On Depth-bounded Message Passing Systems

Thomas A. Henzinger, Thomas Wies, and Damien Zufferey

EPFL School of Computer and Communication Sciences, Switzerland

Abstract. We explore the border between decidability and undecidability of verification problems related to message passing systems that admit unbounded creation of threads and name mobility. Inspired by use cases in real-life programs we introduce the notion of depth-bounded message passing systems. A configuration of a message passing system can be represented as a graph. In a depth-bounded system the length of the longest acyclic path in each reachable configuration is bounded by a constant. While the general reachability problem for depth-bounded systems is undecidable, we prove that control reachability is decidable. In our decidability proof we show that depth-bounded systems are well-structured transition systems to which a forward algorithm for the covering problem can be applied.

1 Introduction

We study the boundary between decidability and undecidability of verification problems related to message passing systems. In particular, we are interested in systems that use the actor model [2,3,9,23,40] for asynchronous message passing. Our motivation stems from the increased practical importance of actors. The actor model is now the preferred or only available concurrency mechanism in various modern programming languages, such as SCALA [37] and ERLANG [7], and is becoming popular among practicing programmers. For instance, the TWITTER microblogging service now uses SCALA actors [38].

In the actor model the only computation entity is the *actor*. An actor can receive messages from, respectively send messages to other actors. The sent messages are stored in an unordered buffer that is owned by the receiving actor. Each time an actor processes a message in its buffer it can locally decide to

- create finitely many new actors,
- send finitely many messages to actors that it knows
- and change its behavior as to how the next message is processed.

Here *knowing* another actor means that the recipient of a message was either created by the sending actor or its name was previously sent to the sending actor. In this paper we consider the more generic setting of the asynchronous π-calculus [8,24] where one speaks about *threads* communicating via *channels* rather than actors with buffers. However, for increased vividness we will for now stay in the terminology of actors.

One can think of the configuration of an actor system as a graph [25]. The vertices in the graph correspond to actors and messages. Edges between vertices indicate whether an actor knows the name of another actor, whether a message is in the buffer of an actor, and whether a message carries the name of an actor. We refer to these graphs as *communication topologies.* In principle, the communication topologies can encode arbitrary data structures (e.g., the tape of a Turing machine) because their size is not bounded and edges can dynamically change during execution. In general, most problems related to verification of such systems are therefore undecidable [30].

In practice, programming languages that support actors incorporate the actor model in the form of an extension to a sequential core language [7] or a library for a core language that provides other concurrency mechanisms [22,41]. Given this core language, programmers tend to use actors in a rather restrictive form, despite of their intrinsic computational power. Complex data structures are encoded in the local state of the individual actors rather than the global communication topology. The communication topologies that are reachable in the executions of real programs therefore have a rather simple shape. This raises the question whether one can define a *behavioral class* of actor systems by restricting the shape of the reachable communication topologies such that certain verification problems become decidable. Yet, this class should still cover a significant portion of the use cases that programmers actually care about. In this paper we identify such a behavioral class of message passing systems.

Depth-bounded systems. Using the graph-theoretic notion of *tree-depth* [36] we define the new class of *depth-bounded message passing systems.* Formally, the tree-depth of a graph is the height of a minimal tree whose closure contains the graph. In a depth-bounded system the tree-depth of all reachable communication topologies is bounded by a constant. Intuitively, this condition bounds the length of the maximal acyclic path in each reachable communication topology. Depth-bounded systems still allow name mobility via messages and unbounded creation of both actors and messages. This class therefore covers many interesting use cases of message passing concurrency such as client-server and consumer-producer communication with an unbounded number of clients/producers, and master-worker load balancing.

While the general reachability problem for depth-bounded systems is undecidable, this class is still an interesting target for automated verification. The main technical contribution of this paper is a proof of decidability of the control reachability problem for depth-bounded systems. Intuitively, control reachability concerns the verification of safety properties that are locally observable by a single actor. This problem subsumes many interesting verification problems that occur in practice. In our decidability proof we apply a special case of Kruskal's tree theorem [19,28] to show that depth-bounded systems induce well-structured transition systems (WSTS) [1,18]. We then show that the covering problem for these systems can be decided using the expand, enlarge, and check algorithm for WSTSs [20]. Interestingly, unlike the standard backward algorithms for the covering problem of WSTSs, this forward algorithm terminates even if the bound of the system is not known a priori.

2 Motivating Example

We now present a typical example of a depth-bounded system. Our example is a publish/subscribe service that provides an interface between publishers of content (organized into finitely many categories) and subscribers to which this content is distributed (depending on the category they are enlisted to). Figure 1 shows an actor-based implementation of this service in SCALA-like pseudo code.

SCALA actors are subclasses of the `Actor` trait. The behavior of an actor is specified by the method `act`. This method is called implicitly when the actor is started. Receiving a message is done by calling `react`. The method `react` implicitly stores a reference to the sender of the received message in the field

```
sealed abstract class Category
case object Cat1 extends Category
...
case object CatN extends Category
case object List
case class Categories(cats: Set[Category])
...
class Server extends Actor {
  def loop(enl: Map[Category,Set[Actor]]){
    val cats = Set(Cat1,...,CatN)
    react {
      case List => {
        reply(Categories(cats))
        react {
          case Subscribe(c) =>
            loop(enl + c -> (enl(c) + sender))
        }
      }
      case Unsubscribe(c) =>
        loop(enl(c) + c -> (enl(c) - sender))
      case Publish => {
        reply(Who)
        react {
          case Credential =>
            if (*) {
              reply(Categories(cats))
              react {
                case Content(c) =>
                  enl(c).forall( _ ! Content(c))
                  loop(enl)
              }
            } else {
              reply(Deny)
              loop(enl)
            }
        }
      }
    }
  }
  override def act() = loop({_ => EmptySet})
}
```

```
class Subscriber(server: Actor) extends Actor {
  def loop(cat: Category): Unit = {
    if (*) {
      react {
        case Content(c) =>
          if (c != cat) error("...")
          ...
      }
    } else {
      server ! Unsubscribe(cat)
      exit('normal)
    }
  }

  override def act(): Unit = {
    server ! List
    react {
      case Categories(cats) =>
        val cat = cats.choose
        loop(cat)
    }
  }
}

class Publisher(server: Actor) extends Actor {
  override def act(): Unit = {
    server ! Publish
    react {
      case Who =>
        reply(Credential)
        react {
          case Categories(cats) =>
            val c = cats.choose
            reply(Content(c))
            if (*) act() else exit('normal)
          case Deny => exit('badCredential)
        }
    }
  }
}
```

Fig. 1. SCALA pseudo code for the publish/subscribe service

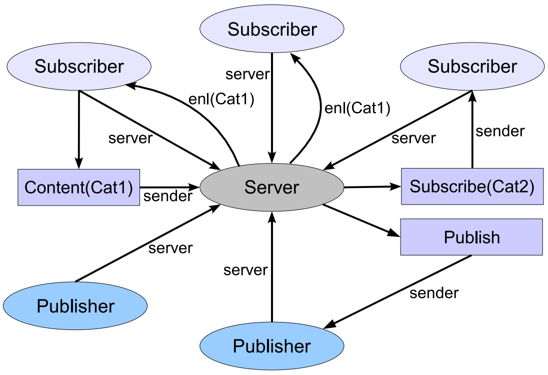

Fig. 2. A reachable configuration of the publish/subscribe service

`sender`. To send a message, we consider methods `!` and `reply`. The statement `a ! m` sends a message `m` to the recipient `a`; `reply(m)` is a shorthand for `sender ! m`.

The implementation of the service uses a client/server architecture. There are three kinds of actors: the server, the subscriber clients, and the publisher clients. In addition we assume an actor that models an environment which nondeterministically generates new subscribers and publishers.

The system works as follows. Subscribers first request a list of available categories by sending a `List` message to the server. Upon reception of `List`, the server sends back the list of categories. The subscriber then chooses one category and enlists itself by sending the appropriate `Subscribe` message to the server. For each category the server keeps track of the set of enlisted subscribers. Whenever it receives a `Subscribe` message the server adds the sender to the corresponding set. After subscription with the server the subscriber waits for incoming `content` messages or may choose to unsubscribe by sending an `Unsubscribe` message.

The protocol for the publishers is similar. A publisher initiates the communication with the server by sending a `Publish` message. The server then asks the publisher for its credentials and may deny the publisher's request if the credentials are not trustworthy. If however the server accepts the credentials then it asks the publisher for the category where it intends to publish. The server then forwards the received content to all subscribers of the corresponding category.

Figure 2 illustrates a reachable configuration of the publish/subscribe service. Notice the star-like shape with the server in the center. A tree of minimal height that overlays this configuration is rooted at the server. This tree has height 3 and, thus, the tree-depth of this configuration is 3. In fact the tree-depth of any reachable configuration of this system is bounded by 3. Therefore, the system is depth-bounded. Note, however, that the size of the reachable configurations is not bounded. Both the number of subscribers and publishers, as well as the number of messages in the buffers of subscribers and the server can grow arbitrarily large.

An interesting property of our service that we would like to verify is whether subscribers only receive content messages of categories they are enlisted to. This property is equivalent to the question whether the method `error` in the class `Subscriber` is ever called. The result presented in this paper implies that checking such properties is decidable for depth-bounded message passing systems.

3 Preliminaries

We first fix the syntax and semantics of our version of the asynchronous π-calculus and briefly introduce well-structured transition systems.

3.1 Asynchronous π-calculus

We consider systems of recursive equations in the polyadic asynchronous π-calculus that have a specific normal form due to Amadio and Meyssonnier [5].

Assume a countable infinite set of names with typical elements x, y and a countable infinite set of process identifiers with typical elements A, B. We assume that each name and identifier has an associated *arity* in $\mathbb{N}$. We denote by $\boldsymbol{x}$ a (possibly empty) vector over names and denote by $[\boldsymbol{x}/\boldsymbol{y}]$ a substitution on names.

Process terms P are composed of the unit process 0, parameterized process identifiers $A(\boldsymbol{x})$, and the standard operations of message creation $\overline{x}(\boldsymbol{y})$, input prefix $x(\boldsymbol{y}).P$, parallel composition $P \mid Q$, and name generation (νx). Hereby, the parameter vectors must respect the arities of names and identifiers. We call the terms of the form $\overline{x}(\boldsymbol{x})$ *messages* and the terms of the form $A(\boldsymbol{x})$ *threads*. We write Π in order to denote indexed parallel composition and we write $(\nu \boldsymbol{x})$ for $(\nu x_1) \dots (\nu x_n)$ where $\boldsymbol{x} = x_1, \dots, x_n$. An occurrence of a name x in a process term P is called *free* if it is not below a (νx) or an input prefix $y(x)$. We denote by $\mathsf{fn}(P)$ the set of all free occurring names in P. We say that P is *closed* if $\mathsf{fn}(P) = \emptyset$. We denote by $P \equiv Q$ the usual congruence relation on process terms, i.e., P is syntactically equal to Q up to renaming and reordering of generated names, elimination of units, and associativity and commutativity of parallel composition.

