A Theory of Name Boundedness

Reiner Hüchting[1], Rupak Majumdar[2], and Roland Meyer[1]

[1]University of Kaiserslautern [2]MPI-SWS

Abstract. We develop a theory of *name-bounded π-calculus processes*, which have a bound on the number of restricted names that holds for all reachable processes. Name boundedness reflects resource constraints in practical reconfigurable systems, like available communication channels in networks and address space limitations in software.

Our focus is on the algorithmic analysis of name-bounded processes. First, we provide an extension of the Karp-Miller construction that terminates and computes the coverability set for any name-bounded process. Moreover, the Karp-Miller tree shows that name-bounded processes have a pumping bound as follows. When a restricted name is distributed to a number of sequential processes that exceeds this bound, the name may be distributed arbitrarily. Second, using the bound, we construct a Petri net bisimilar to the name-bounded process. The Petri net keeps a reference count for each restricted name, and recycles names that are no longer in use. The pumping property ensures that bounded zero tests are sufficient for recycling. With this construction, name-bounded processes inherit decidability properties of Petri nets. In particular, reachability is decidable for them. We complement our decidability results by a non-primitive recursive lower bound.

1 Introduction

The π-calculus is an established formalism for modeling and reasoning about reconfigurable systems that dynamically create and destroy communication links at runtime. While Turing complete in general, there are interesting subclasses of π-calculus where important verification problems remain decidable. In this paper, we propose and investigate a natural subclass of π-calculus processes: *name-bounded processes*. These are processes that have a bound on the number of restricted names that holds for all reachable processes.

Name-bounded processes are interesting for various reasons. First, from a theoretical perspective, they form a natural subclass of π-calculus that is expressive enough to subsume many other classes, but for which, as we will show, analysis questions remain decidable. Second, from a practical perspective, name boundedness captures natural constraints on an implementation that limit the total number of physical resources used at any point without constraining the number of processes. For example, a networked system can use a limited number of IP addresses or communication ports, but allow many clients to multiplex these resources. Finally, name boundedness is useful in verification as an approximate model for Turing-complete systems. For example, a finite abstraction of the shared heap in a concurrent program leads to a name-bounded model.

Motivated by these applications in modeling, synthesis, and verification, we focus on the algorithmic analysis of name-bounded systems. Coverability is decidable for the model using generic well-structuredness arguments. Our main contributions, listed below, show effectiveness and decidability results beyond coverability. We also outline the theoretical tools we develop for the proofs.

Contribution 1: Synthesis Given an *arbitrary* π-calculus process P and a bound $b \in \mathbb{N}$, we prove it decidable to check whether P is name-bounded by b. Phrased differently, given a system P and a resource constraint b on the execution environment, one can decide if all executions of P use at most b resources. Note that the result is non-trivial since 0-bounded systems may already have an infinite state space (they can simulate Petri nets). It should rather be contrasted with the general name boundedness problem, which of course is undecidable.

Tool 1: Karp and Miller algorithm Behind this result is a novel extension of the Karp and Miller construction [9]. Given a process, our algorithm computes a finite representation of the downward closure of the reachability set; and the computation terminates precisely for name-bounded processes. Extending the Karp and Miller construction for Petri nets to name-bounded processes is non-trivial, since it is not sufficient to merely compare two limit elements (as for Petri nets). We extend the construction by tracking names instantiated in intermediary transitions to decide when a transition sequence can be accelerated.

Contribution 2: Petri net construction As second contribution we show that every name-bounded processes can be translated into a bisimilar Petri net. As a result, even subtle verification problems like reachability are decidable for name-bounded systems.

Tool 2: Pumping constant The idea behind the Petri net construction is to provide a finite set of instances $(a, 0), \dots, (a, b-1)$ that can be used to represent the restricted name a. If we encounter a restriction νa in a run, we represent it by an instance that is not present in the current process. To check that an instance is not used, we keep a reference count. The instance is not used if and only if this count is zero.

Unfortunately, Petri nets cannot perform zero-tests. Instead, we show that the reference counts can be bounded using a pumping constant $p \in \mathbb{N}$ such that once an instance (a, k) is known to more than p processes, it can be distributed to arbitrarily many processes. This has the following consequence. If the reference counter of an instance (a, k) exceeds p, the instance cannot be reused. Phrased differently, we only need to keep the precise reference count up to p. This bound allows us to implement zero-tests with a Petri net. We show that the pumping constant can be computed from the Karp-Miller tree.

Contribution 3: Lower bound We show that verification problems for name-bounded processes, be it coverability or reachability, have non-primitive recursive space complexity. In contrast, reachability for the related model of Petri nets is only known to be EXPSPACE-hard [10], and coverability is EXPSPACEcomplete [18]. Moreover, most known extensions of Petri nets with non-primitive recursive lower bounds for coverability have undecidable reachability problems. **Tool 3: Space-bounded Turing machine simulation** To establish the lower bound result, we show how to simulate Turing machines where the tape-size is bounded by the Ackermann-function $A(n)$ with name-bounded processes. Our construction combines a classical result for Petri nets with the expressiveness of π-calculus as follows. With the construction of Mayr and Meyer [11], we obtain a process that generates up to $A(n)$ waiting processes and terminates. We then use a π-calculus modeling trick to arrange these processes into a list representing the Turing tape, and simulate the machine by communicating the state of the machine between tape cells.

Thus, our results resolve —positively— the algorithmic landscape of a natural and expressive fragment of the π-calculus.

Related Work We recall the translations of reconfigurable systems into place/transition Petri nets that have been proposed in the literature. None of them can handle the name-bounded processes we consider here. Translations for restriction-bounded processes (where restrictions do not occur inside recursion) can be found in [14, 3]. These works propose an identity-aware semantics similar to the one we use here, but do not introduce reference counters and pumping constants. The reason we need these tools is the generality of name-bounded processes. Our new class strictly subsumes restriction-bounded and, at the same time, finite control processes (FCPs) [5]. We are thus faced with unbounded parallelism combined with unboundedly many names. FCPs are translated into polynomial-sized safe Petri nets in [15]. We show that such a compact encoding cannot exist for name-bounded processes. In the worst case, the Petri nets have to be non-primitive recursive. The structural translation [13] identifies groups of processes that share restricted names. It yields a finite representation precisely for the class of structurally stationary processes, which are incomparable with name-bounded processes.

There are alternative translations of reconfigurable systems into higher-level Petri nets. Due to the expressive target formalism, it is not possible to deduce decidability results for properties like reachability from them. The class in [1] restricts the processes that can receive on a generated name. A translation into transfer nets yields decidability of control reachability. A translation of π-calculus into high-level Petri nets is given in [6]. In [17], FCPs are encoded into history dependent automata where states are labelled by names to represent restrictions.

Decidability of reachability in related process models, such as a fragment of mobile ambients [4] and a variant of CCS [8], has been shown by reduction to Petri net reachability. Unlike these papers, our construction gives a stronger correspondence: a Petri net that is bisimilar to a name-bounded process.

