forgotten and should be eventually unmapped in N_{Subst}. Among such extended processes, we then define the labelled transition relation

$$\twoheadrightarrow \subseteq (\mathcal{S} \times (\mathcal{I} \cup \mathcal{F} \cup \mathcal{R})^*) \times \mathcal{L} \times \mathbb{P}(\mathcal{C}) \times (\mathcal{S} \times (\mathcal{I} \cup \mathcal{F} \cup \mathcal{R})^*).$$

Each transition carries a label and a set of commands, and will yield a Petri net transition. We have the following transitions among extended processes.

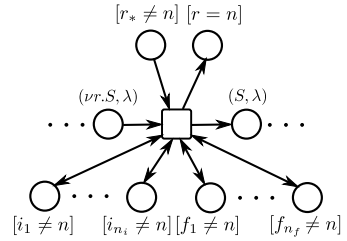

Fig. 2. Translation of a restriction with $map(r, n)$ implemented.

For restrictions $\nu r.S$, we allocate a fresh name. Since we can select any name that is not in use, such a transition exists for every $n \in \mathcal{N}$:

$$(\nu r.S, \lambda) \xrightarrow[\{map(r,n)\}]{\tau} (S, \lambda). \qquad (TRANS_\nu)$$

Fig. 2 depicts the transition, together with the implementation of mapping defined below.

Silent actions yield a τ-labelled transition with empty set of commands as expected:

$$(\tau.S + \textstyle\sum_{i \in I} \pi_i.S_i, \lambda) \xrightarrow[\emptyset]{\tau} (S, \lambda \cdot \lambda'), \qquad (TRANS_\tau)$$

where $\lambda' = fn\left(\tau.S + \sum_{i \in I} \pi_i.S_i\right) \setminus fn\,(S)$. This means λ' contains the names that were free in the choice process but have been forgotten in S. With an ordering on $\mathcal{P} \cup \mathcal{N}$, we can understand this set as a sequence.

Communications are more subtle. Consider $\overline{x}\langle y\rangle.S + \sum_{i \in I} \pi_i.S_i$ that sends y on channel x. Via appropriate tests, we find the names a and b that x and y are bound to. These names then determine the transition label. So for all $a, b \in \mathcal{P} \cup \mathcal{N}$, we have

$$(\overline{x}\langle y\rangle.S + \textstyle\sum_{i \in I} \pi_i.S_i, \lambda) \xrightarrow[\{test([x=a]), test([y=b])\}]{send(a,b)} (S, \lambda \cdot \lambda'). \qquad (TRANS_{snd})$$

Sequence λ' again contains the names that are no longer in use. A receive action in $x(y).S + \sum_{i \in I} \pi_i.S_i$ is handled like a send, but introduces a new binding. For all $a, b \in \mathcal{P} \cup \mathcal{N}$, we get

$$(x(y).S + \textstyle\sum_{i \in I} \pi_i.S_i, \lambda) \xrightarrow[\{test([x=a]), map(y,b)\}]{rec(a,b)} (S, \lambda \cdot \lambda'). \qquad (TRANS_{rec})$$

There are similar transitions for the remaining prefixes π_i with $i \in I$. Fig. 3(left) illustrates the transitions for send and receive actions.

For a call to an identifier $K\lfloor x_1, \ldots, x_n \rfloor$ with $K(f_1, \ldots, f_n) := S$, the idea is to iteratively update the substitution, by binding the formal parameters to the factual ones and then unmapping the names in λ (which will include the factual parameters). Note that $fn\,(S) = \{f_1, \ldots, f_n\}$ by **(NC)**, and due to the assumptions about the form of the process stated in Sect. 2, no equation calls itself, which ensures that we do not accidentally unmap a public name or the just mapped formal parameters. The following transitions are created for each $a \in \mathcal{P} \cup \mathcal{N}$:

$$(K\lfloor x_1, \ldots, x_m \rfloor, \lambda) \xrightarrow[\{test([x_m=a]), map(f_m,a)\}]{\tau} (K\lfloor x_1, \ldots, x_{m-1} \rfloor, \lambda'), \qquad (TRANS_{call_1})$$

where $\lambda' \stackrel{\text{df}}{=} \lambda$ if $x_m \in \lambda$ and $\lambda' \stackrel{\text{df}}{=} \lambda \cdot x_m$ otherwise. (This case distinction ensures that we will unmap a factual parameter precisely once, even if it occurs multiple times in the list of

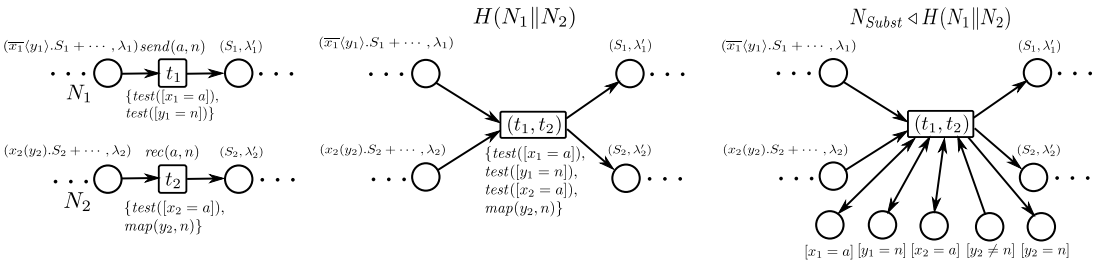

Fig. 3. Translation of communication (left), parallel composition and hiding (center), and implementation of $test([x_1{=}a])$ and $map(y_2, n)$ (right).

factual parameters.) When all parameters have been passed, we unmap the names in $\lambda \neq \varepsilon$, by creating the following transitions for each $a \in \mathcal{P} \cup \mathcal{N}$:

$$(K\lfloor - \rfloor, x \cdot \lambda) \xrightarrow[\{unmap(x,a)\}]{\tau} (K\lfloor - \rfloor, \lambda). \qquad (TRANS_{call_2})$$

When $\lambda = \varepsilon$ has been reached, we transfer the control to the body S of the defining equation:

$$(K\lfloor - \rfloor, \varepsilon) \xrightarrow[\emptyset]{\tau} (S, \varepsilon). \qquad (TRANS_{call_3})$$

Petri net $N(S_{Init})$ is the restriction of $(\mathcal{S} \times (\mathcal{I} \cup \mathcal{F} \cup \mathcal{R})^*, \twoheadrightarrow)$ to the extended processes that are reachable from (S_{Init}, ε) via $\twoheadrightarrow$. The initial marking puts one token on place (S_{Init}, ε) and leaves the remaining places unmarked.

4.5 Operations on nets

Parallel composition $\|$ Parallel composition of labelled nets is classical in Petri net theory. The variant we use is inspired by [2]. The parallel composition $N_1 \parallel N_2$ forms the disjoint union of N_1 and N_2, and then synchronises the transitions t_1 in N_1 that are labelled by $l_1(t_1) = send(a, b)$ (resp. $rec(a, b)$) with the transitions t_2 in N_2 that are labelled by $l_2(t_2) = rec(a, b)$ (resp. $send(a, b)$). The result is a new transition (t_1, t_2) without a label, which carries the union of the commands for t_1 and t_2. Note that a labelled transition that has been used for synchronisation in $N_1 \parallel N_2$ is still available for further synchronisations with N_3. This in particular implies that $\|$ is associative and commutative.

Consider the Petri nets $N_i = (P_i, T_i, F_i, M_{0,i}, l_i, c_i)$ with $i \in \{1, 2\}$, where P_1 and P_2 as well as T_1 and T_2 are disjoint. (Note that the disjointness of these sets follows from the disjointness of sets of process identifiers for different threads.) Their parallel composition is defined as

$$N_1 \parallel N_2 \stackrel{\text{df}}{=} (P_1 \cup P_2, T_1 \cup T_2 \cup T, F_1 \cup F_2 \cup F, M_{0,1} \cup M_{0,2}, l_1 \cup l_2 \cup l, c_1 \cup c_2 \cup c).$$

The set of new transitions is

$$\begin{aligned} T \stackrel{\text{df}}{=} \{(t_1, t_2) \in T_1 \times T_2 \mid\ & l_1(t_1) = rec(a, b) \text{ and } l_2(t_2) = send(a, b), \text{ or} \\ & l_1(t_1) = send(a, b) \text{ and } l_2(t_2) = rec(a, b)\}. \end{aligned}$$

The labellings are $l((t_1, t_2)) \stackrel{\text{df}}{=} \tau$ and $c((t_1, t_2)) \stackrel{\text{df}}{=} c(t_1) \cup c(t_2)$. The flow relation is inherited from the original nets: For each $p_i \in P_i$, $i \in \{1, 2\}$, we have:

$$F(p_i, (t_1, t_2)) \text{ iff } F_i(p_i, t_i) \qquad F((t_1, t_2), p_i) \text{ iff } F_i(t_i, p_i).$$

Hiding H The *hiding operator* removes from a labelled PN N all transitions t with $l(t) \neq \tau$. Consider the labelled Petri net $N = (P, T, F, M_0, l, c)$. We define $H(N) \stackrel{\text{df}}{=} (P, T', F', M_0, l', c')$,

where $T' \stackrel{\text{df}}{=} \{t \in T \mid l(t) = \tau\}$ and the flow relation and the labellings are restricted accordingly: $F' \stackrel{\text{df}}{=} F|_{T'}$, $l' \stackrel{\text{df}}{=} l|_{T'}$, $c' \stackrel{\text{df}}{=} c|_{T'}$. Since $H(N)$ contains only τ-labelled transitions, we can omit the labelling l' from the result. The combination of parallel composition and hiding is illustrated in Fig. 3(center).