A *configuration* is a closed process term of the following normalized form

$$(\nu \boldsymbol{x})(\prod_{i \in I} \overline{x_i}(\boldsymbol{x}_i) \mid \prod_{j \in J} A_j(\boldsymbol{x}_j))$$

where $\boldsymbol{x}$ only contains names that actually occur. Note that any process term can be rewritten into a congruent configuration.

A *process* $\mathcal{P}$ is a pair $(I, \mathcal{E})$ where I is an *initial* configuration and $\mathcal{E}$ is a finite set of parametric equations $A(\boldsymbol{x}) = P$ such that (1) every process identifier in P is defined by exactly one equation in $\mathcal{E}$ and (2) $\mathsf{fn}(P) \subseteq \{\boldsymbol{x}\}$. We assume that all equations in $\mathcal{E}$ have the following normal form:

$$A(\boldsymbol{x}) = x(\boldsymbol{x}').(\nu \boldsymbol{x}'')(\prod_{i \in I} \overline{x}_i(\boldsymbol{x}_i) \mid \prod_{j \in J} A_j(\boldsymbol{x}_j)) \tag{1}$$

Actor systems as π-calculus processes. We can encode actor systems using π-calculus processes. A configuration of an actor system is similar to a process configuration: it consists of actors with their associated behavior and messages that are stored in the buffers of these actors. We can therefore easily encode actors using threads. Unlike in the general π-calculus where a thread can receive messages from any channel whose name it knows, actors can only receive messages from their private buffers, i.e., every message has a unique receiver. In our encoding an actor is therefore a thread of the form $A(\boldsymbol{x}_I; \boldsymbol{x}_O)$ whose parameters are divided into *input/output parameters* $\boldsymbol{x}_I$ and *output parameters* $\boldsymbol{x}_O$. The names in $\boldsymbol{x}_I$ can be used for both sending and receiving messages while the names in $\boldsymbol{x}_O$ can only be used for sending messages. For each pair of actors $A(\boldsymbol{x}_I; \boldsymbol{x}_O)$ and $B(\boldsymbol{y}_I; \boldsymbol{y}_O)$ in a configuration, the names $\boldsymbol{x}_I$ and $\boldsymbol{y}_I$ are disjoint, i.e., the i/o parameters encode the private buffers of actors. We call the above restrictions on configurations the *unique receiver condition* [4]. The preservation of the unique receiver condition is guaranteed by the *actor equations* that define the possible behaviors of actors. Actor equations are of the following form:

$$\begin{aligned}
A(\boldsymbol{x}_I; \boldsymbol{x}_O) &= \sum_{j \in J} x_j(\boldsymbol{x}_j).B_j(\boldsymbol{x}_I; \boldsymbol{x}_{jO}) \\
&\qquad \text{where for all } j \in J, x_j \in \{\boldsymbol{x}_I\} \\
B(\boldsymbol{x}_I; \boldsymbol{x}_O) &= (\nu \boldsymbol{y}_I)(\prod_{j \in J} \overline{x_j}(\boldsymbol{x}_j) \mid \prod_{k \in K} A_k(\boldsymbol{y}_{kI}, \boldsymbol{y}_{kO}) \mid A(\boldsymbol{x}_I; \boldsymbol{x}'_O)) \\
&\qquad \text{where } \{\boldsymbol{y}_I\} = \bigcup_{k \in K} \{\boldsymbol{y}_{kI}\} \text{ and for all } k, k' \in K, \{\boldsymbol{y}_{kI}\} \cap \{\boldsymbol{y}_{k'I}\} = \emptyset
\end{aligned}$$

We divide the process identifiers that are used to describe actors into two categories: receiving states $A(\boldsymbol{x}_I; \boldsymbol{x}_O)$ and dispatching states $B(\boldsymbol{x}_I; \boldsymbol{x}_O)$. The equations for the receiving states specify how an actor processes a received message. Hereby, the *external choice* operator Σ is used to differentiate how a received message is dispatched, depending on the kind of the message. The equations for the dispatching states specify the actual behaviors of the actors: upon reception of a message an actor can send finitely many new messages $\overline{x_j}(\boldsymbol{x}_j)$ to other known actors, create finitely many new actors $A_k(\boldsymbol{y}_{kI}, \boldsymbol{y}_{kO})$, and continue its own computation in a new receiving state $A(\boldsymbol{x}_I; \boldsymbol{x}'_O)$. Again we assume that the free names occuring on the right-hand sides of actor equations are contained in the list of parameters on the corresponding left-hand sides.

Note that actor equations can be normalized to equations of the form (1) (potentially by introducing additional process identifiers and equations). In particular, the external choice operator can be eliminated (see, e.g., [35]).

An *actor system* is a process whose initial configuration obeys the unique receiver condition and whose equations are the normalizations of actor equations.

Operational semantics. Given a process $\mathcal{P} = (I, \mathcal{E})$, we define a transition relation $\rightarrow_{\mathcal{E}}$ on configurations that captures the usual π-calculus reduction rules as follows. Let P and Q be configurations then we have $P \rightarrow_{\mathcal{E}} Q$ if and only if the following conditions hold:

1. $P \equiv (\nu \boldsymbol{u})(A(\boldsymbol{v}) \mid \overline{w}(\boldsymbol{w}) \mid P')$,
2. the defining equation of A in $\mathcal{E}$ is of the form $A(\boldsymbol{x}) = x(\boldsymbol{x}').(\nu \boldsymbol{x}'')(Q')$,
3. $\sigma = [\boldsymbol{v}/\boldsymbol{x}, \boldsymbol{w}/\boldsymbol{x}', \boldsymbol{y}/\boldsymbol{x}'']$ where $\boldsymbol{y}$ are fresh names

4. $\sigma(x) = w$
5. $Q \equiv (\nu \boldsymbol{u}, \boldsymbol{y})(P' \mid \sigma(Q'))$.

We denote by $\rightarrow_{\mathcal{E}}^{*}$ the reflexive transitive closure of the relation $\rightarrow_{\mathcal{E}}$. We say that a configuration P is *reachable* in process $\mathcal{P}$ if and only if $I \rightarrow_{\mathcal{E}}^{*} P$. Finally, we denote by $\mathit{Reach}(\mathcal{P})$ the set of all reachable configurations of process $\mathcal{P}$.

Reachability and control reachability. We consider two problems related to the verification of actor systems. The reachability problem and the more restrictive control reachability problem.

Definition 1 (Reachability problem). *Given a process $\mathcal{P}$ and a configuration P. The reachability problem is to decide whether P is reachable in $\mathcal{P}$.*

Definition 2 (Control reachability problem). *Given a process $(I, \mathcal{E})$ and a process identifier A. The control reachability problem is to decide whether there exists a configuration P of the form $P \equiv (\nu \boldsymbol{x})(A(\boldsymbol{y}) \mid Q)$ such that P is reachable in $\mathcal{P}$.*

3.2 Well-Structured Transition Systems (WSTS)

We briefly recall the relevant theory of well-structured transition systems [1,15, 16,18].

Well-quasi-ordering. A pair $(X, \leq)$ of a set X and a binary relation $\leq$ on X is called *well-quasi-ordered set* (wqo) if and only if (1) $\leq$ is a quasi-ordering (i.e., reflexive and transitive) and (2) any infinite sequence $x_0, x_1, x_2, \ldots$ of elements from X contains an increasing pair $x_i \leq x_j$ with $i < j$.

Let $(X, \leq)$ be a well-quasi-ordered set. A set $I \subseteq X$ is called *upward-closed* if for any pair x, y such that $y \geq x$, $x \in I$ implies $y \in I$. Similarly, I is called *downward-closed* if for any pair x, y such that $y \leq x$, $x \in I$ implies $y \in I$. The upward-closure of $Y \subseteq X$ is defined as $\uparrow Y = \{\, x \mid \exists y \in Y.\, x \geq y \,\}$. Correspondingly, we denote by $\downarrow Y$ the downward-closure of Y.

Well-structured transition system. A *well-structured transition systems* (WSTS) is a transition system $T = (S, s_0, \rightarrow, \leq)$ where S is a set of configurations, $s_0 \in S$ an initial configuration, $\rightarrow \subseteq S \times S$ a transition relation, and $\leq \subseteq S \times S$ a relation satisfying the following two conditions: (well-quasi-ordering) $\leq$ is a well-quasi-ordering on S; and (compatibility) $\leq$ is upward compatible with respect to $\rightarrow$, i.e., for all s_1, s_2, t_1 such that $s_1 \leq t_1$ and $s_1 \rightarrow s_2$, there exists t_2 such that $t_1 \rightarrow^{*} t_2$ and $s_2 \leq t_2$.

Given a well-structured transition system $(S, s_0, \rightarrow, \leq)$ we define the function *Pred* that maps a set of configurations $C \subseteq S$ to the set of its direct predecessors, and the function *Post* that maps C to its direct successors

$$\mathit{Pred}(C) \overset{\text{def}}{=} \{\, s \in S \mid \exists s' \in C.\, s \rightarrow s' \,\}$$
$$\mathit{Post}(C) \overset{\text{def}}{=} \{\, s' \in S \mid \exists s \in C.\, s \rightarrow s' \,\} \quad .$$

Definition 3 (Covering Problem). *Given a WSTS $(S, s_0, \to, \leq)$ and a configuration $t \in S$, the* covering problem *is to decide whether there exists a configuration $t' \in S$ such that $s_0 \to^* t'$ and $t \leq t'$.*

4 Depth-bounded Systems

We now formally define depth-bounded systems. First we give a general behavioral definition of such systems in terms of their reachable configurations. We then describe a syntactic criterion on the defining equations of processes that ensures depth-boundedness.

4.1 A Behavioral Notion of Depth-bounded Systems

We start by making formal our notion of communication topologies of π-calculus configurations.

Communication topology. We use standard notation for directed and undirected graphs. A *directed labelled graph* over a finite set of labels L is a tuple (G, l_v, l_e) where G is a directed graph, $l_v : V(G) \to L$ is a *vertex labelling function*, and $l_e : V(G) \times V(G) \to L$ is an *edge labelling function*.