Name-bounded processes are bounded in depth in the sense that the nesting of restrictions is limited [12]. Therefore, positive results for depth-bounded systems, like decidability of coverability [20], carry over to name boundedness. However, reachability is undecidable for depth-bounded systems [14], and the coverability set is not computable [2]. This motivated our search for subclasses of depth boundedness and we show here that both problems, reachability and coverability set computation, can be solved for name-bounded models. Further, we can decide whether an arbitrary process is name-bounded by a given $b \in \mathbb{N}$.

Our results rely on an adaptation of the Karp-Miller construction [9]. Finkel and Goubault-Larrecq [7] extend the same algorithm to compute coverability sets for general well-structured systems. There are two reasons why we propose a specific variant. First, we want our algorithm to be sound and complete for general, Turing-complete systems, but terminate only for name-bounded ones. This is needed to decide name boundedness when $b \in \mathbb{N}$ is given. The approach in [7] is always guaranteed to terminate, and hence does not handle Turing-complete models. Second, soundness requires an acceleration that depends on the labels along the path. It is unclear how to encode this into [7].

2 Name-Bounded Processes

The π-calculus We recall the basics on π-calculus [16, 19]. The π-calculus encodes computation using processes that exchange messages over channels. Messages and channels are untyped: a message that is received may serve as channel in further interactions. Formally, messages and channels are *names* a, b, x, y from a countable set of names $\mathcal{N}$. Processes communicate by synchronizing on *prefixes* π that are *sending* $\overline{x}\langle y\rangle$ or *receiving* $x(z)$. From these elementary communications, we build models of reconfigurable systems using non-deterministic choice $+$, parallel composition $\mid$, restriction νa, and parameterized recursion $K\lfloor\tilde{a}\rfloor$. To implement recursion, we introduce process identifiers, ranging over K, together with defining equations $K(\tilde{x}) := P$, where P is again a process and $\tilde{x}$ is a sequence of distinct names so that $|\tilde{a}| = |\tilde{x}|$. Every process relies on finitely many defining equations.

Formally, *processes* P, Q, R from the set of processes $\mathcal{P}$ are defined by

$$M ::= \mathbf{0} \;\shortmid\; \pi.P \;\shortmid\; M_1 + M_2 \qquad\qquad P ::= M \;\shortmid\; K\lfloor\tilde{a}\rfloor \;\shortmid\; P_1 \mid P_2 \;\shortmid\; \nu a.P$$

Processes M and $K\lfloor\tilde{a}\rfloor$ are called *sequential* as they are the basic building blocks for parallel compositions. We also use syntax S to indicate that the given process is sequential, and write $\mathcal{S}$ for the set of all sequential processes. We abbreviate k-fold parallel compositions of the same process P by P^k, where $P^0 := \mathbf{0}$.

The intended semantics of a call $K\lfloor\tilde{a}\rfloor$ is that the process behaves like P, but with $\tilde{x}$ replaced by $\tilde{a}$. This replacement is formalized by the application of *substitutions*. A substitution $\{\tilde{a}/\tilde{x}\}$ is a function from $\mathcal{N}$ to $\mathcal{N}$ that maps $\tilde{x}$ to $\tilde{a}$ and leaves all names outside $\tilde{x}$ unchanged. The application of $\{\tilde{a}/\tilde{x}\}$ to process P is denoted by $P\{\tilde{a}/\tilde{x}\}$ and defined in the standard way [19].

Receive prefixes $x(y)$ and restrictions νa *bind* the names y and a. A name that is not bound in a process is *free*. We assume free and bound names to be disjoint, and that every name is bound at most once. We use $\mathcal{F}$ to denote the names that can occur free in processes, and $\mathcal{F}(P)$ for the set of free names in P. Similarly, $\mathcal{R}$ is the set of names that can occur in restrictions, and $\mathcal{R}(P)$ the restricted names in P. In a defining equation $K(\tilde{x}) := P$, we require $\mathcal{F}(P) \subseteq \tilde{x}$ to avoid the invention of free names in a recursive call.

To ease the definition of the π-calculus semantics, we define a *structural congruence relation* $\equiv\ \subseteq \mathcal{P} \times \mathcal{P}$. It is the smallest congruence that allows for α-conversion of bound names, requires choice $+$ and parallel $\mid$ to be associative and commutative with $\mathbf{0}$ as neutral element, and where restrictions satisfy

$$\nu a.\mathbf{0} \equiv \mathbf{0} \quad \nu a.\nu b.P \equiv \nu b.\nu a.P \quad \nu a.(P \mid Q) \equiv P \mid \nu a.Q \text{ if } a \notin \mathcal{F}(P).$$

We also use a quasi-ordering $\preceq\ \subseteq \mathcal{P} \times \mathcal{P}$ among processes called *embedding*. It is the smallest relation that satisfies $\nu\tilde{a}.Q \preceq \nu\tilde{a}.(Q \mid R)$ and that is closed under structural congruence: $Q \equiv Q' \preceq R' \equiv R$ entails $Q \preceq R$. It can be shown that $\preceq$ is indeed reflexive and transitive, and thus a quasi-ordering [12]. For a set of processes $\mathcal{P}' \subseteq \mathcal{P}$, we define $\mathcal{P}' \downarrow\ := \{Q \mid \exists P \in \mathcal{P}' : Q \preceq P\}$.

The behavior of processes is given by the *reaction relation* $\rightarrow\ \subseteq \mathcal{P} \times \mathcal{P}$. It is the smallest relation that satisfies

$$x(z).P + M \mid \overline{x}\langle y\rangle.Q + N \rightarrow P\{y/z\} \mid Q \qquad K\lfloor\tilde{a}\rfloor \rightarrow P\{\tilde{a}/\tilde{x}\} \text{ with } K(\tilde{x}) := P,$$

and that is closed under parallel composition, restriction, as well as structural congruence. A process Q is reachable from P if $P \rightarrow^* Q$, where $\rightarrow^*$ is the reflexive transitive closure of $\rightarrow$. The *reachability set* of P, written $\mathit{Reach}(P)$, is the set of all processes reachable from P. The *transition system* of a process is the quotient of the reachable processes along structural congruence: $\mathcal{T}(P) := (\mathit{Reach}(P)/_{\equiv}, \hookrightarrow, \underline{P})$, where $\underline{P} \hookrightarrow \underline{Q}$ iff $P \rightarrow Q$.

We shall use Milner's *standard form* of processes [16], which maximizes the scope of restrictions. The formal definition is inductive. A parallel composition of sequential processes S is in standard form. If P_{sf} is in standard form then also $\nu a.P_{\mathit{sf}}$ is, provided $a \in \mathcal{F}(P_{\mathit{sf}})$.