Implementation operation $\triangleleft$ Consider the two Petri nets $N_1 = N_{Subst} = (P_1, \emptyset, \emptyset, M_{0,1})$ and $N_2 = H(N(S_{Init,1}) \parallel \ldots \parallel N(S_{Init,n})) = (P_2, T, F_2, M_{0,2}, c)$ defined above. The implementation operation

$$N_1 \triangleleft N_2 \stackrel{\text{df}}{=} (P_1 \cup P_2, T, F_2 \cup F, M_{0,1} \cup M_{0,2})$$

yields a standard Petri net without labelling. Its purpose is to implement the commands carried by the transitions of N_2 by adding arcs between the two nets. We fix a transition $t \in T$ and a command $c \in c(t)$, and define the arcs that have to be added between t and some places of N_1 to implement c. We do the case analysis for the possible types of c:

test$([x{=}b])$ We add a loop to place $[x{=}b]$: $([x{=}b], t), (t, [x{=}b]) \in F$.

map(x, p), *map*(x, n), *map*(r, n) A map command differentiates according to whether the first component is an input or a formal parameter $x \in \mathcal{I} \cup \mathcal{F}$, or whether it is a restricted name $r \in \mathcal{R}$. If x is assigned a public name, $map(x, p) \in c(t)$ with $p \in \mathcal{P}$, we just add a token to the substitution net, $(t, [x{=}p]) \in F$. If x is assigned some $n \in \mathcal{N}$, $map(x, n) \in c(t)$, we additionally remove the token from the reference counter: $(t, [x{=}n]), ([x{\neq}n], t) \in F$. To represent restricted name $r \in \mathcal{R}$ by a name $n \in \mathcal{N}$, we first check that no other name is currently mapped to n using the reference counter for n. In case n is currently unused, we introduce the binding $[r{=}n]$ to the substitution net: $([r_*{\neq}n], t), (t, [r{=}n]) \in F$ and $\{([x{\neq}n], t), (t, [x{\neq}n]) \mid x \in \mathcal{I} \cup \mathcal{F}\} \subseteq F$.

unmap(x, p), *unmap*(x, n), *unmap*(r, n) An unmap command removes the binding of $x \in \mathcal{I} \cup \mathcal{F}$: $([x{=}p/n], t) \in F$; moreover, in case of $n \in \mathcal{N}$, it updates the reference counter: $(t, [x{\neq}n]) \in F$. When we remove the binding of $r \in \mathcal{R}$ to $n \in \mathcal{N}$, we update $[r_*{\neq}n]$ in the reference counter: $([r{=}n], t), (t, [r_*{\neq}n]) \in F$.

Fig. 2 illustrates the implementation of mapping for a restriction, $map(r, n)$. Tests and mapping of an input name are shown in Fig. 3(right).

5 Correctness of the translation

To show the correctness of the proposed translation we relate F and $N(F)$ by a suitable form of bisimulation. The problem is that $N(F)$ may perform several steps to mimic one transition of F. The reason is that changes to substitutions (as induced e.g. by $\nu r.S$) are handled by transitions in $N(F)$ whereas F uses structural congruence; i.e. a substitution change does not necessarily lead to a step in the reaction relation of F. To obtain a clean relationship between the models, we restrict the transition system of $N(F)$ to so-called *stable markings* and *race free transition sequences* between them. Intuitively, stable markings correspond to the choices and process calls in F, and race free transition sequences mimic the reaction steps between them. We show below that this restriction is insignificant, as any transition sequence is equivalent to some race free one.

Marking M of $N(F) = N_{Subst} \triangleleft H(N(S_{Init,1}) \parallel \ldots \parallel N(S_{Init,n}))$ is called *stable* if, in every control flow net $N(S_{Init,i})$, it marks a place (S, λ) where S either is a choice or a call to a process identifier with full parameter list. We denote by $\mathcal{R}_{Stbl}(N(F))$ the set of stable markings that are reachable in $N(F)$. We furthermore refer to places (S, λ) where S either is a choice or a call to a process identifier as *stable places*.

A transition sequence $t_1, \ldots, t_n$ between stable markings $M, M' \in \mathcal{R}_{Stbl}(N(F))$ is *race free* if exactly one t_i is either of the form $(TRANS_\tau)$ for a silent action, of the form (t, t') for communication actions $(TRANS_{snd})$, $(TRANS_{rec})$, or of the form $(TRANS_{call_2})$ for an identifier call, cf. Sect. 4.4. Thus, a race free transition sequence corresponds to precisely one

step in the reaction relation of F, characterised by t_i, while the other transitions t_j implement the substitution changes between M, M'. In particular, no intermediary marking is stable.

We denote the fact that there is such a race free transition sequence by $M \Rightarrow M'$. The *stable transition system of* $N(F)$ is now

$$\mathcal{T}_{Stbl}(N(F)) \stackrel{\text{df}}{=} (\mathcal{R}_{Stbl}(N(F)), \Rightarrow, M_0).$$

Here, M_0 is the initial marking of $N(F)$. By the assumption on $S_{Init,i}$ from Sect. 4.1, the marking is stable. Using $\mathcal{T}_{Stbl}(N(F))$, we can now formulate our main theorem for the correctness of our translation:

Theorem 1. *The transition system of F and the stable transition system of $N(F)$ are bisimilar, $\mathcal{T}(F) \sim \mathcal{T}_{Stbl}(N(F))$, via the bisimulation $\mathcal{B}$ defined below.*

To define the bisimulation relation, we use the fact that every process reachable from F is structurally congruent to

$$\nu\tilde{a}.(S_1\sigma_1 \mid \ldots \mid S_n\sigma_n).$$

Here, S_i is a choice or an identifier call that has been derived from S with $K(\tilde{f}) := S$. Derived means $(S, \varepsilon) \twoheadrightarrow^+ (S_i, \lambda_i)$ so that no intermediary process is a call to a process identifier. As second requirement, we have

$$\sigma_i : fn(S_i) \cup \lambda_i \to \tilde{a} \cup \mathcal{P}. \quad \textbf{(DOM)}$$

This means the domain of σ_i is the free names in S_i together with the names λ_i that have already been forgotten. The two sets are disjoint, $fn(S_i) \cap \lambda_i = \emptyset$. The above process actually is in standard form [15], but makes additional assumptions about the shape of threads and the domain of substitutions.

We define $\mathcal{B} \subseteq \mathcal{R}(F)/_{\equiv} \times \mathcal{R}_{Stbl}(N(F))$ to contain $(\underline{G}, M_1 \cup M_2) \in \mathcal{B}$ if there is a process $\nu\tilde{a}.(S_1\sigma_1 \mid \ldots \mid S_n\sigma_n) \equiv G$ as above so that the following holds: Marking M_1 of N_{Subst} corresponds to $\sigma_1 \cup \ldots \cup \sigma_n$, and for the control flow marking, we have $M_2(S_i, \lambda_i) = 1$ for all $i \in \{1, \ldots, n\}$.

5.1 Bisimulation Proof

We now turn to the bisimulation proof. We have to show that for each pair $(\underline{G}, M) \in \mathcal{B}$, every transition $\underline{G} \hookrightarrow \underline{G'}$ can be mimicked by a race free transition sequence in $N(F)$, i.e. there is a stable marking M' with $M \Rightarrow M'$ such that $(\underline{G'}, M') \in \mathcal{B}$. Moreover and in turn, the race free transition sequences in $N(F)$ should be imitated in process F. The proof is split into two parts, formulated as Lemmas 1 and 2, for both directions respectively.

Lemma 1. *Let $(\underline{G}, M) \in \mathcal{B}$. For all $\underline{G'}$ with $\underline{G} \hookrightarrow \underline{G'}$ there is a stable marking $M' \in \mathcal{R}_{Stbl}(N(F))$ such that $M \Rightarrow M'$ and $(\underline{G'}, M') \in \mathcal{B}$.*

Proof. Process G is structurally congruent to $\nu\tilde{a}.(S_1\sigma_1 \mid \ldots \mid S_n\sigma_n)$. By the base cases of the reaction rules, transition $\underline{G} \hookrightarrow \underline{G'}$ exists iff (1) either two processes $S_i\sigma_i$ and $S_j\sigma_j$ with $i \neq j \in \{1, \ldots, n\}$ communicate, (2) we resolve a call to a process identifier in some $S_i\sigma_i$, $i \in \{1, \ldots, n\}$, or (3) we have a τ action. Silent steps are easier than the former two and hence omitted in the proof.

Case 1: Communication For simplicity, we assume that: the first two threads communicate using the first prefixes; after the communication, the first thread yields choice or call S_1'; the second process creates precisely one restricted name before becoming a choice or a call S_2'; the communication is over restricted names and a restricted name is sent. The remaining cases are along similar lines. We thus have $G \equiv \nu\tilde{a}.(S_1\sigma_1 \mid \ldots \mid S_n\sigma_n)$ with

$$\begin{aligned} S_1 &= \overline{x_1}\langle y_1\rangle.S_1' + M_1 & \sigma_1(x_1) &= \sigma_2(x_2) \in \tilde{a} \\ S_2 &= x_2(y_2).\nu r.S_2' + M_2 & \sigma_1(y_1) &\in \tilde{a}. \end{aligned}$$

The process resulting from the communication is

$$G' \stackrel{\text{df}}{=} \nu\tilde{a}.a_r.(S_1'\sigma_1 \mid S_2'\sigma_2' \mid S_3\sigma_3 \mid \ldots \mid S_n\sigma_n) \quad \text{with} \quad \sigma_2' \stackrel{\text{df}}{=} \sigma_2\{\sigma_1(y_1)/y_2\}\{a_r/r\}.$$

We argue that G' has the desired normal form. The processes S_1' and S_2' are choices or calls. Moreover, $(S, \varepsilon) \twoheadrightarrow^* (S_2, \lambda_2)$ implies $(S, \varepsilon) \twoheadrightarrow^* (S_2', \lambda_2 \cdot \lambda_2')$. This means S_1' and S_2' have been derived as required. It remains to show **(DOM)**. We do the proof for σ_2', the reasoning for σ_1 is simpler:

$$\begin{aligned}
& dom(\sigma_2') && (1)\\
&= dom(\sigma_2) \cup \{y_2, r\} && (2)\\
&= \lambda_2 \cup fn\,(S_2) \cup \{y_2, r\} && (3)\\
&= \lambda_2 \cup (fn\,(S_2) \setminus fn\,(\nu r.S_2')) \cup (fn\,(\nu r.S_2') \setminus \{y_2\}) \cup \{y_2, r\} && (4)\\
&= \lambda_2 \cup (fn\,(S_2) \setminus fn\,(\nu r.S_2')) \cup fn\,(S_2') \cup \{y_2, r\} && (5)\\
&= \lambda_2 \cup (fn\,(S_2) \setminus fn\,(\nu r.S_2')) \cup fn\,(S_2') && (6)\\
&= \lambda_2 \cdot \lambda_2' \cup fn\,(S_2')\,. && (7)
\end{aligned}$$

Equation (3) is **(DOM)** for σ_2. Equation (4) uses the fact that

$$fn\,(S_2) = (fn\,(S_2) \setminus fn\,(\nu r.S_2')) \cup (fn\,(\nu r.S_2') \setminus \{y_2\}).$$

This is due to $fn\,(\nu r.S_2') \setminus \{y_2\} \subseteq fn(S_2)$. Equation (5) is due to

$$(fn\,(\nu r.S_2') \setminus \{y_2\}) \cup \{y_2, r\} = fn\,(S_2') \cup \{y_2, r\}.$$

Equation (6) holds by $\{y_2, r\} \subseteq fn(S_2')$. Finally, Equation (7) holds by definition of the extended transition relation $\twoheadrightarrow$.