Let $\mathcal{P} = (I, \mathcal{E})$ be a process. Let further n be the maximal arity of all vectors of names occurring in I and $\mathcal{E}$, and let $\mathcal{A}$ be the set of all process identifiers occurring in $I, \mathcal{E}$. Define a set of labels $L \overset{\text{def}}{=} \{0, \dots, n\} \cup \mathcal{A} \cup \{\bullet\}$ where $\bullet$ is distinct from all process identifiers. Let P be a configuration of process $\mathcal{P}$ of the form

$$(\nu \boldsymbol{x})(\Pi_{i \in I} \overline{x_i}(\boldsymbol{x}_i) \mid \Pi_{j \in J} A_j(\boldsymbol{x}_j))$$

where $\boldsymbol{x} = x_1, \dots, x_m$, and the index sets $\{1..m\}$, I, and J are disjoint. The function ct maps P to a directed labelled graph over L as follows: the graph consists of vertices corresponding to threads, messages, and names occurring in the configuration. Each thread vertex is labelled by the process identifier of the corresponding thread in the configuration. There are edges between thread vertices and name vertices indicating that one of the names in the parameter vector of the thread is the name associated with that name vertex. Similarly, we have edges between message vertices and name vertices. Formally, $\mathsf{ct}(P)$ is a graph $((V, E), l_v, l_e)$ where

- V is a union of disjoint sets of vertices $\{v_i\}_{i \in I}$, $\{v_j\}_{j \in J}$ and $\{v_1, \dots, v_m\}$,
- $E = \{\, (v_h, v_k) \mid h \in J \cup I \ \wedge\ 1 \leq k \leq m \ \wedge\ x_{h_r} = x_k \text{ for some } 1 \leq r \leq n \,\} \cup$
  $\{\, (v_i, v_k) \mid i \in I \ \wedge\ 1 \leq k \leq m \ \wedge\ x_i = x_k \,\}$
- $l_v(v_k) = \begin{cases} A_k & \text{if } k \in J \\ \bullet & \text{otherwise} \end{cases}$
- $l_e(v_h, v_k) = \begin{cases} r & \text{if } h \in I \cup J \ \wedge\ 1 \leq k \leq m \ \wedge\ x_{h_r} = x_k \\ 0 & \text{if } h \in I \ \wedge\ 1 \leq k \leq m \ \wedge\ x_i = x_k \end{cases}$

We call $\mathsf{ct}(P)$ the *communication topology* of configuration P.

Tree-depth. The key ingredient for defining depth-bounded systems is the notion of the tree-depth of a graph [36].

A *tree* T is an undirected graph such that every pair of distinct vertices in T is connected by exactly one path. A *rooted tree* is a tree with a dedicated root vertex. A *rooted forest* is a disjoint union of rooted trees. The *height* of a vertex v in a rooted forest F, denoted $\mathrm{height}(F, v)$, is the number of vertices on the path from the root (of the tree to which v belongs) to v. The *height* of F is the maximal height of the vertices in F. Let v, w be vertices of F and let T be the tree in F to which y belongs. The vertex x is an *ancestor* of vertex y in F, denoted $x \preceq y$, if x belongs to the path linking y and the root of T. The *closure* $\mathit{clos}(F)$ of a rooted forest F is the graph consisting of the vertices of F and the edge set $\{\, \{x, y\} \mid x \preceq y, x \neq y \,\}$.

Definition 4 (Tree-depth). *The* tree-depth $\mathtt{td}(G)$ *of an undirected graph* G *is the minimum height of all rooted forests* F *such that* $G \subseteq \mathit{clos}(F)$.

The *tree-depth* $\mathtt{td}(G)$ of a directed labelled graph $G = ((V, E), l_v, l_e)$ is the tree-depth of the induced undirected graph with vertices V and edge set $\{\{v_1, v_2\} \mid (v_1, v_2) \in E\}$. The tree-depth of a configuration is the tree-depth of its communication topology. Finally, we say that a set of configurations $\mathcal{C}$ is *depth-bounded* if there exists $k \in \mathbb{N}$ such that all configurations $P \in \mathcal{C}$ have tree-depth at most k.

Definition 5 (Depth-bounded process). *A process* $\mathcal{P}$ *is called* depth-bounded *if its set of reachable configurations* $\mathit{Reach}(\mathcal{P})$ *is depth-bounded.*

4.2 A Syntactic Notion of Depth-bounded Systems

While the question whether a given process is depth-bounded is itself undecidable, one can identify simple syntactic conditions on the defining equations of the process that guarantee depth-boundedness and that are often satisfied in practice. We now describe one such syntactic condition.

We restrict ourself to processes that satisfy the unique receiver condition (cf. Section 3.1) such as actor systems. In order to enforce bounded tree-depth of such systems, we need to restrict the creation of new threads and their associated mailboxes. If the system allows to create and arbitrarily link threads then its tree-depth is unbounded. The following syntactic criterion limits the maximal size of the chains of threads and associated mailbox channels in the reachable communication topologies.

Our syntactic criterion for depth-boundedness only restricts the defining equations of the system. The initial configuration is unconstrained. Assume that all process identifiers used in equations of the system are divided into levels 0 to n. The first restriction on the defining equations of the system is that threads with process identifiers of level i can create only threads with process identifiers that are at least on level $i + 1$. If the system did not allow name mobility via messages, this restriction would impose a tree-like shape on the reachable communication topologies with threads of level 0 at the roots. In order to prevent chaining threads beyond the parent-child relation, we also restrict the name mobility appropriately. We partition the names that each thread knows into three categories *private*, *ascending*, and *descending*. The names of the thread's own mailboxes are the private names of the thread. If a thread sends a name to a thread of higher level, the receiving thread considers the sent name as a descending name. Conversely, a name sent to a thread of smaller or equal level is considered as an ascending name by the receiving thread. The restriction on name mobility are then as follows. A thread can send a private name to any other thread it knows. An ascending name can only be forwarded to threads of higher levels, respectively, descending names only to threads of lower levels.

The systems that obey the restrictions described above are depth-bounded. The reason for depth-boundedness is that in a reachable communication topology of such a system, every path between two threads of the same level goes over the private name of a common ancestor of lower level. Since the ancestor threads themselves have finitely many private names, and since the level of ancestors is bounded by 0, the length of any acyclic path in the communication topology is bounded. Note that the maximal depth of the reachable communication topologies does not only depend on the maximal level of the process identifiers, but also on the number of free parameters of the process identifiers and the shape of the communication topology of the initial configuration.

5 Decidability of the Control Reachability Problem

We now come to the main technical result of this paper. We show that the control reachability problem is decidable for depth-bounded processes. The control reachability problem for processes can be rephrased as a covering problem with respect to the following natural quasi-ordering $\leq$ on configurations: let P and Q be configurations then $P \leq Q$ if and only if Q corresponds to P extended by some process term Q', formally, if $P \equiv (\nu \boldsymbol{x})P'$ then there exist $\boldsymbol{y}$ and Q' such that

$$Q \equiv (\nu \boldsymbol{y}, \boldsymbol{x})(P' \mid Q')$$

In the remainder of this section we will prove that depth-bounded systems are well-structured transition systems for the quasi-ordering $\leq$ and that they are amenable to a forward analysis that decides the covering problem. The proofs for all the statements made in this section can be found in Appendix A.

5.1 Depth-bounded Systems are Well-structured

First, it is easy to prove that $\leq$ is indeed a quasi-ordering on configurations and upward compatible with respect to π-calculus reductions.

Proposition 6. *The relation $\leq$ is a quasi-ordering on configurations.*

Proposition 7. *Let $\mathcal{P}$ be a process. Then $\leq$ is upward compatible with respect to the transition relation of $\mathcal{P}$.*

It remains to show that $\leq$ is also a well-quasi-ordering on depth-bounded sets of configurations. For this purpose we encode configurations into labelled trees. The absence of infinite anti-chains over depth-bounded configurations then follows from a variation of Kruskal's tree theorem [28] that is due to Friedman [19].

First, it is instructive to understand the implications of $P \leq Q$ on the underlying communication topologies. Given two labelled graphs G_1 and G_2, we say G_1 is (isomorphic to) a ***subgraph*** of G_2, written $G_1 \hookrightarrow G_2$, iff there exists an injective label-preserving homomorphism from G_1 to G_2.

Lemma 8. *Let P and Q be configurations. Then $P \leq Q$ iff $\mathsf{ct}(P) \hookrightarrow \mathsf{ct}(Q)$.*

Tree encoding of configurations. A labelled rooted tree over a finite set of labels L is a pair (T, l) where T is a rooted tree and $l : V(T) \to L$ a vertex labelling function. We extend the relation $\hookrightarrow$ to rooted labelled trees, as expected, and we say that a tree T_1 is a *subtree* of tree T_2 whenever $T_1 \hookrightarrow T_2$ holds. In the following we fix a finite set of labels L. Let L_k be the set of all isomorphism classes of directed labelled graphs G over labels $L \cup (L \times \{1..k\})$ such that G has at most k vertices. Clearly, since L is finite, L_k is finite.

Given a directed labelled graph G over labels L that has tree-depth at most k, we can construct a labelled rooted tree (T, l) over the set of labels L_k from G as follows. First, let F be a rooted forest of minimal height whose closure contains the undirected graph induced by G. The rooted tree T is constructed from the forest F by extending F with a fresh root vertex r that has edges to all the roots of the trees in F. The labelling function l is defined as follows. Let $v \in V(T)$ be a vertex in T. If $v = r$ then $l(r)$ is the empty graph. Otherwise v is a vertex in F (and thus in G). Let P be the subgraph of G that is induced by the vertices on the path from v to the root (of the tree in F to which v belongs). Now construct a graph P_h from P by adding to the label of each vertex of P its height in F. Then $l(v)$ is the isomorphism class of P_h. Since G has tree-depth at most k, $P_h \in L_k$. Thus, l is well-defined. Let Trees_k be the function mapping a labelled directed graph G of tree-depth at most k to the set of all labelled rooted trees over L_k that can be constructed from G as described above. Furthermore, let tree_k be a function mapping each such G to some tree in $\mathit{Trees}_k(G)$.

Lemma 9. *Let $k \in \mathbb{N}$ and T_1, T_2 be trees in $\mathrm{rng}(\mathsf{tree}_k)$. If T_1 is a subtree of T_2 then $G_1 \hookrightarrow G_2$ for all $G_1 \in \mathsf{tree}_k^{-1}(T_1)$ and $G_2 \in \mathsf{tree}_k^{-1}(T_2)$.*

Let T be a rooted tree and $x, y \in V(T)$ two vertices. The infimum of x and y, denoted $x \inf y$, is the vertex $z \in V(T)$ with the greatest height such that $z \preceq x$ and $z \preceq y$. Given rooted trees T_1 and T_2, a function φ is an *inf-preserving embedding* from T_1 into T_2 iff (1) $\varphi : V(T_1) \to V(T_2)$ is injective, and (2) for all $x, y \in V(T_1)$, $\varphi(x \inf y) = \varphi(x) \inf \varphi(y)$. An embedding between two rooted labelled trees over the same set of labels is *label-preserving* iff it maps vertices to vertices with the same label. The inf and label-preserving embeddings induce a well-quasi-ordering on labelled trees. In particular, we have the following.