Name Boundedness Process $P \in \mathcal{P}$ is *name-bounded* if there is a bound on the number of restricted names that holds for all reachable processes $Q \in \mathit{Reach}(P)$. However, restricted names that do not occur free can be removed by structural congruence: $\nu a.P \equiv P$ provided $a \notin \mathcal{F}(P)$. This motivates the definition of the number of *active restrictions*: $\mathit{arn}(S) := 0$, $\mathit{arn}(P \mid Q) := \mathit{arn}(P) + \mathit{arn}(Q)$, and $\mathit{arn}(\nu a.P) := 1 + \mathit{arn}(P)$ if $a \in \mathcal{F}(P)$ and $\mathit{arn}(\nu a.P) := \mathit{arn}(P)$ otherwise.

Definition 1. *Process $P \in \mathcal{P}$ is* b-name-bounded *with $b \in \mathbb{N}$ if for all processes $Q \in \mathit{Reach}(P)$ we have $\mathit{arn}(Q) \leq b$. Process P is* name-bounded *if it is b-name-bounded for some $b \in \mathbb{N}$.*

For example, process $\nu a.K_1\lfloor a\rfloor$ with $K_1(x) := K_1\lfloor x\rfloor^2$ is 1-name-bounded, whereas $\nu b.K_2\lfloor b\rfloor$ with $K_2(y) := \nu b.(K_2\lfloor b\rfloor^2)$ is not name-bounded.

The definition of name boundedness refers to all reachable processes. This motivates the following decision problems. The *name boundedness problem (***NB***)* asks, given $P \in \mathcal{P}$, is P name-bounded? The *restricted name boundedness problem (***RNB***)* asks, given $P \in \mathcal{P}$ and $b \in \mathbb{N}$, if P is b-name-bounded.

Theorem 1. *(1)* **NB** *is r.e.-complete and (2)* **RNB** *is decidable.*

Decidability of **RNB**, from which the recursive enumerability of **NB** follows, is not obvious, as 0-name-bounded systems can already simulate Petri nets. We prove the result using an algorithm for effectively constructing coverability sets for name-bounded processes that we describe next.

3 Coverability and the Karp-Miller Construction

A process Q is said to be *coverable* from P if there exists some $R \in \mathit{Reach}(P)$ such that $Q \preceq R$. The *coverability set* of P is the set of all processes coverable from P. Equivalently, the coverability set is the downward closure, with respect to the embedding order, of the reachability set: $\mathit{Reach}(P)\downarrow$. Given a name-bounded process P, our next goal is to compute a *finite* representation for $\mathit{Reach}(P)\downarrow$.

We construct the coverability set by unfolding the reachability tree of P, and accelerating reaction sequences that can be repeated. The acceleration procedure takes a sequence $Q_1 \to^* Q_2$ and constructs a closed-form representation of all processes that are coverable with any number of iterations of the sequence. We call the procedure a Karp-Miller tree, since it closely resembles the data structure used for the coverability analysis of Petri nets [9, 7].

3.1 Identity-aware Processes

In the Karp-Miller tree construction, restricted names have to be handled with care. To see the problem, reconsider $K_1(x) := K_1\lfloor x\rfloor^2$ and $K_2(y) := \nu b.(K_2\lfloor b\rfloor^2)$ from above. Then $\nu a.K_1\lfloor a\rfloor$ is name-bounded, but $\nu b.K_2\lfloor b\rfloor$ is not. In their reachability trees, reaction $\nu a.K_1\lfloor a\rfloor \to \nu a.(K_1\lfloor a\rfloor^2)$ uses copies of the same name a, while $\nu b.K_2\lfloor b\rfloor \to \nu b.(K_2\lfloor b\rfloor^2)$ forgets name b and re-creates it again.

The examples suggest that we have to track the identities of names over transitions. Inspired by [14], we introduce a notion of *identity-aware processes.* They replace restricted names νa by free names of the form (a, i) taken from a set of *instances.* Since free names are not subject to α-conversion, instances are preserved by transitions. To mimic name boundedness, transitions among identity-aware processes do not choose instances (a, i) arbitrarily, but compute the least index i that is not present in the target process. Moreover, identity-aware transitions are labelled by the newly generated instances, which allows us to distinguish them from old ones. In the example above, we obtain

$$K_1\lfloor(a,0)\rfloor \xrightarrow{\emptyset}_{ia} K_1\lfloor(a,0)\rfloor^2 \quad \text{as opposed to} \quad K_2\lfloor(b,0)\rfloor \xrightarrow{\{(b,0)\}}_{ia} K_2\lfloor(b,0)\rfloor^2.$$

Formally, an *instance of a restricted name* $a \in \mathcal{R}$ is a pair (a, i) from the set of instances $\mathcal{I} := \mathcal{R} \times \mathbb{N}$. A process is called *identity-aware* if it has the form

$$P_{ia} = S_1 \mid \ldots \mid S_n \quad \text{with} \quad \mathcal{F}(P_{ia}) \subseteq \mathcal{F} \cup \mathcal{I}.$$

As it is a parallel composition of choices and calls, there are no active restrictions. Moreover, and different from ordinary processes, P_{ia} is allowed to have some instances *free*. We use $\mathcal{P}_{ia}$ to refer to the set of *all identitiy-aware processes*. We let $\mathcal{I}(P_{ia}) := \mathcal{F}(P_{ia}) \cap \mathcal{I}$ return the instances in process P_{ia}.

We now define a transition relation among identity-aware processes. The idea is to compute instances that represent restricted names, rather than choosing them non-deterministically. For each restriction $a \in \mathcal{R}$, we introduce the function

$$\mathit{min}_a(P_{ia}) := (a,k) \quad \text{where} \quad k = \mathit{min}\,\{i \in \mathbb{N} \mid (a,i) \notin \mathcal{I}(P_{ia})\}\,.$$

It determines the *least instance that is not free in* P_{ia}. With this function, we can turn processes in standard form P_{sf} into identity-aware processes:

$$\mathit{ia}(S_1 \mid \ldots \mid S_n) := S_1 \mid \ldots \mid S_n \qquad \mathit{ia}(\nu a.P_{sf}) := \mathit{ia}(P_{sf})\{\mathit{min}_a(\mathit{ia}(P_{sf}))/a\}.$$

The *identity-aware transition relation* $\rightarrow_{ia} \subseteq \mathcal{P}_{ia} \times 2^{\mathcal{I}} \times \mathcal{P}_{ia}$ is now defined by

$$P_{ia} \xrightarrow{\mathcal{FI}(P_{ia},Q_{ia})}_{ia} Q_{ia} \qquad \text{iff} \quad P_{ia} \rightarrow Q_{sf} \quad \text{and} \quad Q_{ia} = \mathit{ia}(Q_{sf}).$$

Here, $\mathcal{FI}(P_{ia}, Q_{ia})$ is the set of fresh instances that are determined by $\mathit{ia}(Q_{sf})$. We usually write $P_{ia} \rightarrow_{ia} Q_{ia}$ and only mention $\mathcal{FI}(P_{ia}, Q_{ia})$ where it is needed. When we consider transition sequences, we form the union of the instances: $P_{ia} \rightarrow_{ia} Q_{ia} \rightarrow_{ia} R_{ia}$ yields $\mathcal{FI}(P_{ia}, R_{ia}) := \mathcal{FI}(P_{ia}, Q_{ia}) \cup \mathcal{FI}(Q_{ia}, R_{ia})$. We denote the set of all identity-aware processes that are reachable from P_{ia} via the identity-aware transition relation by $\mathit{Reach}_{ia}(P_{ia})$. The corresponding *identity-aware transition system* is $\mathcal{T}_{ia}(P_{ia}) := (\mathit{Reach}_{ia}(P_{ia})/_{\equiv}, \hookrightarrow_{ia}\;, \underline{P_{ia}})$, where $\underline{Q_{ia}} \hookrightarrow_{ia} \underline{R_{ia}}$ iff $Q_{ia} \rightarrow_{ia} R_{ia}$. This is indeed well-defined. The identity-aware transition system is bisimilar to the original one.