We now argue that (1.a) there is $M' \in \mathcal{R}_{Stbl}(N(F))$ so that $M \Rightarrow M'$ and (1.b) $(\underline{G'}, M') \in \mathcal{B}$.
Claim 1.a: There is $M' \in \mathcal{R}_{Stbl}(N(F))$ with $M \Rightarrow M'$ We decompose $M = M_1 \cup M_2$ so that M_1 is the substitution marking and M_2 is the control flow marking. Since $(\underline{G}, M_1 \cup M_2) \in \mathcal{B}$, we have $M_2((S_1, \lambda_1)) = 1 = M_2((S_2, \lambda_2))$. Moreover, M_1 corresponds to $\sigma_1 \cup \ldots \cup \sigma_n$. In the following, we also use σ to refer to this union. Since $\sigma_1(x_1), \sigma_2(x_2), \sigma_1(y_1) \in \tilde{a}$, by **(COR3)** we have fresh names $n_1, n_2, n_3 \in \mathcal{N}$ with $M_1([x_1{=}n_1]) = 1 = M_1([x_2{=}n_2]) = M_1([y_1{=}n_3])$. Since $\sigma_1(x_1) = \sigma_2(x_2)$, we conclude $n_1 = n_2$ by **(COR4)**.

It remains to argue that there is a fresh name available in $\mathcal{N}$ to represent r. As $r \notin dom(\sigma)$, we have

$$|dom(\sigma)| < |\mathcal{I}| + |\mathcal{F}| + |\mathcal{R}| = |\mathcal{N}|.$$

With **(COR1)**, for $x \notin dom(\sigma)$ we have $M_1([x{=}n]) = 0$ for all $n \in \mathcal{N}$. For $x \in dom(\sigma)$, we have at most one place $[x{=}a]$ marked by **(SM2)**. Together, this means there is a name $n \in \mathcal{N}$ with $M_1([x{=}n]) = 0$ for all $x \in \mathcal{I} \cup \mathcal{F} \cup \mathcal{R}$. Let this name be n_r. Since we have $M_1([x{=}n_r]) = 0$ for $x \in \mathcal{I} \cup \mathcal{F} \cup \mathcal{R}$, constraints **(SM1)** and **(SM3)** ensure $M_1([x{\neq}n_r]) = 1$ for $x \in \mathcal{I} \cup \mathcal{F} \cup \{r_*\}$.

Before parallel composition, the original net $N(S_{Init,1})$ had the following transition sequence leaving place (S_1, λ_1):

$$(S_1, \lambda_1) \xrightarrow[\{test([x_1{=}n_1]), test([y_1{=}n_3])\}]{send(n_1, n_3)} (S_1', \lambda_1').$$

Similarly, from (S_2, λ_2) in $N(S_{Init,2})$ we have

$$(S_2, \lambda_2) \xrightarrow[\{test([x_2{=}n_1]), map([y_2{=}n_3],)\})]{rec(n_1, n_3)} (\nu r.S_2', \lambda_2 \cdot \lambda_2') \xrightarrow[\{map(r, n_r)\}]{\tau} (S_2', \lambda_2 \cdot \lambda_2').$$

Parallel composition joins the communicating transitions of the two nets, and we denote the result by (t_1, t_2). Then hiding removes the original transitions t_1 labelled by $send(n_1, n_3)$ and t_2 labelled by $rec(n_1, n_3)$. Then, for (t_1, t_2) and for the transition t_r mapping r to a fresh name, the implementation operation adds arcs to and from N_{Subst}.

We now show that the transition sequence $(t_1, t_2)\, t_r$ is enabled, starting with (t_1, t_2). We argued that (S_i, λ_i) carries a token. This means the control flow is at the right place. We have $M_1([x_1{=}n_1]) = 1 = M_1([x_2{=}n_1]) = M_1([y_1{=}n_3])$. Hence, the test arcs to the substitution net are enabled. We have $y_2 \in bn(S_2)$. Hence, the name is not in the domain of σ_2 by **(DOM)** and **(NC)**. With **(COR1)**, $M([y_2{=}a]) = 0$ holds for all names $a \in \mathcal{P} \cup \mathcal{N}$. In particular, $M([y_2{=}n_3]) = 0$. With **(SM3)**, we conclude $M([y_2{\neq}n_3]) = 1$. This ensures $map(y_2, n_3)$ is enabled. For t_r, we have $M([x{\neq}n_r]) = 1$ for all $x \in \mathcal{I} \cup \mathcal{F} \cup \{r_*\}$. Hence, the transition is enabled.

The resulting marking M' puts tokens on (S_1', λ_1') and $(S_2', \lambda_2 \cdot \lambda_2')$ which are stable places. This means M' is stable. The marking is reachable as M was reachable. Moreover, transition sequence $(t_1, t_2) \cdot t_r$ above is race free.

Claim 1.b: $(\underline{G'}, M') \in \mathcal{B}$ Again $M' = M_1' \cup M_2'$ where M_1' is the marking of N_{Subst} and M_2' is the control flow. For the control flow, we moved the single token from (S_1, λ_1) to (S_1', λ_1') and from (S_2, λ_2) to $(S_2', \lambda_2 \cdot \lambda_2')$ as required.

For N_{Subst}, we show that we obtain a substitution marking. We already argued that $M_1([y_2{=}a]) = 0$ for all $a \in \mathcal{P} \cup \mathcal{N}$ and hence $M_1([y_2{\neq}n_3]) = 1$. We consume the latter token and move it to $M_1'([y_2{=}n_3]) = 1$. This means we still map y_2 to at most one name as required by **(SM2)**. Moreover, the invariant on reference counting **(SM3)** is satisfied.

Name r is not in the domain of σ_2. Hence, the places $[r{=}a]$ are empty for all $a \in \mathcal{N} \cup \mathcal{P}$. We move the token from $M_1([r_*{\neq}n_r]) = 1$ to $M_1'([r{=}n_r]) = 1$. As a result, the places $[r{=}a]$ for all $a \in \mathcal{N} \cup \mathcal{P}$ together carry at most one token as required by **(SM2)**. Moreover, the places $[r{=}n_r]$ for all $r \in \mathcal{R}$ plus $[r_*{\neq}n_r]$ carry precisely one token. This proves **(SM1)**. We have a substitution marking.

We have to show that M_1' corresponds to $\sigma' \stackrel{\text{df}}{=} \sigma_1 \cup \sigma_2' \cup \sigma_3 \cup \ldots \cup \sigma_n$. We only introduce bindings for y_2 and r. For y_2 we have $\sigma_2'(y_2) = \sigma_1(y_1) \in \tilde{a}$. Hence, it is correct that we map $M_1'([y_2{=}n_3]) = 1$ with $n_3 \in \mathcal{N}$. The reasoning is similar for r with $\sigma_2'(r) = a_r$. **(COR3)** holds. Marking M_1' only introduce tokens to the places $[y_2{=}n_3]$ and $[r{=}n_r]$ with $\{y_2, r\} \subseteq dom(\sigma')$. For the remaining names $x \in \mathcal{I} \cup \mathcal{F} \cup \mathcal{R} \setminus \{y_2, r\}$, it coincides with M_1. Note that for $x \notin dom(\sigma')$ we have $x \notin dom(\sigma)$. Hence, by **(COR1)** for M_1, we get $M_1'([x{=}a]) = M_1([x{=}a]) = 0$ for all $a \in \mathcal{P} \cup \mathcal{N}$. This proves **(COR1)** for M_1'.

It remains to show **(COR4)**: the equality required by σ' coincides with the choice of fresh names. For r we have $M_1'([r{=}n_r]) = 1$ and $M_1'([x{=}n_r]) = 0$ for all other names $r \neq x \in \mathcal{I} \cup \mathcal{F} \cup \mathcal{R}$. This coincides with the requirement that $\sigma'(r) \neq \sigma'(x)$. For y_2, we only consider $x \notin \{y_2, r\}$ and get

$$\begin{aligned} \sigma'(y_2) = \sigma'(x) \quad &\text{iff} \quad \sigma(y_1) = \sigma(x) \\ &\text{iff } M_1([y_1{=}n]) = M_1([x{=}n]) \text{ for all } n \in \mathcal{N} \\ &\text{iff } M_1'([y_1{=}n]) = M_1'([x{=}n]) \text{ for all } n \in \mathcal{N} \\ &\text{iff } M_1'([y_2{=}n]) = M_1'([x{=}n]) \text{ for all } n \in \mathcal{N}. \end{aligned}$$

The first equivalence holds by $\sigma'(y_2) = \sigma(y_1)$. The second equivalence is **(COR4)** for σ, the third is the observation that M_1 and M_1' coincide on all names except y_2 and r. The last equivalence is the fact that the rows for y_1 and y_2 coincide. This is by the fact that every name y has at most one place $[y{=}a]$ marked **(SM2)**, in combination with $M_1'([y_1{=}n_3]) = 1 = M_1'([y_2{=}n_3])$.