Theorem 10 (Friedman [19]). *Let $T_1, T_2, \ldots$ be an infinite sequence of labelled trees over a finite set of labels L. Then there exist $i < j$ and an inf and label-preserving embedding from T_i to T_j.*

Clearly, if a tree is a subtree of another tree then there exists an inf and label preserving embedding between these trees. For trees that result from the tree encoding of configurations the converse holds, too. Vertices of different levels of such trees have always different labels. Thus, an inf and label-preserving embedding between such trees also preserves antecedence of vertices.

Lemma 11. *Let $k \in \mathbb{N}$ and T_1, T_2 be trees in* $\mathrm{rng}(\mathsf{tree}_k)$*. Then the following two properties are equivalent*

1. *there exists an inf and label-preserving embedding from T_1 to T_2*
2. *T_1 is a subtree of T_2.*

Given a finite set of process identifiers PI, let $\mathcal{C}(PI, k)$ be the set of all configurations over PI that have tree-depth at most k. From Proposition 6, Theorem 10, and Lemmas 8, 9, and 11 now follows that each $\mathcal{C}(PI, k)$ is well-quasi-ordered.

Proposition 12. *Let $k \in \mathbb{N}$ and PI be a finite set of process identifiers. Then $(\mathcal{C}(PI, k), \leq)$ is a well-quasi-ordered set.*

Theorem 13. *Let $\mathcal{P} = (I, \mathcal{E})$ be a depth-bounded process of bound k and let PI be the process identifiers appearing in $I, \mathcal{E}$. Then for all $k' \geq k$, the tuple $(\mathcal{C}(PI, k'), I, \rightarrow_{\mathcal{E}}, \leq)$ is a well-structured transition system.*

The standard algorithm for deciding the covering problem for WSTS is as follows. Starting from the configuration t that is to be covered one computes the upward closure $\uparrow \mathit{Pred}^*(\uparrow t)$ of the backward-reachable configurations of t and then checks whether this set contains the initial configuration. The well-quasi-ordering ensures that the backward analysis terminates. While the forward-reachable configurations of a depth-bounded system have bounded tree-depth, this is not necessarily the case for the backward-reachable configurations. Thus, the set of backward-reachable configurations might not be well-quasi-ordered. Using a backward algorithm we can therefore only decide the covering problem for depth-bounded systems whose bound is known a priori. In the following we show that there exists a forward algorithm that overcomes this limitation.

5.2 A Forward Algorithm for the Covering Problem

The idea of a forward algorithm for solving the covering problem of a WSTS is to compute the *cover* $\downarrow \mathit{Post}^*(\downarrow s_0)$ of the initial configuration s_0 and then check whether this set contains the configuration to be covered. A well-known example of such an algorithm is the Karp and Miller algorithm [27] for Petri nets. Finding forward algorithms for WSTS is more complicated than finding backward algorithms. In order to effectively compute the cover, one needs to find a *completion* of the wqo set that contains all the limits of downward-closed sets. A formal characterization of these completions of wqo sets has been given in [20] and [17].

Adequate domain of limits. An *adequate domain of limits* (ADL) [20] for a well-quasi-ordered set $(X, \leq)$ is a tuple $(Y, \sqsubseteq, \gamma)$ where Y is a set disjoint from X; (L1) the map $\gamma : Y \cup X \to 2^X$ is such that $\gamma(z)$ is downward-closed for all $z \in X \cup Y$, and $\gamma(x) = \downarrow \{x\}$ for all $x \in X$; (L2) there is a limit point $\top \in Y$ such that $\gamma(\top) = X$; (L3) $z \sqsubseteq z'$ if and only if $\gamma(z) \subseteq \gamma(z')$; and (L4) for any downward-closed set D of X, there is a finite subset $E \subseteq Y \cup X$ such that $\gamma(E) = D$, where γ is extended to sets as expected: $\gamma(E) = \bigcup_{z \in E} \gamma(z)$. A *weak adequate domain of limits* (WADL) [17] for $(X, \leq)$ is a a tuple $(Y, \sqsubseteq, \gamma)$ satisfying (L1),(L3), and (L4). Note that any weak adequate domain of limits can be extended to an adequate domain of limits.

A WSTS $(X, x_0, \to, \leq)$ and an adequate domain of limits $(Y, \sqsubseteq, \gamma)$ are *effective* [20] if the following conditions are satisfied: (E1) X and Y are recursively enumerable; (E2) for any $x_1, x_2 \in X$, one can decide whether $x_1 \to x_2$; (E3) for any $z \in X \cup Y$ and for any finite subset $Z \subseteq X \cup Y$, one can decide whether $\mathit{Post}(\gamma(z)) \subseteq \gamma(Z)$; and (E4) for any finite subsets $Z_1, Z_2 \subseteq X \cup Y$, one can decide whether $\gamma(Z_1) \subseteq \gamma(Z_2)$. The expand, enlarge, and check algorithm presented in [20] decides the covering problem for effective well-structured transition systems with an adequate domain of limits.

Theorem 14 (Geeraerts et al. [20]). *There exists an algorithm to decide the covering problem for effective WSTSs with an adequate domain of limits.*

Extended process terms. We now describe an effective adequate domain of limits for depth-bounded configurations. In order to finitely represent the limits of infinite downward-closed sets of configurations we need to be able to express that certain subterms in a configuration can be replicated arbitrarily often. A natural solution to this problem is to extend process terms with the replication operator ! that is used as a recursion primitive in the standard definition of the π-calculus [31, 32]. Instead of using replication to express recursion, we use it to effectively represent infinite sets of configurations.

An *extended process term* is constructed from the operations of standard process terms defined in Section 3.1 and the replication operation $!P$. We extend the congruence relation $\equiv$ from process terms to extended process terms by adding the axiom $!P \equiv (P \mid !P)$. Using this extended congruence relation we carry over the definitions of the transition relations of processes and the quasi-ordering $\leq$ from process terms to extended process terms. We then define the denotation $\gamma(P)$ of an extended process term P as its downward closure restricted to non-extended process terms:

$$\gamma(P) = \{\, P' \mid P' \text{ configuration and } P' \leq P \,\}$$

The quasi-ordering $\sqsubseteq$ on extended process terms that is required for the adequate domain of limits is defined by condition (L3).

Finkel and Goubault-Larrecq characterize the minimal candidates for the WADLs of a wqo set X in terms of its ideal completion [17, Proposition 3.3].

This means that the set of all downward-closed *directed subsets*[1] of X form a WADL for X. Extended process terms are the ideal completions of sets of depth-bounded process terms.

Proposition 15. *The directed downward-closed sets of depth-bounded configurations are exactly the denotations of extended process terms.*

For proving that a downward-closed directed set of depth-bounded configurations D is the denotation of an extended process term, we use the fact that D contains a chain of process terms whose tree encodings form a chain of trees $\mathcal{T}$ ordered by the subtree relation. The trees $\mathcal{T}$ are again well-quasi-ordered. From $\mathcal{T}$ one can then construct a *hedge automaton* $\mathcal{A}$ [10, Chapter 8] whose tree language $\mathcal{L}(\mathcal{A})$ is both large and small in $\mathcal{T}$, i.e., the downward closure of the configurations obtained by reversing the tree encoding operation on the trees in $\mathcal{L}(\mathcal{A})$ is the set D. From the automaton $\mathcal{A}$ one can then easily construct the extended process term. The inverse direction also uses the construction of a hedge automaton. For proof details see Appendix A.

Let PI be a finite set of process identifiers. We denote by $\mathcal{L}(PI, k)$ the set of all extended process terms over PI such that the elements of $\mathcal{L}(PI, k)$ denote sets of k-bounded configurations in $\mathcal{C}(PI, k)$, and $\mathcal{L}(PI, k)$ itself does not contain the configurations in $\mathcal{C}(PI, k)$.

Proposition 16. *Let $k \in \mathbb{N}$ and let PI be a finite sets of process identifiers. Then $(\mathcal{L}(PI, k), \sqsubseteq, \gamma)$ is a weak adequate domain of limits for the well-quasi-ordered set $(\mathcal{C}(PI, k), \leq)$.*

It remains to argue that the WSTSs induced by depth-bounded processes together with their WADLs of extended process terms are effective. The conditions (E1) and (E2) are clearly satisfied. Also given an extended process term z we can compute a finite set of extended process terms denoting $Post(\gamma(z))$. Note further that Proposition 16 implies that for any finite subsets $Z_1, Z_2 \subseteq \mathcal{L}(PI, k)$, $\gamma(Z_1) \subseteq \gamma(Z_2)$ holds if and only if for all $z_1 \in Z_1$ there exists $z_2 \in Z_2$ such that $\gamma(z_1) \subseteq \gamma(z_2)$. The inclusion problem $\gamma(z_1) \subseteq \gamma(z_2)$ can be reduced to the language inclusion problem for deterministic hedge automata, which is decidable. For this purpose, one computes deterministic hedge automata from the finitely many tree encodings of the configurations of z_1 and z_2 and then checks whether the language of some automaton of z_1 is included in the language of some automaton of z_2. Thus conditions (E3) and (E4) are also satisfied.

Finally, let us explain why the expand, enlarge, and check algorithm [20] terminates on depth-bounded systems even if the bound of the system is not known a priori. The idea of the algorithm is to simultaneously enumerate two infinite increasing chains. The first chain $X_0 \subseteq X_1 \dots$ is a sequence of finite subsets of X that contains all reachable configurations of the analyzed system. The second chain $Y_0 \subseteq Y_1 \subseteq \dots$ is a sequence of finite subsets of Y that contains

[1] A directed set D for a quasi-ordered set X is a nonempty subset of X such that each pair of points $x, y \in D$ has a common upper bound in D.

all limits Y. In each iteration i the algorithm computes an under and an over-approximation of the analyzed system for the current pair (X_i, Y_i) of elements in the chain. This approximations are such that the under-approximation is guaranteed to detect that t can be covered if X_i contains a path to a covering state. The over-approximation is guaranteed to detect that t can not be covered if Y_i can express $\downarrow Post^*(\downarrow s_0)$ and this set does not cover t. The conditions on the chains ensure that one of the two conditions eventually holds for some $i \in \mathbb{N}$.

For deciding the covering problem of depth-bounded systems we can now simply enumerate the sets $\mathcal{C}(PI) = \bigcup_{k\in\mathbb{N}} \mathcal{C}(PI, k)$ and $\mathcal{L}(PI) = \bigcup_{k\in\mathbb{N}} \mathcal{L}(PI, k)$. Then in each iteration of the algorithm the pair (X_i, Y_i) is contained in some $(\mathcal{C}(PI, k), \mathcal{L}(PI, k))$ and the conditions on the chains for termination of the algorithm are still satisfied.

Theorem 17. *The covering problem is decidable for the well-structured transition systems induced by depth-bounded processes.*

Corollary 18. *The control reachability problem is decidable for depth-bounded processes.*

6 Undecidability of the Reachability Problem

We will now prove that the general reachability problem for depth-bounded systems is undecidable. For this purpopse we reduce the reachability problem for reset nets to the reachability problem in a depth-bounded system that satisfies the unique receiver condition [4, 5]. The problem of reachability for reset nets is undecidable [6, 14]. In the following, we assume familiarity with Petri nets and their semantics. For a formal definition of reset nets see, e.g., [6, 14].