Proposition 1. $\mathcal{T}(P_{sf}) \approx \mathcal{T}_{ia}(\mathit{ia}(P_{sf}))$.

The bisimulation that relates the transition systems of P_{sf} and $P_{ia} = \mathit{ia}(P_{sf})$ is $\mathcal{B} \subseteq \mathit{Reach}_{ia}(P_{ia})/_{\equiv} \times \mathit{Reach}(P_{sf})/_{\equiv}$ defined by $\underline{Q_{ia}} \;\; \mathcal{B} \;\; \underline{\nu\, \mathcal{I}(Q_{ia}).Q_{ia}}$. We call an identity-aware process P_{ia} name-bounded if $\nu\mathcal{I}(P_{ia}).P_{ia}$ is. Equivalently, there is a finite set of instances that are used in any reachable process.

We elaborate on the shape of processes $Q_{ia} \in \mathit{Reach}_{ia}(P_{ia})$, making use of derivatives as introduced in [13]. Intuitively, process Q_{ia} consists of subterms of P_{ia} to which substitutions are applied. The idea of subterms is formalized by the notion of *derivatives* $\mathcal{D}(P)$. Function $\mathcal{D} : \mathcal{P} \rightarrow 2^{\mathcal{P}}$ returns the set of processes that can be found by removing prefixes, restrictions, and splitting up parallel compositions — in P and in its defining equations:

$$\begin{aligned} \mathcal{D}(\mathbf{0}) &:= \emptyset & \mathcal{D}(K\lfloor\tilde{a}\rfloor) &:= \{K\lfloor\tilde{a}\rfloor\} \cup \mathcal{D}(Q) \text{ if } K(\tilde{x}) := Q \\ \mathcal{D}(\pi.P) &:= \{\pi.P\} \cup \mathcal{D}(P) & \mathcal{D}(M+N) &:= \{M+N\} \cup \mathcal{D}(M) \cup \mathcal{D}(N) \\ \mathcal{D}(P \mid Q) &:= \mathcal{D}(P) \cup \mathcal{D}(Q) & \mathcal{D}(\nu a.P) &:= \mathcal{D}(P). \end{aligned}$$

Since processes rely on finitely many defining equations, the set remains finite. Derivatives do not keep track of what names are instantiated, nor what names are received in input prefixes. To correctly represent Q_{ia}, we map the names

Algorithm 1 Karp & Miller Tree Construction

```
procedure KM($P_{ia}$)
    $V := \{\mathsf{root}{:}P_{ia}\}$;   $\rightarrow_{KM} := \emptyset$;   $Work := \mathsf{root}{:}P_{ia}$;
    while $Work$ not empty do
        Pop $\mathsf{n}_1{:}L_1$ from $Work$;
        for all $L_1 \rightarrow_{ia} L_2$ up to $\equiv$ do
            if there is $\mathsf{n}{:}L \rightarrow^*_{KM} \mathsf{n}_1{:}L_1$ such that $L_2 \equiv L \mid L_{rem}$ and
                                    $\mathcal{I}(L_{rem}) \cap \mathcal{FI}(L, L_2) = \emptyset$ then
                $L_2 := L \mid L^{\omega}_{rem}$;
            let $\mathsf{n}_2$ be a new node
            $V := V \cup \{\mathsf{n}_2{:}L_2\}$;   $\rightarrow_{KM} := \rightarrow_{KM} \cup \{\mathsf{n}_1{:}L_1 \xrightarrow{\mathcal{FI}(L_1,L_2)}_{KM} \mathsf{n}_2{:}L_2\}$;
            $Work := Work \cdot (\mathsf{n}_2{:}L_2)$ provided $L_2$ does not occur from $\mathsf{root}$ to $\mathsf{n}_1$;
    return $(V, \rightarrow_{KM}, \mathsf{root}{:}P_{ia})$.
end procedure
```

occurring in derivatives to either free names in P_{ia} or to instances of restricted names. The corresponding set of substitutions is

$$\Sigma(P_{ia}) := \mathcal{F}(\mathcal{D}(P_{ia})) \to \mathcal{F}(P_{ia}) \cup (\mathcal{R}(P_{ia}) \times \mathbb{N}).$$

Lemma 1. *For every $Q_{ia} \in Reach_{ia}(P_{ia})$ there are $D_1, \dots, D_n \in \mathcal{D}(P_{ia})$ and $\sigma_1, \dots, \sigma_n \in \Sigma(P_{ia})$ so that $Q_{ia} \equiv D_1\sigma_1 \mid \dots \mid D_n\sigma_n$.*

Note that a name-bounded P_{ia} only uses a finite number of instances from $\mathcal{R}(P_{ia}) \times \mathbb{N}$, and therefore σ_1 to σ_n are taken from a finite subset of $\Sigma(P_{ia})$.

3.2 Karp and Miller Trees

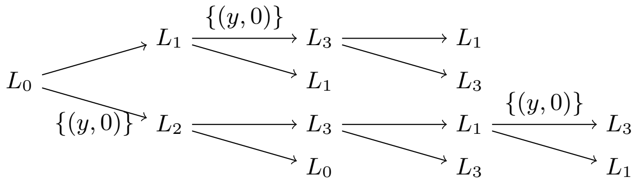
The Karp-Miller tree for a process P_{ia} is a rooted, directed tree. The nodes of the tree are labeled with either a single process reachable from P_{ia}, or a *limit process*, representing a set of processes summarizing the effect of repeating a reaction sequence. The root is labeled with P_{ia}.

The definition of limits is inspired by replication in π-calculus, and similar to [20]. A limit is either a sequential process S, a process of the form L^{ω}, or a parallel composition $L_1 \mid L_2$ of limit processes. Intuitively, limit S^{ω} represents an unbounded set of processes S^j for arbitrarily large j. We extend structural congruence to limit processes with the following rules:

$$S^{\omega} \mid S \equiv S^{\omega} \qquad S^{\omega} \mid S^{\omega} \equiv S^{\omega} \qquad (S^{\omega})^{\omega} \equiv S^{\omega} \qquad (L_1 \mid L_2)^{\omega} \equiv L_1^{\omega} \mid L_2^{\omega}$$

While decidability of structural congruence for processes with replication is problematic, it is not an issue here since ω distributes over parallel composition.