Case 2: Identifier calls We have $G \equiv \nu\tilde{a}.(K\lfloor\tilde{x}\rfloor\sigma_1 \mid \ldots \mid S_n\sigma_n)$ with $K(\tilde{f}) := S$. We assume S already is a choice or a call. The process resulting from the call $K\lfloor\tilde{x}\rfloor\sigma_1$ is

$$G' \stackrel{\text{df}}{=} \nu\tilde{a}.(S\sigma_1' \mid \ldots \mid S_n\sigma_n) \quad \text{with} \quad \sigma_1' \stackrel{\text{df}}{=} \{\sigma_1(\tilde{x})/\tilde{f}\}.$$

We argue that G' has the desired normal form. The process S is a choice or a call. It has been derived trivially as it is the defining process. For **(DOM)**, we have as desired

$$dom(\sigma'_1) = \tilde{f} = fn\,(S) = fn\,(S) \cup \emptyset.$$

We now argue that (2.a) there is $M' \in \mathcal{R}_{Stbl}(N(F))$ so that $M \Rightarrow M'$ and (2.b) $(\underline{G'}, M') \in \mathcal{B}$.
Claim 2.a: There is $M' \in \mathcal{R}_{Stbl}(N(F))$ **with** $M \Rightarrow M'$ We decompose $M = M_1 \cup M_2$ so that M_1 is the substitution marking and M_2 is the control flow marking. Since $(\underline{G}, M_1 \cup M_2) \in \mathcal{B}$, we know that M_1 corresponds to substitution $\sigma \stackrel{\text{df}}{=} \sigma_1 \cup \ldots \cup \sigma_n$. For M_2, we have $M_2(K\lfloor \tilde{x} \rfloor, \lambda) = 1$. Moreover, by **(DOM)**, we have $\tilde{x} \cup \lambda = dom(\sigma_1)$. Hence, for every name $x_i \in \tilde{x} \cup \lambda$ we have a name $a_i \in \mathcal{P} \cup \mathcal{N}$ so that $M_1([x_i{=}a_i]) = 1$ by **(COR2)** and **(COR3)**. Since an equation does not call itself and since all formal parameters are unique by **(NC)**, we have $\tilde{f} \cap dom(\sigma) = \emptyset$ by **(DOM)**. This means $M_1([f{=}a]) = 0$ for all $f \in \tilde{f}$ and all $a \in \mathcal{N} \cup \mathcal{P}$. With **(SM3)**, we get $M_1([f{\neq}n]) = 1$ for all $f \in \tilde{f}$ and all $n \in \mathcal{N}$.

By definition, Petri net $N(S_{Init,1})$ has the following transition sequence:

$$(K\lfloor \tilde{x} \rfloor, \lambda) \xrightarrow[\{test([x_i=a_i]), map(f_i,a_i)\}]{\tau}{}^{+} (K\lfloor - \rfloor, \lambda') \xrightarrow[\{unmap(x_i,a_i)\}]{\tau}{}^{+} (K\lfloor - \rfloor, \varepsilon) \xrightarrow[\emptyset]{\tau} (S, \varepsilon).$$

The first transition sequence introduces the bindings for $\tilde{f}$ and moves the names in $\tilde{x}$ to λ. The result is $(K\lfloor - \rfloor, \lambda')$ with $\lambda' = \lambda \cdot \tilde{x}'$, where $\tilde{x}'$ is obtained from $\tilde{x}$ by removing the duplicates. The next transition sequence unmaps all names in λ'. Finally, the token is moved to (S, ε).

We now show that the composed sequence is enabled. For the first sequence, the tests are enabled with $M_1([x_i{=}a_i]) = 1$. For formal parameters, mapping $map(f, p)$ with $p \in \mathcal{P}$ is always enabled, and $map(f, n)$ with $n \in \mathcal{N}$ requires $M_1([f{\neq}n]) = 1$. This holds by the above argumentation. The second transition sequence removes the tokens from $[x_i{=}a_i]$. Since we do not repeat names in $\tilde{x}'$ and since $\tilde{x} \cap \lambda = \emptyset$, all transitions are enabled. For $a_i = n \in \mathcal{N}$, unmapping introduces a token to $[x_i{\neq}n]$ or to $[r_*{\neq}n]$.

The resulting marking M' puts tokens on (S, ε), which is a stable place. This means M' is stable. The marking is reachable as M was reachable. Moreover, the transition sequence above is race free.
Claim 2.b: $(\underline{G'}, M') \in \mathcal{B}$ Again we have $M' = M'_1 \cup M'_2$ where M'_1 is the marking of N_{Subst} and M'_2 is the control flow marking. For the control flow, we moved the single token from $(K\lfloor \tilde{x} \rfloor, \lambda)$ to (S, ε) as required.

For N_{Subst}, we show that we obtain a substitution marking. We already argued that $M_1([f{=}a]) = 0$ for all $a \in \mathcal{P} \cup \mathcal{N}$ and hence $M_1([f{\neq}n]) = 1$ for all $n \in \mathcal{N}$. We introduce a token $M'_1([f_i{=}a_i]) = 1$, potentially consuming the complement marking if $a_i = n \in \mathcal{N}$. This means **(SM2)** holds: every name is bound at most once. The second transition sequence manipulates the places for $\tilde{x}' \cup \lambda$. These names are disjoint from $\tilde{f}$ due to $\tilde{f} \cap dom(\sigma_1) = \emptyset$ explained above. We remove all tokens $M_1([x_i{=}a_i]) = 1$ with $x_i \in \tilde{x}' \cup \lambda$. The implementation of unmap ensures we reinstall complement markings. More precisely, if $x_i = r \in \mathcal{R}$ and $a_i = n$, we mark $M'_1([r_*{\neq}n]) = 1$. Since by **(SM1)**, name r was the only restriction bound to n, the constraint continues to hold with $[r_*{\neq}n]$ marked. If $M_1([x_i{=}n]) = 1$ with $x_i \in \mathcal{I} \cup \mathcal{F}$, we get $M'_1([x_i{\neq}n]) = 1$. Hence, **(SM3)** continues to hold. We have a substitution marking.

We have to show that M'_1 corresponds to $\sigma' \stackrel{\text{df}}{=} \sigma'_1 \cup \sigma_2 \cup \ldots \cup \sigma_n$. We focus on σ'_1 and assume $\sigma'_1(f_i) \in \tilde{a}$. This means $\sigma_1(x_i) \in \tilde{a}$ for the corresponding name $x_i \in \tilde{x}$. Since M_1 corresponds to σ, by **(COR3)** for M_1 we have $M_1([x_i{=}n]) = 1$ for a name $n \in \mathcal{N}$. By **(SM2)**, x_i is bound to only one name. This means n has to be the name a_i, $n = a_i$, that we chose for the transition. As a result, we have $M'_1([f{=}n]) = 1$ with $n \in \mathcal{N}$ as required. For $\sigma'_1(f) \in \mathcal{P}$, the reasoning is similar. For the names in $\mathcal{I} \cup \mathcal{F} \cup \mathcal{R} \setminus (dom(\sigma_1) \cup \tilde{f})$, markings M_1 and M'_1 coincide. Hence, if $x \notin dom(\sigma')$ we either have $x \notin dom(\sigma)$ or we have $x \in dom(\sigma_1)$. In the former case, we get $M'_1([x{=}a]) = M_1([x{=}a]) = 0$ for all $a \in \mathcal{P} \cup \mathcal{N}$ by **(COR1)** for σ. In the latter case, the name has been explicitly unmapped by the second transition sequence. Hence, **(COR1)** holds for σ'.

It remains to show **(COR4)**: the equality required by σ' coincides with the choice of fresh names. Consider $f_i, f_j \in \tilde{f}$:

$$\begin{aligned}\sigma'(f_i) = \sigma'(f_j) \quad &\text{iff} \quad \sigma(x_i) = \sigma(x_j)\\ &\text{iff } M_1([x_i{=}n]) = M_1([x_j{=}n]) \text{ for all } n \in \mathcal{N}\\ &\text{iff } M_1'([f_i{=}n]) = M_1'([f_j{=}n]) \text{ for all } n \in \mathcal{N}.\end{aligned}$$

The first equivalence holds by $\sigma_1'(f_i) = \sigma_1(x_i)$ and similar for f_j. The second is **(COR4)** for σ. The third equivalence is the fact that the rows for x_i in M_1 and for f_i in M_1' coincide. This is by the fact that x_i and f_i mark at most one place $[x_i{=}a_i]$ and $[f_i{=}a_i]$ by **(SM2)**, and by the fact that this name a_i coincides. The reasoning for $\sigma'(f) = \sigma'(x)$ with $x \in dom(\sigma_2 \cup \ldots \cup \sigma_n)$ is similar. □

We now turn to the reverse direction and argue that $\underline{G}$ can imitate race free transition sequences enabled by M.

Lemma 2. *Let $(\underline{G}, M) \in \mathcal{B}$. For all $M' \in \mathcal{R}_{Stbl}(N(F))$ so that $M \Rightarrow M'$ there is a process $\underline{G'}$ with $\underline{G} \hookrightarrow \underline{G'}$ and $(\underline{G'}, M') \in \mathcal{B}$.*

Proof. A race free transition sequence $M \Rightarrow M'$ corresponds to a communication among two processes (1), to an identifier call (2), or to a silent action (3). We only consider the first case, the remaining two are along similar lines.