Reset nets are Petri nets which have, in addition to the standard arcs that connect places and transitions, special reset arcs. A reset arc between a place and a transition "flushes" the place when the transition fires, i.e., it removes all tokens from the place in a single operation. We can simulate a reset net using a depth-bounded system with a single thread whose parameters are names corresponding to the places in the net. Tokens in places are modeled by messages sent to the corresponding names. The firing of a transition of the net is simulated by receiving messages from the respective names of places in the preset of the transition (token consumption) and sending messages to the respective names of places in the postset of the transition (token generation). A reset arc can be modeled by generating a fresh name for the flushed place and assigning this name to the corresponding parameter of the thread. The old name that was previously assigned to this parameter becomes *dead* and pending messages can no longer be received. This process is similar to a flush of a place in a reset net [5].

Theorem 19. *The reachability problem for depth-bounded processes is undecidable.*

7 Further Related Work

We now discuss further related work in the verification of π-calculus processes and the analysis of well-structured transition systems.

The control reachability problem for the π-calculus has been studied in [5, 11, 33, 42]. The approaches taken in [33, 42] consider only finitary systems that impose a bound on the number of threads that can be dynamically created. Delzanno [11] considers an abstraction-based approach that applies to the full asynchronous π-calculus. This approach is sound but in general incomplete. More closely related to our work is [5] which considers *input-bounded systems*, a syntactically defined fragment of the asynchronous π-calculus that allows name creation and name mobility and has similar theoretical properties as depth-bounded systems (control reachability is decidable, general reachability is not). Input-bounded systems and depth-bounded systems are incomparable. Unlike depth-bounded systems, input-bounded systems cannot truly model the dynamic creation of an unbounded set of threads by a given thread such that all of these threads remain active and communicate. Because of such restrictions, input-bounded systems are less interesting from a practical point of view. Conversely, input-bounded systems are not depth-bounded because they enable the creation of unbounded chains of inactive threads. We suspect that there is a relaxation of the depth-boundedness condition that subsumes both fragments and for which control-reachability is still decidable.

There is a significant body of work on well-structured transition systems [1, 18]. The well-quasi-ordering on trees that is given by Kruskal's tree theorem has been used, e.g., for the analysis of tree pattern rewriting systems for XML documents [21] and for the analysis of biological systems [12]. However, in both cases the configurations of the underlying well-structured transition systems are directly given by trees rather than general graphs. More closely related to our work is the application of the graph minor theorem [39] in the context of graph rewriting systems [26]. Graph minors are homomorphisms that induce a well-quasi-ordering on graphs. However, the graph minor order is not upward compatible with respect to the π-calculus semantics. Therefore, this approach is not applicable in our context. We restrict ourself to a specific class of compatible graph minors, namely, subgraph isomorphisms and a specific class of graphs, namely, graphs of bounded tree-depth. The concept of depth-boundedness easily generalizes from the analysis of π-calculus processes to the analysis of graph rewriting systems that are naturally ordered by subgraph isomorphism.

8 Conclusion

We identified the novel class of depth-bounded message passing systems. The key ingredient for the definition of this class is the graph-theoretic notion of tree-depth. Depth-bounded systems cover many practical use cases of message passing with dynamic creation of threads and name mobility. By proving that the control reachability problem for depth-bounded systems is decidable we showed that this class is also amenable to automated verification.

The question whether a given message passing system is depth-bounded is of independent theoretical interest and an intriguing problem for future research.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
2. G. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems.* PhD thesis, MIT CSAIL, 1986.
3. G. Agha and P. Thati. An algebraic theory of actors and its application to a simple object-based language. In *Essays in Memory of Ole-Johan Dahl*, pages 26–57, 2004.
4. R. M. Amadio. On modelling mobility. *Theor. Comput. Sci.*, 240(1):147–176, 2000.
5. R. M. Amadio and C. Meyssonnier. On decidability of the control reachability problem in the asynchronous pi-calculus. *Nord. J. Comput.*, 9(1):70–101, 2002.
6. T. Araki and T. Kasami. Some decision problems related to the reachability problem for petri nets. *Theor. Comput. Sci.*, 3(1):85–104, 1976.
7. J. Armstrong. A history of Erlang. In B. G. Ryder and B. Hailpern, editors, *HOPL*, pages 1–26. ACM, 2007.
8. G. Boudol. Asynchrony and the pi-calculus. Technical report, INRIA Report 1702, INRIA, Sophia Antipolis, 1992.
9. W. Clinger. *Foundations of Actor Semantics.* PhD thesis, MIT CSAIL, 1981.
10. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. Available from: `http://tata.gforge.inria.fr/`, 2008. release November, 18th 2008.
11. G. Delzanno. A symbolic procedure for control reachability in the asynchronous pi-calculus: Extended abstract. *Electr. Notes Theor. Comput. Sci.*, 98:21–33, 2004.
12. G. Delzanno and L. Begin. A biologically inspired model with fusion and clonation of membranes. In *UC*, pages 64–82, 2008.
13. R. Diestel. Relating Subsets of a Poset, and a Partition Theorem for WQOs. *Order*, 18(3):275–279, 2001.
14. C. Dufourd, A. Finkel, and P. Schnoebelen. Reset nets between decidability and undecidability. In K. G. Larsen, S. Skyum, and G. Winskel, editors, *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998.
15. A. Finkel. A Generalization of the Procedure of Karp and Miller to Well Structured Transition Systems. In T. Ottmann, editor, *ICALP*, volume 267 of *Lecture Notes in Computer Science*, pages 499–508. Springer, 1987.
16. A. Finkel. Reduction and covering of infinite reachability trees. *Inf. Comput.*, 89(2):144–179, 1990.
17. A. Finkel and J. Goubault-Larrecq. Forward Analysis for WSTS, Part I: Completions. In *STACS*, volume 09001 of *Dagstuhl Seminar Proceedings*, pages 433–444, 2009.
18. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
19. H. M. Friedman. Internal finite tree embeddings. In *Reflections on the foundations of mathematics*, volume 15 of *Lecture Notes in Logic*, pages 60–91. Association for Symbolic Logic, 2002.
20. G. Geeraerts, J.-F. Raskin, and L. V. Begin. Expand, enlarge and check: New algorithms for the coverability problem of wsts. *J. Comput. Syst. Sci.*, 72(1):180–203, 2006.

21. B. Genest, A. Muscholl, O. Serre, and M. Zeitoun. Tree pattern rewriting systems. In *ATVA*, pages 332–346, 2008.
22. P. Haller and M. Odersky. Scala actors: Unifying thread-based and event-based programming. *Theor. Comput. Sci.*, 410(2-3):202–220, 2009.
23. C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *IJCAI*, pages 235–245, 1973.
24. K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *ECOOP*, pages 133–147, 1991.
25. D. Janssens, M. Lens, and G. Rozenberg. Computation graphs for actor grammars. *J. Comput. Syst. Sci.*, 46(1):60–90, 1993.
26. S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. In *CAV*, pages 214–226, 2008.
27. R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.
28. J. B. Kruskal. Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions on the American Mathematical Society*, 95(2):210–225, 1960.
29. R. Laver. On Fraïssé's Order Type Conjecture. *Ann. of Math.*, 93(1):89–111, 1971.
30. R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.
31. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, i. *Inf. Comput.*, 100(1):1–40, 1992.
32. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, ii. *Inf. Comput.*, 100(1):41–77, 1992.
33. U. Montanari and M. Pistore. Checking bisimilarity for finitary pi-calculus. In *CONCUR*, pages 42–56, 1995.
34. C. S. J. A. Nash-Williams. On better-quasi-ordering transfinite sequences. *Proc. Camb. Phil. Soc.*, 64:273–290, 1968.
35. U. Nestmann and B. C. Pierce. Decoding choice encodings. *Inf. Comput.*, 163(1):1–59, 2000.
36. J. Nešetřil and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006.
37. M. Odersky. The scala experiment - can we provide better language support for component systems? In *APLAS*, page 364, 2004.
38. R. Pointer. kestrel - small, scalable message queue daemon. `http://robey.lag.net/2008/11/27/scarling-to-kestrel.html`.
39. N. Robertson and P. D. Seymour. Graph Minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory*, 90(2):234–357, 2004.
40. M. Sirjani, M. M. Jaghoori, C. Baier, and F. Arbab. Compositional semantics of an actor-based language using constraint automata. In *COORDINATION*, volume 4038 of *LNCS*, pages 281–297, 2006.
41. S. Srinivasan and A. Mycroft. Kilim: Isolation-typed actors for java. In *ECOOP*, volume 5142 of *LNCS*, pages 104–128. Springer, 2008.
42. P. Yang, C. R. Ramakrishnan, and S. A. Smolka. A logical encoding of the pi-calculus: Model checking mobile processes using tabled resolution. In *VMCAI*, pages 116–131, 2003.

A Additional Proofs

A.1 Proof of Proposition 6

Proposition 6. *The relation $\leq$ is a quasi-ordering on configurations.*

Proof. Let $P \equiv (\nu \boldsymbol{x})P'$. Then by definition of the congruence relation on process terms we have $P \equiv (\nu \boldsymbol{x})(P' \mid 0)$. Thus $\leq$ is reflexive.