By associativity and commutativity of parallel composition and the above laws for limit processes, we can bring each limit process L into the standard form $L \equiv S_1^{k_1} \mid \dots \mid S_n^{k_n}$, where $S_i \not\equiv S_j$ for $i \neq j$ and $k_i \in \mathbb{N} \cup \{\omega\}$. Thus, we order the sequential processes into groups of structurally congruent ones



$$L_0 := K_1\lfloor(a,0)\rfloor \mid K_3 \qquad L_2 := K_1\lfloor(a,0)\rfloor \mid \overline{(y,0)} \mid (y,0).K_3$$

$$L_1 := K_1\lfloor(a,0)\rfloor^{\omega} \mid K_3 \qquad L_3 := K_1\lfloor(a,0)\rfloor^{\omega} \mid \overline{(y,0)} \mid (y,0).K_3.$$

Fig. 1. Algorithm 1 on $K_1\lfloor(a,0)\rfloor \mid K_3$ with $K_1(x) := K_1\lfloor x\rfloor^2$ and $K_3 := \nu y.(\overline{y} \mid y.K_3)$

and join them if we find ω. With this normal form, we can understand limit L as a multiset $L : \mathcal{S} \to \mathbb{N} \cup \{\omega\}$ that assigns $L(S_i) := k_i$. If process $S \in \mathcal{S}$ does not occur in the parallel composition above, it is assigned zero. We use $\mathit{Fin}(L) := \{S \in \mathcal{S} \mid 1 \leq L(S) \in \mathbb{N}\}$ for the sequential processes that occur finite in L, and $\mathit{Inf}(L) := \{S \in \mathcal{S} \mid L(S) = \omega\}$ for those that are ω. We also extend $\preceq$ to limits in the natural way, and note that it coincides with multiset inclusion.

Algorithm 1 shows a worklist algorithm to construct the Karp-Miller tree for a process P_{ia}. The construction starts with a root node labeled with P_{ia} and unrolls the reachability tree by executing enabled reactions. This means the edges $\mathsf{n}_1 : L_1 \to_{KM} \mathsf{n}_2 : L_2$ of the Karp-Miller tree mimic identity-aware reactions, and so are labeled with sets of fresh instances. Additionally, reaction sequences leading to repeating limits $L \preceq L \mid L_{rem}$ are accelerated, provided L_{rem} does not remember newly generated instances. We define $KM(P_{ia}) := \{L \mid \text{there is some node labeled with } L \text{ in the tree}\}$. Figure 1 gives the execution of the algorithm on an example process.

To understand the subtle acceleration condition in the if-statement, consider process $K_2\lfloor(b,0)\rfloor$ with $K_2(y) := \nu b.(K_2\lfloor b\rfloor^2)$:

$$K_2\lfloor(b,0)\rfloor \xrightarrow{\{(b,0)\}}_{ia} K_2\lfloor(b,0)\rfloor^2 \xrightarrow{\{(b,1)\}}_{ia} K_2\lfloor(b,1)\rfloor^2 \mid K_2\lfloor(b,0)\rfloor.$$

After the first transition, we find $K_2\lfloor(b,0)\rfloor \preceq K_2\lfloor(b,0)\rfloor^2$. But the intersection $\mathcal{I}(K_2\lfloor(b,0)\rfloor) \cap \{(b,0)\} \neq \emptyset$ forbids us to accelerate $K_2\lfloor(b,0)\rfloor^2$ to $K_2\lfloor(b,0)\rfloor^{\omega}$. That this is indeed correct can be seen in the third process. It is not true that arbitrarily many processes will get to know $(b,0)$.

The following lemmas encode the correctness of the construction.

Lemma 2 (Completeness). *For every $Q_{ia} \in \mathit{Reach}_{ia}(P_{ia})$ there is a limit $L \in KM(P_{ia})$ and an injective substitution $\sigma : \mathcal{I}(Q_{ia}) \to \mathcal{I}(L)$ so that $Q_{ia}\sigma \preceq L$.*

Lemma 3 (Soundness). *For every $L \in KM(P_{ia})$ and every $k \in \mathbb{N}$ there is $Q_{ia} \in \mathit{Reach}_{ia}(P_{ia})$ with $Q_{ia}(S) = L(S)$ for all $S \in \mathit{Fin}(L)$, $Q_{ia}(S) \geq k$ for all $S \in \mathit{Inf}(L)$, and $Q_{ia}(S) = 0$ otherwise.*

Note that the previous two lemmas do not assume P_{ia} to be name-bounded. This is important in proving Theorem 1(2) below.

Lemma 4 (Termination). *Algorithm 1 terminates with input $P_{ia} \in \mathcal{P}_{ia}$ if and only if P_{ia} is name-bounded.*

If the input process is not name-bounded, completeness in Lemma 2 implies that Algorithm 1 cannot terminate. The converse direction comes with two problems.

First, we have to guarantee that we find repeating limits $\mathsf{n}_1 : L_1 \rightarrow^*_{KM} \mathsf{n}_2 : L_2 \rightarrow_{ia} L$ with $L \equiv L_1 \mid L_{rem}$. The trick is to understand limits as multisets over a finite set, and then resort to the theory of well-quasi orderings. Lemma 1 shows that we can restrict ourselves to derivatives plus substitutions: $D\sigma$ and $(D\sigma)^{\omega}$. This means limits of P_{ia} are multisets over the finite set $\mathcal{D}(P_{ia}) \times \Sigma(P_{ia}) \times \{\varepsilon, \omega\}$. With this finiteness, multiset inclusion and hence $\preceq$ is a well-quasi ordering by Dickson's lemma. This guarantees repetitions.

Second, acceleration has a side condition which additionally requires $\mathcal{I}(L_{rem}) \cap \mathcal{FI}(L_1, L) = \emptyset$: the processes to be accelerated should not contain instances that were generated along the way. The following lemma shows that a failure of the side condition would contradict name boundedness.

Lemma 5. *Consider Algorithm 1 with $P_{ia} \in \mathcal{P}_{ia}$ as input. If the execution encounters $\mathsf{n}_1 : L_1 \rightarrow^*_{KM} \mathsf{n}_2 : L_2 \rightarrow_{ia} L$ with $L \equiv L_1 \mid L_{rem}$ and such that $\mathcal{I}(L_{rem}) \cap \mathcal{FI}(L_1, L) \neq \emptyset$, then P_{ia} is not name-bounded.*

Note that the lemma correctly predicts name unboundedness in the example $K_2\lfloor(b, 0)\rfloor$ above. Applied in contraposition, the lemma shows that for name-bounded processes the side condition always holds: if we find repeating elements L_1 and $L \equiv L_1 \mid L_{rem}$, then we are already sure that $\mathcal{I}(L_{rem}) \cap \mathcal{FI}(L_1, L) = \emptyset$.