Case 1: Communication We reconstruct the race free transition sequence $M \Rightarrow M'$ to derive information about the shape of M and M'. Since we model a communication, we have $M(S_1, \lambda_1) = 1$ in the net $N(S_{Init,1})$ with $S_1 = \overline{x_1}\langle y_1\rangle.S_1' + \Sigma_{i\in I_1}\pi_{i,1}.S_{i,1}$. Similarly, $M(S_2, \lambda_2) = 1$ in $N(S_{Init,2})$ with $S_2 = x_2(y_2).\nu r.S_2' + \Sigma_{i\in I_2}\pi_{i,2}.S_{i,2}$. Here, S_1' and S_2' are meant to be choices or identifier calls. Thus, again the first two processes communicate and the second generates a fresh name. The race free transition sequence $M \to^+ M'$ in $N(F)$ is now $(t_1, t_2)\, t_r$ where

$$\begin{aligned}t_1 = (S_1, \lambda_1) &\xrightarrow[\{test([x_1{=}n_1]), test([y_1{=}n_2])\}]{send(n_1,n_2)} (S_1', \lambda_1')\\ t_2 = (S_2, \lambda_2) &\xrightarrow[\{test([x_2{=}n_1]), map([y_2{=}n_2],\})]{rec(n_1,n_2)} (\nu r.S_2', \lambda_2 \cdot \lambda_2')\\ t_r = (\nu r.S_2', \lambda_2 \cdot \lambda_2') &\xrightarrow[\{map(r,n_r)\}]{\tau} (S_2', \lambda_2 \cdot \lambda_2').\end{aligned}$$

For marking M, the test commands that label transition (t_1, t_2) allow us to conclude the marking in Line (8).

$$M([x_1{=}n_1]) = 1 \quad M([x_2{=}n_1]) = 1 \quad M([y_1{=}n_2]) = 1 \tag{8}$$
$$M([y_2{\neq}n_2]) = 1 \quad M([y_2{=}a]) = 0 \quad \forall a \in \mathcal{P} \cup \mathcal{N}. \tag{9}$$

In the following Line (9), the implementation of mapping requires a token on $[y_2{\neq}n_2]$. By **(SM3)**, this only gives $M([y_2{=}n_2]) = 0$. We derive that actually all $[y_2{=}a]$ are unmarked as follows. We have $(\underline{G}, M) \in \mathcal{B}$, which means M is known to correspond to a process. This process has a substitution that does not contain y_2 in its domain. This is due to **(DOM)** in combination with the fact that y_2 is bound. Constraint **(COR1)** yields $M([y_2{=}a]) = 0$ for all $a \in \mathcal{P} \cup \mathcal{N}$.

$$M([x{\neq}n_r]) = 1\ \forall x \in \mathcal{I} \cup \mathcal{F} \cup \{r_*\} \quad M([x{=}n_r]) = 0\ \forall x \in \mathcal{I} \cup \mathcal{F} \cup \mathcal{R} \tag{10}$$
$$M([r{=}n]) = 0\ \forall n \in \mathcal{N}. \tag{11}$$

That t_r is enabled gives the first marking in Line (10). With **(SM1)** and **(SM3)**, we conclude that no name maps to n_r. Like for y_2, we get that r does not map to any fresh name, Line (11).

In the control flow, marking M' differs from M in that (S_1', λ_1') and $(S_2', \lambda_2 \cdot \lambda_2')$ instead of (S_1, λ_1) and (S_2, λ_2) are marked in $N(S_{Init,1})$ and $N(S_{Init,2})$. For the substitution net, we only give the places on which the marking has changed. The following is immediate from the definition of implementation:

$$M'([y_2{=}n_2]) = 1 \qquad M'([y_2{\neq}n_2]) = 0$$
$$M'([r{=}n_r]) = 1 \qquad M'([r_*{\neq}n_r]) = 0.$$

We note that $M' = M_1' \cup M_2'$ is stable. Moreover, marking M_1' on N_{Subst} is indeed a substitution marking.

We now argue that (1.a) there is $G' \in \mathcal{R}(F)/_{\equiv}$ so that $\underline{G} \hookrightarrow \underline{G'}$ and (1.b) $(\underline{G'}, M') \in \mathcal{B}$.
Claim 1.a: There is $G' \in \mathcal{R}(F)/_{\equiv}$ so that $\underline{G} \hookrightarrow \underline{G'}$ We assume that $\underline{G}$ and $M_1 \cup M_2$ are related by $\mathcal{B}$. Hence, there is a process in normal form that satisfies

$$G \equiv \nu\tilde{a}.(S_1\sigma_1 \mid S_2\sigma_2 \mid \ldots \mid S_n\sigma_n).$$

From marking $M_1 \cup M_2$, we now derive the following information:

$$S_1 = \overline{x_1}\langle y_1\rangle.S_1' + M_1 \qquad \sigma_1(x_1) = \sigma_2(x_2) \in \tilde{a}$$
$$S_2 = x_2(y_2).\nu r.S_2' + M_2 \qquad \sigma_1(y_1) \in \tilde{a}.$$

The equalities on S_1 and S_2 are due to the markings of $N(S_{Init,1})$ and $N(S_{Init,2})$. For the substitution, we use the fact that M_1 corresponds to $\sigma_1 \cup \ldots \cup \sigma_n$. We have $M_1([x_1{=}n_1]) = 1 = M_1([x_2{=}n_1])$ with $n_1 \in \mathcal{N}$. Since x_1 and x_2 are bound to at most one name by **(SM2)**, this allows us to conclude that the markings of $[x_1{=}n]$ and $[x_2{=}n]$ coincide for all names $n \in \mathcal{N}$. Hence, we get $\sigma_1(x_1) = \sigma_2(x_2)$ by **(COR4)**. By **(DOM)**, we have that $\sigma_1(x_1) \in \tilde{a} \cup \mathcal{P}$. If $\sigma_1(x_1)$ was in $\mathcal{P}$, we had $M_1([x_1{=}\sigma_1(x_1)]) = 1$ by **(COR2)**. This is not the case, hence $\sigma_1(x_1) \in \tilde{a}$. For y_1, the reasoning is similar. We already mentioned above that $\{y_2, r\} \notin \mathit{dom}(\sigma_1 \cup \ldots \cup \sigma_n)$.

The normal form process has a reaction to

$$G' \stackrel{\text{df}}{=} \nu\tilde{a}.a_r.(S_1'\sigma_1 \mid S_2'\sigma_2' \mid \ldots \mid S_n\sigma_n) \quad \text{with} \quad \sigma_2' \stackrel{\text{df}}{=} \sigma_2\{\sigma_1(y_1)/y_2\}\{a_r/r\}.$$

Hence, we have $\underline{G} \hookrightarrow \underline{G'}$. Since G was reachable from F, we have G' reachable from F. Moreover, we already argued in the proof of Lemma 1 that G' has the required normal form.
Claim 1.b: $(\underline{G'}, M_1' \cup M_2') \in \mathcal{B}$ It remains to show that G' and M' are related as required. For the threads, the reasoning is as in Lemma 1 above. It remains to check that M_1' corresponds to $\sigma' \stackrel{\text{df}}{=} \sigma_1 \cup \sigma_2' \cup \ldots \cup \sigma_n$. **(COR1)** to **(COR3)** are as before. We have $\sigma'(r) \neq \sigma'(x)$ with $r \neq x \in \mathit{dom}(\sigma')$. This coincides with the fact that $M_1'([r{=}n_r]) = 1$ and $M_1'([x{=}n_r]) = 0$ for all $r \neq x \in \mathcal{I} \cup \mathcal{F} \cup \mathcal{R}$. The remaining equalities are checked as before. □

Proof (of Theorem 1). It remains to show that $\mathcal{B}$ relates $\underline{F}$ and M_0, the initial marking of $N(F)$. By our assumptions from Sect. 4.1, we have

$$F = \nu\tilde{a}.(S_{Init,1}\sigma_1 \mid \ldots \mid S_{Init,n}\sigma_n).$$

Here, $S_{Init,i}$ are choices or calls that have been derived from artificial defining equations. Moreover, $\sigma_i : \tilde{f}_{Init,i} \to \tilde{a} \cup \mathcal{P}$ with

$$\mathit{dom}(\sigma_i) = \tilde{f}_{Init,i} = \mathit{fn}\,(S_{Init,i}) \cup \emptyset.$$

This shows **(DOM)**, and concludes the proof that F is in normal form.

For the initial marking $M_0 = M_{0,1} \cup M_{0,2}$ of $N(F)$, we have that $M_{0,1}$ corresponds to $\sigma_1 \cup \ldots \cup \sigma_n$ as needed. In the control flow nets $N(S_{Init,i})$, we have the necessary tokens on $(S_{Init,i}, \varepsilon)$.

Imitation of transitions holds by Lemma 1 and by Lemma 2. □

5.2 Reachability analysis

Theorem 1 allows one to check for reachability from F by means of $\mathcal{T}_{Stbl}(N(F))$. More precisely, in order to check whether a process $\underline{G}$ is reachable from $\underline{F}$, one has to check the reachability of a stable marking M with $(\underline{G}, M) \in \mathcal{B}$ in $N(F)$ from an initial marking M_0 via a combination of race free transition sequences. We now show that every transition sequence reaching a stable marking can be decomposed into a series of race free transition sequences. This means the restriction to race free sequences is not necessary — it suffices to check the conventional reachability of M in $N(F)$, which is a standard problem implemented in model checking tools.

Lemma 3. *Every transition sequence $M_1 \rightarrow^+ M_2$ between stable markings $M_1, M_2 \in \mathcal{R}_{Stbl}(N(F))$ can be decomposed into a sequence $M_1 \Rightarrow^+ M_2$ of race free transition sequences.*

Proof. Consider a transition sequence $M_1 \rightarrow^+ M_2$ between stable markings $M_1, M_2 \in \mathcal{R}_{Stbl}(N(F))$ that is not race free. This means the sequence reflects at least two of the following reactions: process identifier calls are replaced with their definitions, silent steps are performed, or communications take place. We discuss the case of two concurrent communications between (S_1, λ_1) and (S_2, λ_2) as well as (S_a, λ_a) and (S_b, λ_b). The control flow nets are finite automata that carry a single token. Therefore, all four processes have to belong to different nets $N(S_{Init,1})$ to $N(S_{Init,b})$. To see this, assume to the contrary this was not the case. Since each net only carries one token, two processes would coincide, say (S_1, λ_1) and (S_a, λ_a). But then the two transitions corresponding to the two communications would compete for the token, and only one would be executable. A contradiction. Due to **(NC)**, the map and test instructions in these four nets do not conflict and can be arbitrarily reordered. Hence, the two communications can be decomposed into race free sequences $M_1 \Rightarrow M$ and $M \Rightarrow M_2$. □

So far we would have to check reachability for all stable markings M that are bisimilar with $\underline{G}$. Below we show that in fact it is sufficient to check for reachability of a single such marking.

Lemma 4. *Let M with $(\underline{G}, M) \in \mathcal{B}$ be a stable marking reachable in $N(F)$. Then every other marking M' with $(\underline{G}, M') \in \mathcal{B}$ is also reachable from some valid initial marking of $N(F)$.*

Proof. With Lemma 3, we can assume that M is reachable by a sequence of race free transition sequences, and we denote the sequence of stable markings from the initial marking to M by

$$M_0 \Rightarrow \ldots \Rightarrow M_n = M.$$

By induction on the length of this sequence we show that $M'_n = M'$ is reachable from a valid initial marking M'_0, i.e. from a marking that corresponds to the substitution of the initial process F (cf. Section 4.3).