For proving that $\leq$ is transitive, let P, Q, R be configurations of the form $P \equiv (\nu \boldsymbol{x})P'$, $Q \equiv (\nu \boldsymbol{y})Q'$, and $R \equiv (\nu \boldsymbol{z})R'$, and assume $P \leq Q$ and $Q \leq R$. Then we have $Q \equiv (\nu \boldsymbol{y'}, \boldsymbol{x})(P' \mid Q'')$ and $R \equiv (\nu \boldsymbol{z'}, \boldsymbol{y})(Q' \mid R'')$. We can rewrite R as $(\nu \boldsymbol{z'}, \boldsymbol{y'}, \boldsymbol{x})(P' \mid Q'' \mid R'')$. Thus, we have $P \leq R$. □

A.2 Proof of Proposition 7

Proposition 7. *Let $\mathcal{P}$ be a process. Then $\leq$ is upward compatible with respect to the transition relation of $\mathcal{P}$.*

Proof. Let $\mathcal{P} = (I, \mathcal{E})$ and P, R be configurations of $\mathcal{P}$ with $P \equiv (\nu \boldsymbol{u})P'$ and $P \leq R$, i.e., R is of the form $R \equiv (\nu \boldsymbol{z}, \boldsymbol{u})(P' \mid R')$. Further, let Q be a configuration such that $P \rightarrow_{\mathcal{E}} Q$. From the definition of $\rightarrow_{\mathcal{E}}$ follows that there is some process identifier A and defining equation $A(\boldsymbol{x}) = x(\boldsymbol{x'}).(\nu \boldsymbol{x''})(Q')$ in $\mathcal{E}$ such that P is of the refined form $P \equiv (\nu \boldsymbol{u})(A(\boldsymbol{v}) \mid \overline{w}(\boldsymbol{w}) \mid P'')$ and Q is of the form $Q \equiv (\nu \boldsymbol{u}, \boldsymbol{y})(P'' \mid \sigma(Q'))$, where $\boldsymbol{y}$ are fresh names and σ the proper substitution. It follows that R is of the refined form $R \equiv (\nu \boldsymbol{z}, \boldsymbol{u})(A(\boldsymbol{v}) \mid \overline{w}(\boldsymbol{w}) \mid P'' \mid R')$. Now define $S \overset{\text{def}}{=} (\nu \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{y})(P'' \mid \sigma(Q') \mid R')$. Then we have $R \rightarrow_{\mathcal{E}} S$ and $Q \leq S$. □

A.3 Proof of Lemma 8

Lemma 8. *Let P and Q be configurations. Then $P \leq Q$ iff $\mathsf{ct}(P) \hookrightarrow \mathsf{ct}(Q)$.*

Proof. The "$\Rightarrow$" direction follows immediately from the definitions of ct, $\leq$, and $\hookrightarrow$. For the "$\Leftarrow$" direction assume $\mathsf{ct}(P) \hookrightarrow \mathsf{ct}(Q)$ and assume that P has the form $P = (\nu v)P'$. We construct names $\boldsymbol{y}$ and a process term Q' from $\mathsf{ct}(Q)$ such that $Q \equiv (\nu \boldsymbol{y}, \boldsymbol{x})(P' \mid Q')$. Let $\mathsf{ct}(Q) = (V, E, l)$. First, partition the vertices V and edges E into pairs of disjoint sets V_1 and V_2, respectively, E_1 and E_2 such that (V_1, E_1) is the subgraph of $\mathsf{ct}(Q)$ to which $\mathsf{ct}(P)$ is isomorphic. Partition V_1 further into sets $V_{1,i}$ and $V_{1,o}$ such that $V_{1,o}$ contains the vertices from V_1 that have incoming edges and $V_{1,o}$ all other vertices of V_1. Similarly, partition V_2 into $V_{2,i}$ and $V_{2,o}$. The vertices $V_{1,i}$ are isomorphic to the vertices in $\mathsf{ct}(P)$ that correspond to the names $\boldsymbol{x}$, and the vertices $V_{1,o}$ are isomorphic to the vertices in $\mathsf{ct}(P)$ that correspond to the messages and threads in P'. We can now choose a fresh name y for each vertex in $V_{2,i}$ and define $\boldsymbol{y}$ as the vector over these fresh names. Then Q' is the parallel composition of messages and threads with messages corresponding to vertices $v \in V_{2,o}$ such that $l(v) = \bullet$, threads corresponding to vertices $v \in V_{2,o}$ that are labelled by the corresponding process identifier, and names chosen according to the names associated with vertices in $V_{1,i} \cup V_{2,i}$ that are connected to the vertices v with edges in E_2. □

A.4 Proof of Lemma 9

Lemma 9. *Let $k \in \mathbb{N}$ and T_1, T_2 be trees in $\mathrm{rng}(\mathsf{tree}_k)$. If T_1 is a subtree of T_2 then $G_1 \hookrightarrow G_2$ for all $G_1 \in \mathsf{tree}_k^{-1}(T_1)$ and $G_2 \in \mathsf{tree}_k^{-1}(T_2)$.*

Proof. If T_1 is a subtree of T_2 then there exists a label-preserving injective homomorphism $\varphi_T : V(T_1) \to V(T_2)$ between T_1 and T_2. Let $G_1 \in \mathsf{tree}_k^{-1}(T_1)$ and $G_2 \in \mathsf{tree}_k^{-1}(G_2)$. Note that the vertices of T_1 are exactly the vertices of G_1 except for the added root vertex. The same is true for T_2 and G_2. What is more, φ_T maps the root of T_1 to the root of T_2 and vertices of G_1 to vertices of G_2. Thus, we can define $\varphi_G : V(G_1) \to V(G_2)$, the restriction of φ_T to vertices in G_1 respectively G_2. We will now show that φ_G is an injective label-preserving homomorphism from G_1 to G_2.

The fact that φ_G is injective immediately follows from the fact that φ_T is injective. For proving that φ_G is a homomorphism, let v_1, w_1 be vertices in $V(G_1)$ such that there exists an edge $(v_1, w_1) \in E(G_1)$ with label ℓ. Since G_1 is contained in the closure of the forest that is used to construct T_1, we conclude that $v_1 \preceq w_1$ or $w_1 \preceq v_1$ holds in T_1. Without loss of generality assume that $v_1 \preceq w_1$ holds. Let $h_{v_1} = \mathrm{height}(T_1, v_1)$ and $h_{w_1} = \mathrm{height}(T_1, w_1)$. Let further P_1 be the subgraph of G_1 with the extended vertex labels that is induced by the vertices on the path from r to w, as described in the construction of tree_k. From $v_1 \preceq w_1$ follows $v_1 \in V(P_1)$. Hence, there exists an edge $(v_1, w_1) \in E(P_1)$ with label ℓ. Now, let $v_2 = \varphi_G(v_1)$ and $w_2 = \varphi_G(w_1)$. Since T_1 is a subtree of T_2 and φ_T preserves adjacency, we know that $v_2 \preceq w_2$ holds in T_2 and $\mathrm{height}(T_2, v_2) = h_{v_1}$, respectively, $\mathrm{height}(T_2, w_2) = h_{w_1}$. Let P_2 be the subgraph of G_2 induced by the vertices on the path from the root of T_2 to w_2, again, with the extended labels as for P_1. Since P_1 and P_2 are representatives of the isomorphism classes that serve as labels for w_1 in T_1 and w_2 in T_2, and since φ_T preserves labels, we know that P_1 and P_2 are isomorphic. Hence, there is an edge $(v', w') \in E(P_2)$ with label ℓ such that v_1 has the same label in P_1 as v' in P_2, respectively, w_1 the same label as w'. Since the second component of the extended vertex label corresponds to the height of the vertex in the respective tree, we conclude $h_{v_1} = \mathrm{height}(T_2, v')$ and $h_{w_1} = \mathrm{height}(T_2, w')$. The height of a vertex in a path of a tree uniquely determines the vertex. Thus, we have $v' = v_2$ and $w' = w_2$, since both v_2 and w_2 are in P_2. Therefore φ_G is a homomorphism.

We already proved implicitly that φ_G preserves edge labels. The proof that φ_G also preserves the vertex labelling follows a similar argumentation. □

A.5 Proof of Lemma 11

Lemma 11. *Let $k \in \mathbb{N}$ and T_1, T_2 be trees in* $\mathrm{rng}(\mathsf{tree}_k)$*. Then the following two properties are equivalent*

1. *there exists an inf and label-preserving embedding from T_1 to T_2*
2. *T_1 is a subtree of T_2.*

Proof. Let φ be an inf and label-preserving embedding from T_1 to T_2. Note that by definition of tree_k, a label of a vertex $v \in V(T)$ for a tree $T \in \mathrm{rng}(\mathsf{tree}_k)$ depends on the path from the root of the tree to v. Thus, if two vertices $v_1 \in V(T_1)$ and $v_2 \in V(T_2)$ have the same label then they also have the same height. Since φ is label-preserving, we deduce that φ maps vertices in T_1 to vertices in T_2 of the same height. From this and the second property of inf-preserving follows that adjacency is preserved by the embedding. Since T_1, T_2 are trees, non-adjacency is also preserved (because the path between any two vertices is unique). Therefore T_1 is a subtree of T_2. □

A.6 Proof of Proposition 12

Proposition 12. *Let $k \in \mathbb{N}$ and PI be a finite set of process identifiers. Then $(\mathcal{C}(PI, k), \leq)$ is a well-quasi-ordered set.*

Proof. Proposition 6 states that $\leq$ is a quasi-ordering for arbitrary configurations. It remains to show that $\leq$ has no infinite antichains in $\mathcal{C}$. Thus, let $P_1, P_2, \ldots$ be an infinite sequence of configuration in $\mathcal{C}$. Let L be the set of all labels used in $\mathsf{ct}(P)$ for the configurations $P \in \mathcal{C}$. Since the configurations in $\mathcal{C}$ range over finitely many process identifiers, L is finite. From the fact that $\mathcal{C}$ is depth-bounded follows that there exists some $k \in \mathbb{N}$ such that for all $i \in \mathbb{N}$, $\mathsf{ct}(P_i)$ has tree-depth at most k. Thus, we can define the infinite sequence $\mathsf{tree}_k(\mathsf{ct}(P_1)), \mathsf{tree}_k(\mathsf{ct}(P_2)), \ldots$ of labelled trees over the finite set of labels L_k. From Theorem 10 follows that there exists $i < j$ and an inf and label-preserving embedding from $\mathsf{tree}_k(\mathsf{ct}(P_i))$ to $\mathsf{tree}_k(\mathsf{ct}(P_j))$. From Lemma 11 and Lemma 9 follows that $\mathsf{ct}(P_i)$ is a subgraph of $\mathsf{ct}(P_j)$. Finally, from Lemma 8 follows that $P_i \leq P_j$. Hence, $(\mathcal{C}, \leq)$ is a well-quasi-ordered set. □

A.7 Proof of Proposition 15

Before we prove Proposition 15, we recall some properties of well-quasi-orderings and better-quasi-orderings [34], and define hedge automata.

Finite partitions of well-quasi-ordered sets. Let $(X, \leq)$ be a well-quasi-ordered set. We extend the ordering $\leq$ to an ordering $\leq$ on subsets of X as expected: for $Y_1, Y_2 \subseteq X$, we have $Y_1 \leq Y_2$ iff for all $y_1 \in Y_1$ there exists $y_2 \in Y_2$ if $y_1 \leq y_2$. For $Y \subseteq X$ we call $Y' \subseteq X$ *large* in Y iff $Y \leq Y'$. Conversely, we call Y' *small* in Y if $Y' \leq Y$. A subset $Y \subseteq X$ of X is called *irreducible* if for any $Y_1, Y_2 \subseteq Y$, $Y \leq Y_1 \cup Y_2$ implies $Y \leq Y_1$ or $Y \leq Y_2$.

Proposition 20 (Diestel [13]). *Let $(X, \leq)$ be a well-quasi-ordered set. Then for any countable $Y \subseteq X$ the following are equivalent:*

1. *Y is irreducible*
2. *Y contains a chain C such that $Y \leq C$*
3. *Y is directed*

Theorem 21 (Diestel [13]). *If $(X, \leq)$ is a well-quasi-ordered set then X can be partitioned into finitely many irreducible subsets.*

We call a partition $\mathcal{P} \subseteq 2^X$ of a well-quasi-ordered set $(X, \leq)$ an *infinite chain partition* if and only if (1) $\mathcal{P}$ is finite and (2) for all $Y \in \mathcal{P}$, either Y is a singleton or Y contains an infinite chain C such that $Y \leq C$.