Theorem 2. *Consider a name-bounded process $P \in \mathcal{P}$ with $P \equiv \nu\mathcal{I}(P_{ia}).P_{ia}$. Then $Reach(P){\downarrow} = \{\nu\mathcal{I}(L).L \mid L \in KM(P_{ia})\}{\downarrow}$ and the latter set is finite.*

Proof of Theorem 1(2). The Karp and Miller tree allows us to prove Theorem 1. To check whether $P \in \mathcal{P}$ is b-name-bounded, we execute Algorithm 1 until it either terminates or we find a limit with more than b instances. In the former case, we report that the process is b-name-bounded, in the latter that it is not.

Assume P is b-name-bounded. Then Algorithm 1 terminates by Lemma 4. By Lemma 3, it only constructs limits with up to b instances. Therefore, the second termination condition will not apply and we report P to be b-name-bounded. If P is not b-name-bounded, there is a reachable process with more than b instances. By completeness, Algorithm 1 will construct a limit with more than b instances. Then, we report P is not b-name-bounded. □

4 From Name-Bounded Processes to Petri Nets

4.1 Process Bounds

To give a reduction to Petri nets, we require a notion of "pumping" instances. We introduce *process bounds* as a pumping mechanism.

Consider a sequence $\rho = P_{ia} \rightarrow_{ia} \ldots \rightarrow_{ia} Q_{ia}$ and some $p \in \mathbb{N}$. We use $\mathcal{I}_{>p}(\rho)$ to denote the *instances that were known to more than p sequential processes at some moment during* ρ. As instances can be reused, $\mathcal{I}_{>p}(\rho)$ is a multiset. It is divided into $\mathcal{I}_{>p}(\rho) = \mathcal{I}^{active}_{>p}(\rho) + \mathcal{I}^{dead}_{>p}(\rho)$. The instances that once exceeded the bound and are still active in the final process Q_{ia} are given by $\mathcal{I}^{active}_{>p}(\rho)$. Note that $\mathcal{I}^{active}_{>p}(\rho)$ is indeed a set, not just a multiset. The instances that have been forgotten along the way is the multiset $\mathcal{I}^{dead}_{>p}(\rho)$.

The definitions are by induction on the length of the sequence, where the induction step is as follows:

$$\begin{aligned}\mathcal{I}^{dead}_{>p}(\rho.Q^1_{ia} \rightarrow_{ia} Q^2_{ia}) &:= \mathcal{I}^{dead}_{>p}(\rho) + \mathcal{I}_{dead}\\ \mathcal{I}^{active}_{>p}(\rho.Q^1_{ia} \rightarrow_{ia} Q^2_{ia}) &:= \mathcal{I}^{active}_{>p}(\rho) \setminus \mathcal{I}_{dead} \cup \mathcal{I}_{act}.\end{aligned}$$

The auxiliary set $\mathcal{I}_{dead}$ contains the instances that once exceeded the bound and were active up to Q^1_{ia}, but that are forgotten in the transition $Q^1_{ia} \rightarrow_{ia} Q^2_{ia}$. The set $\mathcal{I}_{act}$ contains the instances that occur in more than $p \in \mathbb{N}$ sequential processes of Q^2_{ia}. We use $|Q^2_{ia}|_{(a,i)}$ to denote the number of sequential processes in Q^2_{ia} that have (a, i) as instance:

$$\begin{aligned}\mathcal{I}_{dead} &:= \left\{(a,i) \in \mathcal{I}^{active}_{>p}(\rho) \mid (a,i) \notin \mathcal{I}(Q^2_{ia}) \text{ or } (a,i) \in \mathcal{FI}(Q^1_{ia}, Q^2_{ia})\right\}\\ \mathcal{I}_{act} &:= \left\{(a,i) \in \mathcal{I}(Q^2_{ia}) \mid |Q^2_{ia}|_{(a,i)} > p\right\}.\end{aligned}$$

Choose p to be the largest number of sequential processes that know an instance (a, i) in a limit of the Karp-Miller tree:

$$p := max \left\{ \, |L|_{(a,i)} \ \mid L \in KM(P_{ia}) \text{ and } (a,i) \notin \mathcal{I}(Inf(L))\right\}.$$

This p, called the *process bound*, has an interesting property. Consider a reaction sequence ρ from P_{ia} to Q_{ia}. If we replay it on the Karp-Miller tree, then an instance $(a, k) \in \mathcal{I}_{>p}(\rho)$ that is distributed to more than p sequential processes leads to S^ω. Since accelerated processes are never removed, the limit L that dominates Q_{ia} will contain S^ω. Thus, the limit not only contains the instances $\mathcal{I}(Q_{ia})$ but also the instances from $\mathcal{I}^{dead}_{>p}(\rho)$. Since there are reachable processes R_{ia} that correspond to L, we get a lower bound on the name bound of P_{ia}.

Lemma 6. *Let $P_{ia} \in \mathcal{P}_{ia}$ be b-name-bounded and let p be its process bound. For every ρ from P_{ia} to Q_{ia} we have $|\mathcal{I}^{dead}_{>p}(\rho)| + |\mathcal{I}(Q_{ia})| \leq b$.*

Phrased differently, instances which exceeded process bound p do not need to be reused by the identity-aware semantics to meet name bound b.

4.2 Petri Net Construction

Given a b-name-bounded process $P_{ia} \in \mathcal{P}_{ia}$ that has $p \in \mathbb{N}$ as a process bound, we construct a Petri net $\mathbf{N}\,(P_{ia}, b, p)$ that simulates the identity-aware semantics of P_{ia} in a strong sense: there is a bisimulation between the transition systems of

$\mathbf{N}(P_{ia}, b, p)$ and P_{ia}. We use the standard notion of place-transition Petri nets of the form (S, T, W, M_0) with the standard firing semantics.

We describe the construction of $\mathbf{N}(P_{ia}, b, p)$ as a composition of two parts:

$$\mathbf{N}(P_{ia}, b, p) := \mathbf{Syn}(\mathbf{Ctrl}(P_{ia}, b)) \times \mathbf{Ref}(P_{ia}, b, p).$$

Petri net $\mathbf{N}(P_{ia}, b, p)$ maintains a finite set of instances $\{(a, 0), \ldots, (a, b-1)\}$ to represent each restriction $a \in \mathcal{R}(P_{ia})$. Petri net $\mathbf{Ref}(P_{ia}, b, p)$ implements reference counters for these instances in order to correctly allocate them when restrictions are encountered. The control flow of P_{ia} is captured by $\mathbf{Ctrl}(P_{ia}, b)$. While this net only models the consumption of prefixes, the synchronisation operation $\mathbf{Syn}(\cdot)$ joins send and receive transitions with complementary labels. Finally, the × operator co-ordinates the distribution and generation of instances between $\mathbf{Syn}(\mathbf{Ctrl}(P_{ia}, b))$ and $\mathbf{Ref}(P_{ia}, b, p)$. We turn to the details.