Note that, as M and M' are both bisimilar to $\underline{G}$, they differ only in their substitution markings. For the base case, $M_n = M_0$ is an initial marking in $N(F)$, and as M' is bisimilar to the same processes (and so corresponds to the same substitutions), M' must be such an initial marking as well.

For the induction step, there are a stable marking M_{n-1} and a race free transition sequence t, so that M_n is reachable from M_{n-1} via t and M_{n-1} is bisimilar to some process $\underline{G_{n-1}}$. We construct a marking M'_{n-1} as follows: for all places $[x{=}a], [x{=}b]$ in N_{Subst} where $M_n([x{=}a]) = M'_n([x{=}b]) = 1$, set $M'_{n-1}([x{=}b]) \stackrel{\text{df}}{=} M_{n-1}([x{=}a])$. This means in M' name b takes the role that name a has in M, and we propagate this change to the predecessor marking. Set the marking of all other places to 0 and copy the markings of the control flow nets from M_{n-1}. Now M'_{n-1} is a stable marking that is bisimilar to $\underline{G_{n-1}}$, and there clearly is a race free transition sequence t' by which M'_n is reachable from M'_{n-1}. By the induction hypothesis, M'_{n-1} is reachable from an initial marking M'_0. □

Lemmas 3 and 4 together with Theorem 1 establish the result that, in order to check the reachability of a process $\underline{G}$ from a process $\underline{F}$, one computes a single stable marking M with

$(\underline{G}, M) \in \mathcal{B}$ and checks reachability of M in $N(F)$ from some valid initial marking M_0. Unfortunately, the initial marking M_0 is not uniquely defined, and the reachability of M depends on its choice, i.e. one has to guess a suitable M_0 before performing the reachability check. To describe a better strategy, assume the initial FCP is $F = \nu\tilde{a}.(S_{Init,1}\sigma_1 \mid \ldots \mid S_{Init,n}\sigma_n)$. Recall (Sect. 4.1) that the substitutions σ_i map the artificial formal parameters $\tilde{f}_i = fn(S_{Init,i})$ to public names and names in $\tilde{a}$. Instead of non-deterministically selecting the initial marking of the substitution net, we start with the empty marking and add transitions that set the marking of N_{Subst} so that it corresponds to $\sigma_1 \cup \ldots \cup \sigma_n$. Technically, these transitions are similar to $(TRANS_\nu)$.

With these considerations, the reachability problem for FCPs reduces, in polynomial time, to the one for safe PNs:

Proposition 1 (Reachability). *Let F, G be two FCPs, and M be a marking with $(\underline{G}, M) \in \mathcal{B}$. Then $\underline{F} \to^* \underline{G}$ iff $M_0 \to^* M$ in $N(F)$.*

6 Optimisation of the translation

In this section we propose several practical optimisations of the proposed translation of FCPs to safe PNs. They can significantly reduce the size of the resulting PN and the efficiency of subsequent model checking.

6.1 Communication splitting

Recall that the size of the PN resulting from our translation is dominated by the number of transition modelling communication. We now propose a method of significantly decreasing this number, reducing thus the asymptotic worst-case size of the PN from $O(sz^4)$ down to $O(sz^3)$. Furthermore, its straightforward generalisation yields a polynomial translation from polyadic π-calculus to safe PNs, see Sect. 7.

The idea of the method is to model the communication between potentially synchronisable actions $\overline{a}\langle b\rangle$ and $x(y)$ (corresponding to some stub transitions t' and t'') not by a single atomic step but by a pair of steps: the first one checks that a and x are mapped to the same value by the substitution (this step is not executable if the corresponding values are different), and the second step maps y to the value of b in the substitution.

This is implemented by creating a new control place p_{middle} 'in the middle' of communication and two sets of transitions, t_i^1 and t_j^2. Transitions t_i^1, created for each $i \in dom(a) \cap dom(x)$, work as follows. Each t_i^1

- consumes tokens from the input places of the stubs t' and t'' and produces a token on p_{middle};
- checks by read arcs that $[a = i]$ and $[x = i]$ are marked (i.e. the substitution maps a and x to the same value i and thus the synchronisation is possible).

Transitions t_j^2, created for each $j \in dom(b) \cap dom(y)$, work as follows. Each t_j^2

- consumes a token from p_{middle} and produces tokens on the output places of the stubs t' and t'';
- checks by a read arc that $[b = j]$ is marked, consumes a token from $[y \neq j]$ (if it exists) and produces a token on $[y = j]$ (mapping thus in the substitution y to j, i.e. to the value of b).

If the synchronisation is possible in the current state of the system (i.e. a and x have the same value), exactly one of the transitions t_i^1 is enabled (depending on the common value of a and x); else none of these transitions is enabled. Once some t_i^1 fires, exactly one of the transitions t_j^2 becomes enabled (depending on the value of b), and firing it assigns the value of b to y.

6.2 Abstractions of names

In the PN representation of the substitution described in Sect. 3.1, each bound name and formal parameter is represented by a separate row of places. In practice, it is often the case that some bound names and formal parameters can never be simultaneously active, and so can share the same row of places.

We have implemented a simple sharing scheme by introducing an equivalence $\sim$ on the set of bound names and formal parameters, such that if two names are equivalent then they cannot be simultaneously active. Then the rows of the substitution table will correspond to the equivalence classes of $\sim$, and for each bound name and formal parameter we will introduce the *abstraction* operator *abs*, mapping a name to the corresponding equivalence class of $\sim$. Now the operations on the substitution (initialisation of a restricted name, remapping and unmapping) can be performed on the abstraction of names rather than names themselves.

A possible choice of equivalence $\sim$ and the related abstraction is as follows. For each name $b \in \mathcal{R} \cup \mathcal{I} \cup \mathcal{F}$, we denote by $\mathit{thread}(b)$ the thread where b is defined (note that due to **(NC)** and the assumption that threads do not share defining equations, $\mathit{thread}(b)$ is unique). Furthermore, we define

$$\mathit{type}(b) \stackrel{\text{df}}{=} \begin{cases} 0 \text{ if } b \in \mathcal{R}; \\ 1 \text{ otherwise (i.e. if } b \in \mathcal{I} \cup \mathcal{F}), \end{cases}$$

and $\mathit{depth}(b)$ to be the number of names $b' \in \mathcal{R} \cup \mathcal{I} \cup \mathcal{F}$ in whose scope b resides and such that $\mathit{type}(b) = \mathit{type}(b')$. Then the abstraction of b can be defined as a tuple

$$\mathit{abs}(b) \stackrel{\text{df}}{=} \Big(\mathit{thread}(b), \mathit{type}(b), \mathit{depth}(b)\Big),$$

and two names are considered equivalent w.r.t. $\sim$ iff their abstractions coincide.

Other choices of $\sim$ and *abs* are also possible, and we plan to explore them in our future work.

6.3 Better overapproximations for name domains

Recall that the domain of a bound name or formal parameter is an overapproximation of the set of values from $\mathcal{P} \cup \mathcal{N}$ that it can take. While the rough overapproximation proposed in Sect. 3 is sufficient to make the translation polynomial, its quality can be substantially improved by static analysis, resulting in a much smaller PN. In particular, the number of synchronisations between the communication actions as well as the number of transitions implementing a communication can be significantly reduced; furthermore, the number of transitions implementing passing parameters of the calls and the number of places modelling the substitution can also decrease substantially.

Below we outline a simple iterative procedure that can be used to compute better overapproximations. We start by setting $\mathit{dom}(p) \stackrel{\text{df}}{=} \{p\}$ for each public name p. For each restricted name r we set $\mathit{dom}(r) \stackrel{\text{df}}{=} \{r\}$, interpreting this as that the values of r are taken from the set $\mathcal{N}_r$ of unique values (the procedure below never looks inside $\mathcal{N}_r$, and only exploits the fact that the names from $\mathcal{N}_r$ are different from all the other names). The domains of these names are fixed and will not be changed by the procedure. The domains of all the other names occurring in the FCP are initialised to $\emptyset$; they will grow monotonically during the run of the procedure, converging to some overapproximations.

Each iteration of the procedure consists in identifying two actions, $\overline{a}\langle b \rangle$ and $x(y)$ satisfying the following conditions:

- the actions belong to different threads of the FCP;
- $\mathit{dom}(a)$ and $\mathit{dom}(x)$ are not disjoint;
- $\mathit{dom}(b) \not\subseteq \mathit{dom}(y)$.

If no two such actions can be found, the procedure stops, returning the current values of the domains. Otherwise, $dom(y)$ is replaced by its union with $dom(b)$. Intuitively, the above conditions check that the two actions can potentially synchronise, and so y can be mapped to the value of b, and so its domain has to include that of b.

6.4 Better overapproximation for $|\mathcal{N}|$

The cardinality of $\mathcal{N}$ is an important parameter of the translation, affecting the efficiency of almost all its aspects. While the rough overapproximation proposed in Sect. 3 (taking $|\mathcal{N}|$ to be the total number of bound names and formal parameters) is sufficient to make the translation polynomial, a better one can make the translation much more practical and amenable to model checking.

In the worst case, all the currently active names from $\mathcal{R} \cup \mathcal{I} \cup \mathcal{F}$ in the system can be assigned different values from $\mathcal{N}$. To improve the overapproximation of Sect. 3, we observe that in many cases not all such names can be simultaneously active, i.e. it is enough to overapproximate the number of such names that can be simultaneously active.

Hence we propose the following improved overapproximation of $|\mathcal{N}|$. If there are no occurrences of the restriction operator in the FCP, we set $|\mathcal{N}|$ to 0. Otherwise, for each thread we compute the maximal number of names from $\mathcal{R} \cup \mathcal{I} \cup \mathcal{F}$ that can be simultaneously active in it,

$$\max\{|fn\,(S') \cup \lambda| \mid (S, \varepsilon) \text{ a defining process and } (S, \varepsilon) \twoheadrightarrow^* (S', \lambda)\},$$

and set $|\mathcal{N}|$ to the sum of these numbers.