Proposition 22. *If $(X, \leq)$ is a countable well-quasi-ordered set then there exists an infinite chain partition of X.*

Proof. We can construct an infinite chain partition $\mathcal{P}$ of X recursively using the following procedure: according to Theorem 21, X can be partitioned into finitely many irreducible subsets $Y_1, \dots, Y_n$. By Proposition 20, for each $1 \leq i \leq n$, Y_i contains a chain C_i with $Y_i \leq C_i$. For each $1 \leq i \leq n$, check if Y_i contains an infinite chain with this property. If it does then add Y_i to $\mathcal{P}$. Otherwise pick one finite chain C_i with $Y_i \leq C_i$. Since C_i is finite it contains a greatest element y_i. Then let $Z_i = \{\, y \in Y_i \mid y_i \leq y \,\}$ be the set of elements in Y_i that are equivalent to y_i wrt. the quasi-ordering $\leq$. Since Y_i contains no infinite chains that are large in Y_i, the set Z_i is finite. Then add all singletons $\{z\}$ with $z \in Z_i$ to $\mathcal{P}$ and recursively apply the above procedure on the well-quasi-ordered set $(Y_i - Z_i, \leq)$. Clearly, if this procedure terminates then the resulting set $\mathcal{P}$ is an infinite chain partition of X. Thus, assume that the procedure does not terminate. Then the algorithm constructs a strictly decreasing infinite sequence $Y_1 \supsetneq Y_2 \supsetneq \dots$ of subsets of Y with $Y_i - Y_{i+1} > Y_{i+1} - Y_{i+2}$ for all $i \in \mathbb{N}$. Define $Z_i = Y_i - Y_{i+1}$ then each Z_i is nonempty, i.e. we can choose $z_i \in Z_i$ for each $i \in \mathbb{N}$ such that we get an infinite descending chain $z_1 > z_2 > \dots$ of elements in X. This contradicts the fact that $\leq$ is well-founded. □

Better-quasi-orderings. Let $\leq$ be a quasi-ordering on a set X then define the quasi-ordering $\leq_1$ on subsets of X as follows: for $Y_1, Y_2 \subseteq X$, we have $Y_1 \leq_1 Y_2$ iff there exists an injection $\phi : Y_1 \to Y_2$ such that for all $y_1 \in Y_1$, $\phi(y_1) \leq y_2$. We are interested in wqo sets $(X, \leq)$ whose powerset is again a wqo with respect to $\leq_1$. For this purpose we consider Nash-William's better-quasi-orderings [34]. Better-quasi-orderings are particular well-behaved well-quasi-orderings. Unlike well-quasi-orderings, they are closed under the powerset construction. The formal definition of better-quasi-ordering (bqo) is rather technical and not required for understanding our proof of Proposition 15. We therefore refer to [34] for the actual definition of bqo sets. In the following, we only state the properties of bqo sets that we will need in our proof.

Proposition 23. *Let $(X, \leq)$ be a bqo then*

1. *$(X, \leq)$ is a wqo*
2. *$(2^X, \leq_1)$ is a bqo*
3. *every $Y \subseteq X$ is a bqo with respect to the restriction of $\leq$ to Y.*

Properties 1 and 2 are proved in [34]. Property 3 immediately follows from the definition of better-quasi-orderings.

Laver [29] proved a generalization of Kruskal's tree theorem stating that countable trees labelled by a bqo are a bqo under inf-preserving embedding. Similar to Friedman's special case of Kruskal's tree theorem, we get the following special case of Laver's theorem.

Theorem 24. *The set of all countable rooted labelled trees over a finite set of labels is a bqo with respect to inf and label-preserving embedding.*

Hedge automata. Finally, we introduce our version of hedge automata [10, Chapter 8]. A *(nondeterministic) finite hedge automaton* $\mathcal{A}$ over a finite alphabet Σ is a tuple (Q, Σ, Q_f, Δ) where Q is a finite set of states, $Q_f \subseteq Q$ is a set of final states, and Δ is a finite set of transition rules of the following form:

$$a(R) \rightarrow q$$

where $a \in \Sigma$, $q \in Q$, and $R \subseteq Q^*$ is a regular language over Q. These languages R occuring in the transition rules are called *horizontal languages.*

A *run* of $\mathcal{A}$ on a rooted labelled tree T with vertex label function $l : V(T) \rightarrow \Sigma$ is a vertex label function $r : V(T) \rightarrow Q$ such that for each vertex $v \in V(T)$ with $a = l(v)$ and $q = r(v)$ there is a transition rule $a(R) \rightarrow q$ with $r(v_1) \ldots r(v_n) \in R$ where $v_1, \ldots, v_n$ are the immediate successors of v in T. In particular, to apply a rule to a leaf, the empty word ϵ has to be in the horizontal language of the rule R.

A rooted labelled tree T is *accepted* by $\mathcal{A}$ if there is a run r of $\mathcal{A}$ on T such that r labels the root of T by a final state. The *language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all rooted labelled trees over Σ that are accepted by $\mathcal{A}$.

We now have all the necessary ingredients for proving Proposition 15.

Proposition 15. *The directed downward-closed sets of depth-bounded configurations are exactly the denotations of extended process terms.*

Proof. We first prove that every directed downward-closed set of depth-bounded configurations is the denotation of an extended process term. For this purpose, let $D = (P_i)_{i \in \mathbb{N}}$ be such a family of configurations and let k be the maximal tree-depth of all configurations in D. Choose some $Q_0 \in D$ with tree-depth k. Using Q_0 construct an ascending chain $D' = (Q_i)_{i \in \mathbb{N}}$ as follows: for each $i \in \mathbb{N}$ choose $Q_i \in D$ such that $P_i \leq Q_i$ and $Q_{i-1} \leq Q_i$. Such Q_i exists for each $i \in \mathbb{N}$ since D is directed and, by induction, $Q_{i-1} \in D$. Then by construction (1) $D = \downarrow D'$ and (2) all elements in D' have tree-depth k. Let $(G_i)_{i \in \mathbb{N}}$ be the family of directed labelled graphs $G_i = \mathsf{ct}(Q_i)$. Now for each $i \in \mathbb{N}$ choose a tree $T_i \in \mathit{Trees}_k(G_i)$ such that the family $\mathcal{T} = (T_i)_{i \in \mathbb{N}}$ is an ascending chain with respect to the subtree relation. Such a family exists because the G_i are ordered by subgraph isomorphism and all G_i have the same tree-depth. Without loss of generality we assume that the vertex sets of all graphs G_i are pairwise disjoint.

Let $V = \bigcup_{i \in \mathbb{N}} V(T_i)$, $E = \bigcup_{i \in \mathbb{N}} E(T_i)$, and let l be the union of all the vertex labelling functions of the labelled trees T_i. The height of the vertices in the trees T_i range from 1 to $k+1$. For a node $x \in V$ of height $h > 1$ we denote by $\mathit{parent}(v) \in V$ the parent of v in the tree T_i to which v belongs. Similarly, for a node $v \in V$ we denote by $\mathit{Children}(v)$ the set of all vertices that are direct successors of v in the tree to which v belongs. We extend the functions *parent* to sets of vertices, as expected. Furthermore, let $T(v)$ be the subtree rooted in v of the tree T_i to which v belongs. For all $1 \leq h \leq k+1$, let V_h be the set of all vertices in V that have height h. For all h we extend the relation $\hookrightarrow$ from labelled rooted trees to vertices in V_h as expected: for all $v, w \in V_h$, $v \hookrightarrow w$ iff $T(v) \hookrightarrow T(w)$. From Theorem 24, Property 3 of Proposition 23, and the fact that on the tree encodings the relation $\hookrightarrow$ coincides with inf and label-preserving embedding, follows that for all h, $(V_h, \hookrightarrow)$ is a bqo.

We will now construct a finite hedge automaton $\mathcal{A}$ from the family of trees $\mathcal{T}$ whose language is both small and large in $\mathcal{T}$. For this purpose we define an equivalence relation on each V_h that partitions V_h into finitely many equivalence classes. These equivalence classes serve as the states of the automaton.

For each $i \in \mathbb{N}$ fix some injective label-preserving homomorphism $\phi_i : V(T_i) \to V(T_{i+1})$ and denote by $\phi_{[i,j]}$ the composition $\phi_{j-1} \circ \cdots \circ \phi_i$ if $j > i$ and the identify function id if $j = i$. Then define an equivalence relation $\sim$ on V as follows: for all $v_i \in V(T_i)$ and $v_j \in V(T_j)$

$$v_i \sim v_j \quad \text{iff} \quad \begin{array}{l} i \leq j \text{ and } \phi_{[i,j]}(v_i) = v_j \text{ or} \\ i \geq j \text{ and } \phi_{[j,i]}(v_j) = v_i \end{array}$$

Now, recursively define an equivalence relation $\simeq_h$ on V_h for each $1 \leq h \leq k+1$ as follows: for $h = 1$ we simply have $v \simeq_1 w$ for all $v, w \in V_1$. In order to define $\simeq_h$ for $h > 1$ we need some intermediate definitions. Given an equivalence class U in the quotient of V_{h-1} wrt. $\simeq_{h-1}$, let $\mathit{Children}(U)$ be the set of equivalence classes $\tilde{v}$ in the quotient $V_h/_\sim$ such that some $v \in \tilde{v}$ has a parent in U. Since $(V_h, \hookrightarrow)$ is a bqo, and $\mathit{Children}(U) \subseteq 2^{V_h}$, it follows from Proposition 23 that $(\mathit{Children}(U), \hookrightarrow_1)$ is also a bqo and thus a wqo. Furthermore, $\mathit{Children}(U)$ is countable. Thus, by Proposition 22 there exists an infinite chain partition of $\mathit{Children}(U)$. For each U, choose one such infinite chain partition $\mathcal{P}(U)$ of $\mathit{Children}(U)$. Then for $v, w \in V_h$ we have: $v \simeq_h w$ iff there exists $U \in V_{h-1}/_{\simeq_{h-1}}$ such that (1) $\mathit{parent}(v), \mathit{parent}(w) \in U$ and (2) there is $P \in \mathcal{P}(U)$ such that $v, w \in \bigcup P$.