Petri net $\mathbf{Ctrl}(P_{ia}, b)$ maintains a place for each possible sequential process. These places count the number of occurrences of the corresponding process. Since there are finitely many restrictions, each with finitely many instances, as well as finitely many derivatives, the number of places in $\mathbf{Ctrl}(P_{ia}, b)$ is finite.

The transitions are derived from the places. Consider a receive

$$S \equiv x(y).\nu\tilde{a}.Q_{ia} + \ldots \quad \text{with} \quad \nu\tilde{a} = \nu a_1 \ldots \nu a_m \quad \text{and} \quad Q_{ia} \equiv S_1^{o_1} \mid \ldots \mid S_n^{o_n}.$$

The receive operation is implemented by a set of transitions, one transition t for each z in $\mathcal{F}(P_{ia}) \cup (\mathcal{R}(P_{ia}) \times [0, b-1])$ that we can receive for y and each m-tuple of instances (a_1, k_1) to (a_m, k_m) that we may use to represent a_1 to a_m. The post set of t is the set of sequential processes in $Q_{ia}\sigma$. In the above case, this is $S_1\sigma$ to $S_n\sigma$. Substitution σ maps y to z and a_i to (a_i, k_i) for $i \in [1, m]$. The weight of the arc from t to $S_i\sigma$ is given by $Q_{ia}\sigma(S_i\sigma) = o_i$, the number of occurrences of $S_i\sigma$ in $Q_{ia}\sigma$. Additionally, the transition is labeled with information used to synchronize with the reference counters.

The synchronisation operation $\mathbf{Syn}(\mathbf{Ctrl}(P_{ia}, b))$ joins transitions with complementary send and receive labels. Finally, the × connects the synchronized control flow net with the reference counters.

To limit the number of instances to $b \in \mathbb{N}$, we reinstantiate an instance (a, k) that was present in Q_{ia}^1 in a later process Q_{ia}^2, provided it has been forgotten on the path $Q_{ia}^1 \rightarrow_{ia}^{+} Q_{ia}^2$. The Petri net $\mathbf{Ref}(P_{ia}, b, p)$ maintains reference counters that track the number of sequential processes S with $(a, k) \in \mathcal{I}(S)$. An instance (a, k) can be reinstantiated in case the reference counter is zero.

To check that a reference counter is zero, we use the process bound p. (Recall that Petri nets cannot implement zero-tests.) Consider a sequence ρ from P_{ia} to Q_{ia}. An instance (a, k) in ρ that, at some moment, is distributed to more than p sequential processes and that has been forgotten in Q_{ia} is a member of $\mathcal{I}_{>p}^{dead}(\rho)$. Lemma 6 ensures this instance need not be reinstantiated. The name bound is high enough so that there is another instance that is currently not in use and that can represent the restriction. As a result, once a reference count goes beyond p, the name is never recycled. The net $\mathbf{Ref}(P_{ia}, b, p)$ has three places for each instance (a, k): a place $(a, k)^{\omega}$ denoting this name cannot be recycled, a place

(a, k) keeping the current reference count (provided the reference count has never exceeded p), and a complementary place $\overline{(a, k)}$ that ensures the sum of tokens in (a, k) and $\overline{(a, k)}$ together is p while the name is bounded. We implement the zero-test by checking $\overline{(a, k)}$ has p tokens.

Now, a step of the process is simulated by a sequence of steps of the Petri net that update the tokens in $\mathbf{Ctrl}\,(P_{ia}, b)$, but at the same time, instantiate required names and update the reference counts.

4.3 Bisimilarity

To obtain a clean bisimulation between a name-bounded process and its Petri net semantics, we use an idea from [15]: we define distinguished *stable* markings that will actually correspond to processes, as opposed to intermediary markings that occur when we amend reference counters and allocate instances.

Consider $\mathbf{N}\,(P_{ia}, b, p) = (S, T, W, M_0)$. The set of places in $\mathbf{Syn}(\mathbf{Ctrl}\,(P_{ia}, b))$ is the disjoint union $S_{proc} \uplus S_{inter} \subseteq S$ where S_{proc} contains the process places of $\mathbf{Ctrl}\,(P_{ia}, b)$ while S_{inter} contains intermediary places that we added to allocate instances and update reference counters. Now, a marking $M \in \mathbb{N}^S$ is called *stable* if $M(s) = 0$ for all $s \in S_{inter}$. We use $\mathcal{R}_{stbl}(\mathbf{N}\,(P_{ia}, b, p))$ to denote the set of stable markings that are reachable in $\mathbf{N}\,(P_{ia}, b, p)$.

We are interested in transition sequences of $\mathbf{N}\,(P_{ia}, b, p)$ that correspond to *one* communication or identifier-call, rather than interleavings of reactions. Formally, a transition sequence $M^1 \xrightarrow{t_1 \dots t_n} M^2$ between stable markings is called *race-free* if there is a single transition t of $\mathbf{Syn}(\mathbf{Ctrl}\,(P_{ia}, b))$ that is unfolded into $t_1 \dots t_n$ in the composition $\mathbf{N}\,(P_{ia}, b, p)$. We write $M^1 \Rightarrow M^2$ if there is a race-free transition sequence between the two markings. With this, the *stable transition system* is $\mathcal{T}_{stbl}(\mathbf{N}\,(P_{ia}, b, p)) := (\mathcal{R}_{stbl}(\mathbf{N}\,(P_{ia}, b, p)), \Rightarrow, M_0)$.

To define a bisimulation relation, we have to decide which instances to mark as unbounded in the marking of $\mathbf{N}\,(P_{ia}, b, p)$. Unfortunately, a single process Q_{ia} does not carry enough information to define the related markings. We need the full transition sequence $\rho = P_{ia} \rightarrow_{ia} \dots \rightarrow_{ia} Q_{ia}$ that leads to Q_{ia}. We therefore extend the identity-aware transition system $\mathcal{T}_{ia}(P_{ia})$ to a *history-preserving identity-aware transition system* $\mathcal{T}^h_{ia}(P_{ia})$ in which states are such sequences ρ from P_{ia} to Q_{ia}. Clearly, the identity-aware transition system and its history-preserving variant are bisimilar. The history-preserving transition system carries enough information to establish a bisimulation result.

Lemma 7. $\mathcal{T}_{ia}(P_{ia}) \approx \mathcal{T}^h_{ia}(P_{ia}) \approx \mathcal{T}_{stbl}(\mathbf{N}\,(P_{ia}, b, p))$.

Assume P_{ia} has name bound $b \in \mathbb{N}$ and process bound $p \in \mathbb{N}$. Then $\mathcal{B}$ relates transition sequence ρ from P_{ia} to Q_{ia} with marking $M = M_{ctr} + M_{ref}$ if the following holds.