The number of names from $\mathcal{R} \cup \mathcal{I} \cup \mathcal{F}$ that can be simultaneously active in a thread can be computed by separately computing this parameter for each of the defining equations belonging to this thread, as well as the sub-term of the main term of the FCP corresponding to this thread, and taking the maximum of these values. Since these π-calculus expressions are sequential, their parse trees can have only the $+$ operator in every node where a branching occurs, and so the sought value is simply the maximum of the numbers of active names from $\mathcal{R} \cup \mathcal{I} \cup \mathcal{F}$ in the leafs of this parse tree. Furthermore, the names whose domains contain no restricted names can be ignored by this analysis.

6.5 Sharing subnets for unmapping names

When we call $K\lfloor \tilde{a} \rfloor$, some names have to be unmapped in the substitution. The subnet for unmapping a particular name can be shared by all points where such unmapping is necessary. This reduces the size of the resulting PN. This optimisation is especially effective when name abstractions (see above) are used, as the sharing increases significantly in such a case.

6.6 Re-ordering parameters of calls

Consider the following FCP:

$$K\lfloor x, y \rfloor := L\lfloor y, x \rfloor$$
$$L\lfloor u, v \rfloor := K\lfloor v, u \rfloor$$
$$K\lfloor a, b \rfloor$$

When abstractions of the names are computed, the equivalence relation has two equivalence classes, $\{x, u\}$ and $\{y, v\}$. Hence, the substitution has to be modified every time the calls are performed, as the call parameters keep getting flipped.

If the order of the formal parameters in one of the defining equations is changed (together with the order of the factual parameters in the corresponding calls), e.g. as shown below, the

substitution would not require any changes, which significantly reduces the size of the resulting net:

$$K\lfloor x, y\rfloor := L\lfloor x, y\rfloor$$
$$L\lfloor v, u\rfloor := K\lfloor v, u\rfloor$$
$$K\lfloor a, b\rfloor$$

This example illustrates that the order of formal parameters in the defining equations matters, and the translation can gain some savings by changing this order. Searching for the best order of formal parameters can be formulated as an optimisation problem, with the cost function being the total number of changes required in the substitution for all the calls.

6.7 Dropping the restrictions in the main term of the FCP

All the restrictions occurring in the main term of the FCP, can be dropped, making thus the formerly restricted names public (due to **(NC)**, no name clashes can be introduced in this way). This transformation yields a bisimilar π-calculus process, but the corresponding PN becomes smaller.

6.8 Separate pools of values for restricted names

Creation of new names (see Sect. 3.2) introduces arbitration between the values in $\mathcal{N}$, as one of them that is currently unused has to be chosen to initialise a given restricted name. Such arbitration can adversely affect the efficiency of some model checking methods.

It is possible to completely eliminate such arbitration by splitting the set $\mathcal{N}$ into several pools, one for each thread of the FCP, and initialise restricted names in each thread only from the corresponding pool, by sequentially looking for the first unused value in it. This however increases the size of the resulting PN. Moreover, if symmetries reduction is used in model checking, the problem vanishes, see below.

6.9 Using symmetries

The proposed translation introduces a number of symmetries in the resulting PN:

- The values in $\mathcal{N}$ (and thus the corresponding columns of the substitution, see Fig. 1) are interchangeable;
- When enforcing the assumption that threads do not share any defining equations as explained in Sect. 2, some defining equations are replicated.

Hence, it is desirable to exploit these symmetries during model checking. In particular, this would allow for efficient handling of the arbitration arising when a value from $\mathcal{N}$ has to be chosen so that a restricted name r can be initialised with it: if symmetries are used, all the immediate successor states after such an arbitration are equivalent, and so only one of them has to be further explored.

6.10 Translation to different PN classes

The proposed translation produces a safe PN, as this PN class is particularly simple and suited for formal verification. However, if the used model checking method can cope with more powerful PN classes, the following changes can be made.

Translation to bounded PNs In this case, for each $val \in \mathcal{N}$, we can fuse the places $[var \neq val]$, where $var \in \{r_*\} \cup \mathcal{I} \cup \mathcal{F}$, into one, denoted by $[* \neq val]$, replacing thus $|\mathcal{N}| \cdot (|\mathcal{I}| + |\mathcal{F}| + 1)$ safe places with $|\mathcal{N}|$ places of capacity $|\mathcal{I}| + |\mathcal{F}| + 1$. It is still possible to perform all the necessary operations with the substitution; in particular, to find a value $val \in \mathcal{N}$ to which no bound name or formal parameter is currently mapped, and map a given restricted name r_k to val, the PN transition performing the initialisation has to:

- consume by a weighted arc $|\mathcal{I}| + |\mathcal{F}| + 1$ tokens from $[* \neq val]$ (checking thus that val is not currently assigned to any name) and return by a weighted arc $|\mathcal{I}| + |\mathcal{F}|$ tokens back to this place;
- produce a token at $[r_k = val]$ (performing thus the initialisation of r_k with the value val).

Translation to coloured PNs In this case the symmetries present in the PN can be used to fold it. In particular:

- The values in $\mathcal{N}$ are interchangeable, and so the corresponding columns of the substitution can be folded into one column, by giving the tokens corresponding to the elements of $\mathcal{N}$ unique colours.
- One can avoid enforcing the assumption that threads do not share any defining equations (see Sect. 2), and instead use coloured control tokens that are unique for each thread of the FCP.

7 Extensions

In this section we demonstrate how the proposed FCP to PN translation can be adapted to some often used extensions of the basic FCP calculus. In particular, we consider the introduction of match and mismatch operators and polyadic communication.

7.1 Match and mismatch operators

The match and mismatch operators are a common extension of π-calculus. Intuitively, the process $[x = y].S$ behaves as S if $x = y$ and does nothing otherwise, and the process $[x \neq y].S$ behaves as S if $x \neq y$ and does nothing otherwise. To handle these operators, we extend the construction of $N(S_{Init})$ with the following transitions. For each $a \in \mathcal{P} \cup \mathcal{N}$, we have

$$([x = y].S, \lambda) \xrightarrow[\{test([x=a]), test([y=a])\}]{\tau} (S, \lambda) \qquad ([x \neq y].S, \lambda) \xrightarrow[\{test([x=a]), test([y\neq a])\}]{\tau} (S, \lambda).$$

For the latter rule, new places $[x{\neq}a]$ complementing $[x{=}a]$ have to be introduced in the substitution net (some of these places already exist). It is possible to avoid introducing such new places by using the following set of transitions for mismatch instead: for each $a, b \in \mathcal{P} \cup \mathcal{N}$ such that $a \neq b$, we have the transition

$$([x \neq y].S, \lambda) \xrightarrow[\{test([x=a]), test([y=b])\}]{\tau} (S, \lambda).$$

This set of transitions, however, is too large, and thus the former translation is preferable in practice.

Note however that in the presence of match/mismatch operators, the relationship between the behaviours of the original FCP and the PN resulting from the translation becomes somewhat complicated: the latter simulates the former only in a non-deterministic sense, i.e. some executions of the PN are considered invalid and do not correspond to any executions of the original FCP, in particular, false deadlocks could be introduced. For example, in the process $[x = y].[u = v].S$ the former guard can be true while the latter is false, in which case the resulting PN will get stuck between the guards, whereas this does not happen in the original π-calculus process. Nevertheless, such invalid executions can easily be distinguished from valid ones, and so the resulting PN is still suitable for model checking.

7.2 Polyadic π-calculus

Polyadic communication exchanges multiple names in a single reaction. Intuitively, a sending prefix $\overline{a}\langle x_1 \dots x_m\rangle$ (with $m \geq 0$) and a receiving prefix $b(y_1 \dots y_n)$ (with $n \geq 0$ and all y_i being different names) can synchronise iff $\sigma(a) = \sigma(b)$ and $m = n$, and after synchronisation each y_i gets the value of x_i. Formally,

$$\text{(React)} \quad (a(\tilde{y}).P + M) \,|\, (\overline{a}\langle\tilde{x}\rangle.Q + N) \to P\{\tilde{x}/\tilde{y}\} \,|\, Q \quad \text{if } |\tilde{y}| = |\tilde{x}|.$$

A polynomial translation of this extension generalises the 'communication splitting' idea described in Sect. 6: It is possible to perform such communication in stages, where at the first step one checks that a and b are mapped to the same value by the substitution (this step is not executable if the corresponding values are different), and the subsequent steps map, one-by-one, y_i to the value of x_i in the substitution.

8 Experimental results

To demonstrate the practicality of our approach, we implemented the proposed translation of FCPs to safe PNs in the tool Fcp2Pn and tested the translation on a number of benchmarks;[3] to demonstrate that the resulting PNs are suitable for practical model checking they were checked for deadlocks. These benchmarks are briefly described below; note that we occasionally violate the **(NC)** assumption to improve readability.

The *NESS (Newcastle E-Learning Support System)* series of benchmarks, modelling an electronic coursework submission system, is taken from [11]. The model consists of a teacher process T composed in parallel with k students S (the system can be scaled up by increasing the number of students) and an environment process ENV. Every student has its own local channel for communication, h_i, and all students share the channel h:

$$\nu h.\nu h_1 \dots \nu h_k.\Big(T\lfloor nessc, h_1, \dots, h_k\rfloor \,|\, \prod_{i=1}^{k} S\lfloor h, h_i\rfloor \,|\, ENV\lfloor nessc\rfloor\Big)\,.$$

The idea is that the students are supposed to submit their work for assessment to *NESS*. The teacher passes the channel *nessc* of the system to all students, $\overline{h_i}\langle nessc\rangle$, and then waits for the confirmation that they have finished working on the assignment, $h_i(x_i)$. After receiving the *NESS* channel, $h_i(nsc)$, students organise themselves in pairs. To do so, they send their local channel h_i on h and at the same time listen on h to receive a partner, $\overline{h}\langle h_i\rangle \dots + h(x) \dots$ When they finish, exactly one student of each pair sends two channels (own channel h_i and the channel received from the partner) to the support system, $\overline{nsc}\langle h_i\rangle.\overline{nsc}\langle x\rangle$, which give access to their completed joint work. These channels are received by the ENV process. The students finally notify the teacher about the completion of their work, $\overline{h_i}\langle fin\rangle$. Thus, the system is modelled by:

$$\begin{aligned} T(nessc, h_1, \dots, h_k) &:= \prod_{i=1}^{k} \overline{h_i}\langle nessc\rangle.h_i(x_i).\mathbf{0} \\ S(h, h_i) &:= h_i(nsc).(\overline{h}\langle h_i\rangle.\overline{h_i}\langle fin\rangle.\mathbf{0} + h(x).\overline{nsc}\langle h_i\rangle.\overline{nsc}\langle x\rangle.\overline{h_i}\langle fin\rangle.\mathbf{0}) \\ ENV(nessc) &:= nessc(y_1).\ \dots\ .nessc(y_k).\mathbf{0} \end{aligned}$$

To distinguish the proper termination from deadlocks (where some processes are stuck in the middle of their intended behaviour, waiting for a communication to occur), a new transition was added to the PNs resulting from our translation, creating a loop at the state corresponding

[3] The tool and benchmarks are available from http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/fcp2pn.