We can easily prove by induction on h that $\simeq_h$ is indeed an equivalence relation on V_h and that $\simeq_h$ partitions V_h into finitely many equivalence classes. Furthermore, one can easily prove the following properties: let $U \in V_h/_{\simeq_h}$ then

1. all $v \in U$ have the same label

2. U is directed with respect to $\hookrightarrow$
3. if $h = 1$ then U contains exactly the root vertices of all the trees T_i
4. if $h > 1$ then $\mathit{parent}(U) \subseteq U'$ for some $U' \in V_{h-1}/_{\simeq_{h-1}}$ and
   (a) either all vertices in U' have at most one child in U or
   (b) every $v \in U$ is contained in a proper infinite chain $C \subseteq U$ and for every finite subset $V \subseteq U$ there exists $v' \in U'$ such that $V \hookrightarrow_1 \mathit{Children}(v') \cap U$.

Now let $\simeq$ be the union of all the relations $\simeq_h$. Then $\simeq$ is an equivalence relation on V that partitions V into finitely many equivalence classes. For an equivalence class $U \in V/_{\simeq}$, let $C(U)$ be the set of all equivalence classes that contain children of vertices in U. Furthermore, let $l(U)$ be the unique label of all vertices in U, and let $m(U)$ denote 1 if every parent of a vertex $v \in U$ has at most one child in U and, otherwise, let $m(U)$ denote the symbol $+$. Then define a finite hedge automaton $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ as follows:

- $Q = V/_{\simeq}$
- $\Sigma = L_k$.
- $Q_f = V_1/_{\simeq}$
- Δ consists of transition rules of the following form for each $U \in V/_{\simeq}$
  - $l(U)(U_1^{m(U_1)} \cdots U_n^{m(U_n)}) \to U$ if $C(U) = \{U_1, \dots, U_n\}$
  - $l(U)(\epsilon) \to U$ if $C(U) = \emptyset$.

Let T_U be a tree labelled by L_k and r a run of $\mathcal{A}$ on T_U. We show by induction on the height of T_U that if $r(w) = U$ for the root w of T_U then there exists $v \in U$ such that $T_U \hookrightarrow T(v)$.

If $h = 1$ then T_U consists of a single root vertex w that is a leaf. Then the transition rule in Δ used to label w in r is of the form $l(U)(\epsilon) \to U$. Thus, by construction of $\mathcal{A}$ all trees $T(v)$ for vertices $v \in U$ consist of the single leaf vertex v labeled by $l(U)$, i.e., $T_U \hookrightarrow T(v)$ for all $v \in U$.

If $h > 1$ then the transition rule in Δ used to label w must have the form

$$l(U)(U_1^{m_1} \cdots U_n^{m_n}) \to U$$

with $C(U) = \{U_1, \dots, U_n\}$ and $m_i = m(U_i)$ for all $1 \leq i \leq n$. Let

$$T_{1,1}, \dots, T_{1,r_1}, \dots, T_{n,1}, \dots, T_{n,r_n}$$

be the subtrees of T_U rooted at the children of w such that r labels the root of each tree $T_{i,j}$ by U_i. These trees have height $h - 1$ and r is a run of $\mathcal{A}$ on each of these trees. Thus, by induction hypothesis there exist vertices

$$v_{1,1}, \dots, v_{1,r_1} \in U_1 \ \dots \ v_{n,1}, \dots, v_{n,r_n} \in U_n$$

with $T_{i,j} \hookrightarrow T(v_{i,j})$ for all $1 \leq i \leq n$, $1 \leq j \leq r_i$. If two vertices $v_{i,j}$ and $v_{i',j'}$ coincide then we must have $i = i'$. Thus, $r_i > 1$ and $m(U_i) = +$, i.e., by construction of $\mathcal{A}$, there are vertices in U that have more than one child in U_i. Then U_i satisfies property 4.(b) of the relations $\simeq_h$, i.e., U_i contains a proper infinite chain C with $v_{i,j} \in C$. Hence, we can choose two vertices $v'_{i,j}, v'_{i',j'} \in C$

that are (1) distinct, (2) disjoint from all other $v_{i,j}$, and (3) satisfy $T_{i,j} \hookrightarrow T(v'_{i,j})$ and $T_{i,j} \hookrightarrow T(v'_{i',j'})$. Therefore, without loss of generality assume that all the $v_{i,j}$ are distinct. Now for any $1 \leq i \leq n$ we can find $v_i \in U$ such that

$$\{v_{i,1}, \dots, v_{i,r_i}\} \hookrightarrow_1 \mathit{Children}(v_i) \cap U_i$$

Namely, if $r_i = 1$ then $v_i = \mathit{parent}(v_{i,1})$ and if $r_i > 1$ then such v_i exists by property 4.(b). Now, using the fact that U is directed we can inductively construct an upper bound $v \in U$ of all the v_i with respect to the wqo $\hookrightarrow$. Then we have by construction:

$$\{v_{1,1}, \dots, v_{1,r_1}, \dots, v_{n,1}, \dots, v_{n,r_n}\} \hookrightarrow_1 \mathit{Children}(v)$$

We conclude that $\mathit{Children}(w) \hookrightarrow_1 \mathit{Children}(v)$ and $l(v) = l(U)$, i.e., $T_U \hookrightarrow T(v)$, which concludes the induction proof.

From the proved statement follows

$$\forall T \in \mathcal{L}(\mathcal{A})\, \exists i \in \mathbb{N} :\ T \hookrightarrow T_i \tag{2}$$

Using a similar inductive proof we can show that for all equivalence classes $U \in V/_{\simeq}$ and $v \in U$ there exists a tree T_U and a run r of $\mathcal{A}$ on T_U such that $T(v) \hookrightarrow T_U$. From this follows

$$\forall i \in \mathbb{N}\, \exists T \in \mathcal{L}(\mathcal{A}) :\ T_i \hookrightarrow T \tag{3}$$

Note that by construction of $\mathcal{A}$ the tree encoding operation can be reversed on the trees in $\mathcal{L}(\mathcal{A})$. Let $D_{\mathcal{A}}$ be the corresponding set of configurations. From Properties (2) and (3) follows that $D = {\downarrow}D' = {\downarrow}D_{\mathcal{A}}$. From $\mathcal{A}$ we can now easily construct an extended process term E whose denotation is the downward closure of $D_{\mathcal{A}}$. It follows that $D = {\downarrow}D_{\mathcal{A}} = \gamma(E)$.

For proving the other direction of the proposition we start from an extended process term E. Then from E we can again easily construct a finite hedge automaton $\mathcal{A}$ of the same form as above, such that the tree encoding operation can be reversed on all trees accepted by $\mathcal{A}$ and the downward-closure of the resulting configurations $D_{\mathcal{A}}$ coincides with $\gamma(E)$. Using a simple pumping argument on the hedge automaton $\mathcal{A}$ we can show that for every two trees $T_1, T_2 \in \mathcal{L}(\mathcal{A})$ there exists a tree $T \in \mathcal{L}(\mathcal{A})$ such that $T_1 \hookrightarrow T$ and $T_2 \hookrightarrow T$. It follows that $D_{\mathcal{A}}$ is directed and thus $\gamma(E)$.

□

A.8 Proof of Theorem 19

Theorem 19. *The reachability problem for depth-bounded processes is undecidable.*

Proof. Let $N = (Pl, T, F, M_0)$ be a reset net with places Pl, transitions T, flow function $F : (Pl \times T) \cup (T \times Pl) \rightarrow \mathbb{N} \cup Pl$, and initial marking M_0 given by a multiset over Pl. We build a depth-bounded process $\mathcal{P}$ that simulates the transitions of N. In each reachable configuration of $\mathcal{P}$ there is only one thread $P(\boldsymbol{x})$. This thread simulates a linearized execution of the reset net N. Each of the r places in N simply corresponds to a channel at a dedicated position in $\boldsymbol{x} = x_1, \ldots, x_r$. We use a dedicated process identifier A_0 that indicates that the thread in the given configuration of $\mathcal{P}$ is stable and can fire the next transition of N. Let θ be the function that maps a name vector $\boldsymbol{x}$ of length r and a place $p \in Pl$ to the name in $\boldsymbol{x}$ at the position dedicated to p. If a place p has n tokens in a marking M then the corresponding configuration has n messages of the form $\overline{\theta(\boldsymbol{x}, p)}()$. Thus, the initial configuration I of $\mathcal{P}$ is given by:

$$(\nu \boldsymbol{x})(A_0(\boldsymbol{x}) | \prod_{p \in M_0} \overline{\theta(\boldsymbol{x}, p)}())$$

For encoding the transitions of N we further assume process identifiers A_t for every transition $t \in T$ that indicate whether the thread has consumed all tokens of places in the preset of t according to the flow function F and is now ready to produce all the tokens in the postset of t. Finally, we have auxiliary process identifiers of the form $A_{(t,p,i)}, A_{(t,p)}$ and $A_{(p,t,i)}, A_{(p,t)}$ that are used for the linearization of token consumption and generation. Let $t \in T$ and let $\bullet t = \{p_1, \ldots, p_n\}$. For encoding the consumption of all tokens when t is fired we have the following equations in $\mathcal{P}$: for each p_i if $F(t, p_i) = k \in \mathbb{N}$ is a normal arc we have for all $1 \leq i \leq n$ and $0 \leq j \leq k$ equations

$$A_{(p_i,t,j)}(\boldsymbol{x}) = \theta(\boldsymbol{x}, p_i)().A_{(p_i,t,j+1)}$$

where $A_{(p_0,t,0)} = A_0$, $A_{(p_i,t,0)} = A_{(p_{i-1},t)}$ for $i > 1$ and, similarly, $A_{(p_n,t,k)} = A_t$, and $A_{(p_i,t,k)} = A_{(p_i,t)}$ for $i > 1$. If instead $F(t, p_i) = p_i$ is a reset arc, we have an equation

$$A_{(p_{i-1},t)}(\boldsymbol{x}) = (\nu(y))(A_{(p_i,t)}(x_1, \ldots, x_{j-1}, y, x_{j+1}))$$

where $\theta(\boldsymbol{x}, p_i) = x_j$ and $A_{(p_0,t)} = A_{p_n,t} = A_0$. Similarly we have equations for all $t\bullet = q_1, \ldots, q_m$ that produce new tokens for each q_i by sending $F(t, p_i)$ messages to the channel associated with q_i.

Note that the number of threads in each reachable configuration of $\mathcal{P}$ is constant. Since the communication topologies of configurations are bipartite graphs of vertices for names and vertices of threads, it follows that the tree-depth of all reachable configurations of $\mathcal{P}$ is bounded. In particular, the dead names resulting from the simulation of fired transitions with reset arcs are disconnected from the single thread vertex and, thus, do not increase the tree-depth.

A marking M of N and a configuration $C = (\nu \boldsymbol{x})(A_0(\boldsymbol{x}) | P)$ of $\mathcal{P}$ are *equivalent* iff for each $p \in M$ there is a corresponding message $\overline{\theta(\boldsymbol{x}, p)}()$ in P. By construction, a marking of N is reachable in N iff an equivalent configuration is reachable in $\mathcal{P}$. Thus, it follows from [6] that the reachability problem for depth-bounded systems is undecidable. □