For the control-flow marking, we require that there is an injective substitution $\sigma : \mathcal{I}(Q_{ia}) \rightarrow \mathcal{I}(supp(M_{ctr}))$ so that $M_{ctr} \equiv Q_{ia}\sigma$.

For marking M_{ref} of the reference counter, consider $(a, i) \in \mathcal{I}^{active}_{>p}(\rho)$. We require $M_{ref}(\ [(a, i)\sigma]^{\omega}\ ) = 1$. For an instance $(a, i) \in \mathcal{I}(Q_{ia}) \backslash \mathcal{I}^{active}_{>p}(\rho)$, we need

$M_{ref}(\ (a,i)\sigma\ ) = |Q_{ia}|_{(a,i)}$. With Lemma 6, there are $b - |\mathcal{I}(Q_{ia})| \geq |\mathcal{I}^{dead}_{>p}(\rho)|$ instances (a,k) outside the range of σ. We have $M_{ref}(\ (a,k)^{\omega}\ ) = 1$ for $|\mathcal{I}^{dead}_{>p}(\rho)|$ such instances. All other instances carry p tokens on the complement place.

The above constraints describe a marking that is partial in that, for each instance, the token count of only one place (a,k), $\overline{(a,k)}$, or $(a,k)^{\omega}$ is given. The tokens for the remaining places are uniquely determined by the following invariants. Places (a,i) and $\overline{(a,i)}$ are complements with bound p. Also places $\{(a,i), \overline{(a,i)}\}$ and $(a,i)^{\omega}$ are complements. This means, if $(a,i)^{\omega}$ carries a token, the other two are empty and vice versa. Place $(a,i)^{\omega}$ is safe. We can now state our second main result.

Theorem 3. *Let $P \equiv \nu\mathcal{I}(P_{ia}).P_{ia}$ be name-bounded by $b \in \mathbb{N}$ and let $p \in \mathbb{N}$ be the process bound. Then $\mathcal{T}(P) \approx \mathcal{T}_{stbl}(\mathbf{N}\,(P_{ia}, b, p))$.*

Proof. $\mathcal{T}(P) \approx \mathcal{T}_{ia}(P_{ia}) \approx \mathcal{T}^{h}_{ia}(P_{ia}) \approx \mathcal{T}_{stbl}(\mathbf{N}\,(P_{ia}, b, p))$, with Proposition 1 and Lemma 7. □

Corollary 1. *Reachability is decidable for name-bounded processes.*

5 Ackermann Lower Bound

We give a polynomial-time reduction from Turing machines operating on a tape of non-primitive recursive size to name-bounded processes. As a consequence, verification problems for name-bounded systems have non-primitive recursive complexity.

We give the construction of process $P(TM)$ for Turing machine TM. The behavior of $P(TM)$ is divided into three stages. In the first stage, $P(TM)$ generates up-to $A(n)$ parallel processes $W\lfloor p, Q\rfloor$ that are waiting. Here, Q is the set of states in the Turing machine that we deliberately understand as channels. Moreover, there is a single process $\nu c.\nu r.G\lfloor p, c, r, q_0\rfloor$:

$$P(TM) \rightarrow^{*} W\lfloor p, Q\rfloor^{A(n)} \mid \nu c.\nu r.G\lfloor p, c, r, q_0\rfloor.$$

For this generation phase, we rely on a result from Petri net theory [11]. There is a sequence of Petri nets $(N_i)_{i\in\mathbb{N}}$ where the size grows linearly and that produce up to $(A(i))_{i\in\mathbb{N}}$ tokens on a designated place. Such a Petri net N_i can be turned into a restriction-free process creating $A(i)$ copies of $W\lfloor p, Q\rfloor$.

In the second stage, process $G\lfloor p, c, r, q_0\rfloor$ aligns the waiting $W\lfloor p, Q\rfloor$ into a list of processes $C_0\lfloor l, c, r, Q\rfloor$ representing cells in the Turing tape with content 0, input channel c, and pointers l and r to the input channels of their left and right neighbor. The process $G\lfloor p, c, r, q_0\rfloor$ recursively converts $W\lfloor p, Q\rfloor$ to cells, and can non-deterministically decide that the current cell with input channel c is the last one in the list. In this case it sends the initial state q_0 of TM to the cell, which starts the simulation of the Turing machine.

The simulation of the Turing machine is the third stage in the behavior of $P(TM)$. Each process $C_i(l, c, r, Q)$, with $i \in \{0, 1\}$ as current content, waits to receive the head pointer and the current state of TM. On receiving the current state, the process executes one transition of the machine and updates its content, while sending the successor state to its left or right neighbor, based on the transition.

Theorem 4. *(1) Reachability and coverability are non-primitive recursive for name-bounded processes. (2) There is no primitive recursive translation of name-bounded processes into Petri nets that preserves coverability. (3) There is no primitive recursive bound on the size of name and process bounds.*

Theorem 4 shows the Karp-Miller procedure is asymptotically optimal.

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. K. Bansal, E. Koskinen, T. Wies, and D. Zufferey. Structural counter abstraction. In *TACAS*, LNCS 7795, pages 62–77. Springer, 2013.
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. N. Busi and G. Zavattaro. Deciding reachability problems in Turing-complete fragments of Mobile Ambients. *Math. Struct. Comp. Sci.*, 19(6):1223–1263, 2009.
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. A. Finkel and J. Goubault-Larrecq. The theory of WSTS: The case of complete WSTS. In *Petri Nets*, LNCS 7347, pages 3–31. Springer, 2012.
8. C. He. The decidability of the reachability problem for CCS! In *CONCUR*, LNCS 6901, pages 373–388. Springer, 2011.
9. R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.
10. R. J. Lipton. The reachability problem requires exponential space. Technical report, Yale University, Department of Computer Science, 1976.
11. E. W. Mayr and A. R. Meyer. The complexity of the finite containment problem for Petri nets. *JACM*, 28(3):561–576, 1981.
12. R. Meyer. On boundedness in depth in the π-calculus. In *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 and R. Gorrieri. On the relationship between π-calculus and finite p/t Petri nets. In *CONCUR*, LNCS 5170, pages 463–480. Springer, 2009.
15. R. Meyer, V. Khomenko, and R. Hüchting. A polynomial translation of π-calculus (FCP) to safe Petri nets. In *CONCUR*, LNCS 7454, pages 440–455. Springer, 2012.
16. R. Milner. *Communicating and Mobile Systems: the π-Calculus.* CUP, 1999.
17. U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In *CONCUR*, LNCS 962, pages 42–56. Springer, 1995.
18. C. Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comp. Sci.*, 6(2):223–231, 1978.
19. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes.* CUP, 2001.
20. T. Wies, D. Zufferey, and T. A. Henzinger. Forward analysis of depth-bounded processes. In *FOSSACS*, LNCS 6014, pages 94–108. Springer, 2010.