Problem	Process size FCP	nfFCP	$\lvert\mathcal{N}\rvert$	Safe PN $\lvert P\rvert$	$\lvert T\rvert$	Dlck [sec]
NESS (04)	110	110	0	137	145	0.02
NESS (05)†	137	137	0	196	246	0.09
NESS (06)	164	164	0	265	385	0.16
NESS (07)†	191	191	0	344	568	0.45
DNESS (06)	118	118	0	157	103	0.02
DNESS (08)	157	157	0	241	169	0.05
DNESS (10)	196	196	0	341	251	0.13
DNESS (12)	235	235	0	457	349	2.27
DNESS (14)	274	274	0	589	463	1.71

Problem	Process size FCP	nfFCP	$\lvert\mathcal{N}\rvert$	Safe PN $\lvert P\rvert$	$\lvert T\rvert$	Dlck [sec]
CS (2,1)	45	54	7	138	149	1.01
CS (2,2)	48	68	10	243	320	0.16
CS (3,2)	51	80	11	284	431	1.28
CS (3,3)	54	94	14	428	728	3.67
CS (4,4)	60	120	18	663	1368	11.73
CS (5,5)	66	146	22	948	2288	46.61
GSM	175	231	12	636	901	4.39
GSM'	174	230	0	355	503	3.09
PHONES	157	157	0	131	94	0.01

Table 1. Experimental results.

to the successful termination of the system. Obviously, the system successfully terminates iff the number of students is even, i.e. they can be organised into pairs.

The *DNESS* model is a refined version of *NESS*, where the pairing of students is deterministic; thus the number of students is always even, and these benchmarks are deadlock-free.

The *CS* (m,n) series of benchmarks models a client-server system with one server, n clients, and the server spawning m sessions that handle the clients' requests:

$$\begin{aligned} \mathit{CLIENT}(\mathit{url}) &:= \nu \mathit{ip}.\overline{\mathit{url}}\langle \mathit{ip}\rangle.\mathit{ip}(s).s(x).\mathit{CLIENT}\lfloor \mathit{url}\rfloor \\ \mathit{SERVER}(\mathit{url},\mathit{getses}) &:= \mathit{url}(y).\mathit{getses}(s).\overline{y}\langle s\rangle.\mathit{SERVER}\lfloor \mathit{url},\mathit{getses}\rfloor \\ \mathit{SESSION}(\mathit{getses}) &:= \nu \mathit{ses}.\overline{\mathit{getses}}\langle \mathit{ses}\rangle.\overline{\mathit{ses}}\langle \mathit{ses}\rangle.\mathit{SESSION}\lfloor \mathit{getses}\rfloor \\ \nu \mathit{getses}\Big(\mathit{SERVER}(\mathit{url},\mathit{getses}) \mid \prod_{i=1}^{m} \mathit{SESSION}(\mathit{getses}) \mid \prod_{i=1}^{n} \mathit{CLIENT}(\mathit{url})\Big) \end{aligned}$$

On a client's request, the server creates a new session using the *getses* channel, $\mathit{getses}(s)$. A session is modelled by a *SESSION* process. It sends its private channel $\nu \mathit{ses}$ along the *getses* channel to the server. The server forwards the session to the client, $\overline{y}\langle s\rangle$, which establishes the private session, and becomes available for further requests. This case study uses recursion and is scalable in the number of clients and the number of sessions. All these benchmarks are deadlock-free.

The *GSM* benchmark is the well-known specification of the handover procedure in the GSM Public Land Mobile Network. We use the standard π-calculus model with one mobile station, two base stations, and one mobile switching center presented in [18]. We also generated a variant of this benchmark where the sender process

$$\mathit{SENDER}(-) := \nu d.\overline{\mathit{in}}\langle d\rangle.\mathit{SENDER}\lfloor -\rfloor$$

is replaced with

$$\mathit{SENDER}(-) := \overline{\mathit{in}}\langle d\rangle.\mathit{SENDER}\lfloor -\rfloor,$$

i.e. instead of creating (with the ν operator) a new message each time, it sends the same public channel. Since the content of the message is not important, this change is inconsequential from the modelling point of view. However, it significantly reduces the size of the resulting PN, as the modified specification contains no restriction operator and so $\mathcal{N} = \emptyset$.

The *PHONES* benchmark is a classical example taken from [15]. It is another example of a handover procedure for mobile phones communicating with fixed transmitters, where the phones have to switch their transmitters on the go.

The experimental results are given in Table 1, where the meaning of the columns is as follows (from left to right): name of the case study (benchmarks marked with † contain deadlocks); sizes of the original FCP and its normal form (see Sect. 2), and cardinality of $\mathcal{N}$ determined

by static analysis; number of places and transitions in the resulting safe PN; and deadlock checking time.

The experiments were conducted on a PC with an Intel Core 2 Quad Q9400 2.66 GHz (quad-core) processor (a single core was used) and 4G RAM. The deadlock checking was performed with the LoLA tool,[4] configured to assume the safeness of the PN (`CAPACITY 1`), use the stubborn sets and symmetry reductions (`STUBBORN`, `SYMMETRY`), compress states using P-invariants (`PREDUCTION`), use a light-weight data structure for states (`SMALLSTATE`), and check for deadlocks (`DEADLOCK`). The FCP to PN translation times were negligible (< 0.1 sec) in all cases and so are not reported.

The experiments indicate that the sizes of the PNs grow moderately with the sizes of the FCPs, and the PNs are suitable for efficient model checking.

9 Conclusions

We developed a polynomial translation from finite control processes (an important fragment of π-calculus) to safe low-level Petri nets. To our knowledge, this is the first such translation. Furthermore, there is a close correspondence between the control flow of the π-calculus specification and the resulting PN, and the latter is suitable for practical model checking. The translation has been implemented in a tool Fcp2Pn, and the experimental results are encouraging.

We have also proposed a number of optimisations allowing one to reduce the size of the resulting PN, as well as a number of extensions, in particular the match/mismatch operators and polyadic π-calculus.

In future work we plan to further improve the translation by using a more thorough static analysis, and to incorporate the translation into different model checking tool-chains, in particular, ones based on PN unfolding prefixes (currently none of the PN unfolders uses symmetry reduction, which is essential for efficient model checking of PNs resulting from our translation).

Acknowledgements The authors would like to thank Ivan Poliakov for his help in producing the experimental results. This research was supported by the Epsrc grant EP/G037809/1 (Verdad).

References

1. R. Amadio and C. Meyssonnier. On decidability of the control reachability problem in the asynchronous π-calculus. *Nord. J. Comp.*, 9(1):70–101, 2002.
2. E. Best, R. Devillers, and M. Koutny. *Petri Net Algebra.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2001.
3. N. Busi and R. Gorrieri. Distributed semantics for the π-calculus based on Petri nets with inhibitor arcs. *J. Log. Alg. Prog.*, 78(1):138–162, 2009.
4. L. Cardelli and A. Gordon. Mobile ambients. In *Proc. FOSSACS'98*, volume 1378 of *LNCS*, pages 140–155. Springer, 1998.
5. M. Dam. Model checking mobile processes. *Inf. Comp.*, 129(1):35–51, 1996.
6. R. Devillers, H. Klaudel, and M. Koutny. A compositional Petri net translation of general π-calculus terms. *For. Asp. Comp.*, 20(4–5):429–450, 2008.
7. C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. of ICALP*, volume 1443 of *LNCS*, pages 103–115. Springer, 1998.
8. J. Esparza. Decidability and complexity of Petri net problems—an introduction. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 374–428. Springer, 1998.
9. G.-L. Ferrari, S. Gnesi, U. Montanari, and M. Pistore. A model-checking verification environment for mobile processes. *ACM Trans. Softw. Eng. Methodol.*, 12(4):440–473, 2003.

[4] Available from http://service-technology.org/tools/lola.

10. R. Gorrieri and R. Meyer. On the relationship between pi-calculus and finite place/transition petri nets. In *Proc. of CONCUR*, volume 5170 of *LNCS*, pages 463–480. Springer, 2009.
11. V. Khomenko, M. Koutny, and A. Niaouris. Applying Petri net unfoldings for verification of mobile systems. In *Proc. of MOCA*, Bericht FBI-HH-B-267/06, pages 161–178. University of Hamburg, 2006.
12. R. Meyer. On boundedness in depth in the π-calculus. In *Proc. of IFIP TCS*, volume 273 of *IFIP*, pages 477–489. Springer, 2008.
13. R. Meyer. A theory of structural stationarity in the π-calculus. *Acta Inf.*, 46(2):87–137, 2009.
14. R. Meyer, V. Khomenko, and T. Strazny. A practical approach to verification of mobile systems using net unfoldings. *Fundam. Inf.*, 94:439–471, 2009.
15. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. CUP, 1999.
16. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I. *Inf. Comp.*, 100(1):1–40, 1992.
17. U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In *Proc. of CONCUR*, volume 962 of *LNCS*, pages 42–56. Springer, 1995.
18. F. Orava and J. Parrow. An algebraic verification of a mobile network. *For. Asp. Comp.*, 4(6):497–543, 1992.
19. F. Peschanski, H. Klaudel, and R. Devillers. A Petri net interpretation of open reconfigurable systems. In *Proc. of Petri nets'11*, volume 6709, pages 208–227. Springer, 2011.
20. M. Pistore. *History Dependent Automata*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1999.
21. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. CUP, 2001.
22. B. Victor and F. Moller. The mobility workbench: A tool for the π-calculus. In *Proc. of CAV*, volume 818 of *LNCS*, pages 428–440. Springer, 1994.