Selected Techniques for Verification of Infinite-State Systems

Jiří Srba

PhD Dissertation

Faculty of Informatics
Masaryk University
Czech Republic

Selected Techniques for Verification of Infinite-State Systems

A Dissertation
Presented to the Faculty of Informatics
of the Masaryk University in Brno
in Partial Fulfillment of the Requirements for the
PhD Degree

by
Jiří Srba
December 14, 2004

Abstract

The aim of this thesis is to provide a study of selected models that generate infinite-state spaces with a focus on fundamental questions about their automatic verification. We study several verification techniques and identify decidability border-lines for bisimilarity notions over asynchronous transition systems and then we focus on decidability issues for recursive and replicative ping-pong protocols.

Many types of reactive systems can be described (at a certain abstraction level) in a finite way. However, the behaviour of such systems is not always finite (bounded). For example context-free grammars provide a finite description of systems, which can be understood as models of reactive sequential processes with recursion. A process represented in the syntax of context-free grammars can generate a labelled transition system with infinitely many reachable states and verification questions about such processes become non-trivial both with respect to decidability and complexity issues.

We begin the thesis with a short survey presenting the classical results and techniques used in equivalence checking of infinite-state systems. On a semi-formal level we describe several notions of behavioral equivalences (in particular bisimilarity) and mention the most prominent results in the theory, mainly for the classes of context-free processes, commutative context-free processes and their state-extended superclasses. We conclude the first chapter by presenting an overview of selected decidability and complexity results, including the recent achievements characterizing the undecidability levels of problems which cannot be automatically verified. In the following chapter we provide a more focused introduction to the problems studied in the thesis, namely the (un)decidability issues of bisimilarity notions which take into account causal relationships between events (actions) of concurrent systems and the questions of automatic verification of simple cryptographic protocols. The first part of the thesis is concluded by an overview of author's contribution and bibliographical summary.

In the second part of the thesis (consisting of three chapters) we give a detailed description of the achieved results. First, we show undecidability of hereditary history preserving bisimilarity for unfoldings of finite asynchronous transition systems and strengthen the results to even unlabelled systems (i.e. systems where the action alphabet is a singleton set). Next two chapters are devoted to formalization of simple process algebra (with recursion and replication) reflecting the behaviour of basic cryptographic protocols, namely ping-pong protocols. Several decidability questions about the algebra are answered, providing

both positive and negative results and reusing some of the well established techniques from the classical area of infinite-state systems. The most interesting results include the undecidability of the recursive calculus (including reachability and equivalence checking problems) where three and more parties participate in the protocol. The result is valid even if no explicit non-deterministic choice operator is considered in the calculus. On the other hand, the problem with two participants only is shown to be tractable and the reachability problem is proven decidable in polynomial time. We shall also demonstrate a technique for description of active attacks on the protocol, using only the limited syntax available. Finally, a replicative variant of the calculus is studied and the reachability problem is shown decidable by reducing it to reachability in weak process rewrite systems.

Acknowledgments

I would like to thank to my adviser Mojmír Křetínský for his constant support and encouragement. In particular, I am very grateful for his excellent introduction to the "formal world" and for his understanding.

I would also like to express my gratitude to Ivana Černá, my informal adviser in the beginning of my PhD study, for her patience and for the fruitful discussions we had.

My warm thanks go to all staff members at the Faculty of Informatics in Brno, in particular to Luboš Brim and Tony Kučera, and to the members of the ParaDiSe laboratory.

Last but not least, I am very glad to acknowledge a support from BRICS PhD School in Aarhus, where I spent a significant part of my PhD studies under the supervision of Mogens Nielsen.

Jiří Srba
Aalborg, December 14, 2004.

Contents

Part I

Overview

Chapter 1

Introduction

Formal methods are increasingly used for verification of safety-critical and other systems in many different areas like e.g. hardware construction or software validation (ranging from aerospace and defence to transportation and communication).

Automatic reasoning about systems which behaviour can be at a certain abstraction level described as a finite-state one is a well established field in computer science with a wide range of areas including finite-state machines [17, 18, 39, 53, 79], data structures like BDDs (binary decision diagrams) [9, 10, 44] and techniques like partial order reduction [20, 55, 73].

On the other hand, when systems are modelled at an abstraction level which includes features like unbounded data domains (e.g. counters, integer variables, lists, stacks, trees, real-time parameters) and complex control-flow structures (e.g. recursive procedure calls, dynamic process creation, mobility), the state-spaces can be infinite or even uncountable. For such systems new methodology has to be proposed and the limits of automatic verification carefully inspected.

In this chapter we provide an informal introduction to the study of infinite-state systems with a particular focus on decidability and complexity issues.

1.1 Infinite-State Systems

In his Turing Award lecture [22], Juris Hartmanis eloquently discusses, among other things, the fundamental rôle that *computational complexity* theory plays in computer science. He goes on, in the context of describing joint work with Phil Lewis and Richard Stearns, to highlight some of the results obtained on the computational complexity of problems in formal language theory; e.g., all context-free languages are contained in TIME[n^3] and SPACE[$\log^2 n$].

We argue here that the computational complexity of *generative devices* such as grammars or automata takes on a new and interesting light when such devices are interpreted as generating (concurrent) *processes* rather than formal languages, and the traditional notion of language equivalence is replaced by some notion of *semantic equivalence*. When employing grammars to generate processes, we assume them to be in Greibach Normal Form (GNF). In this way, a state of a process corresponds to a sequence of nonterminals; and the

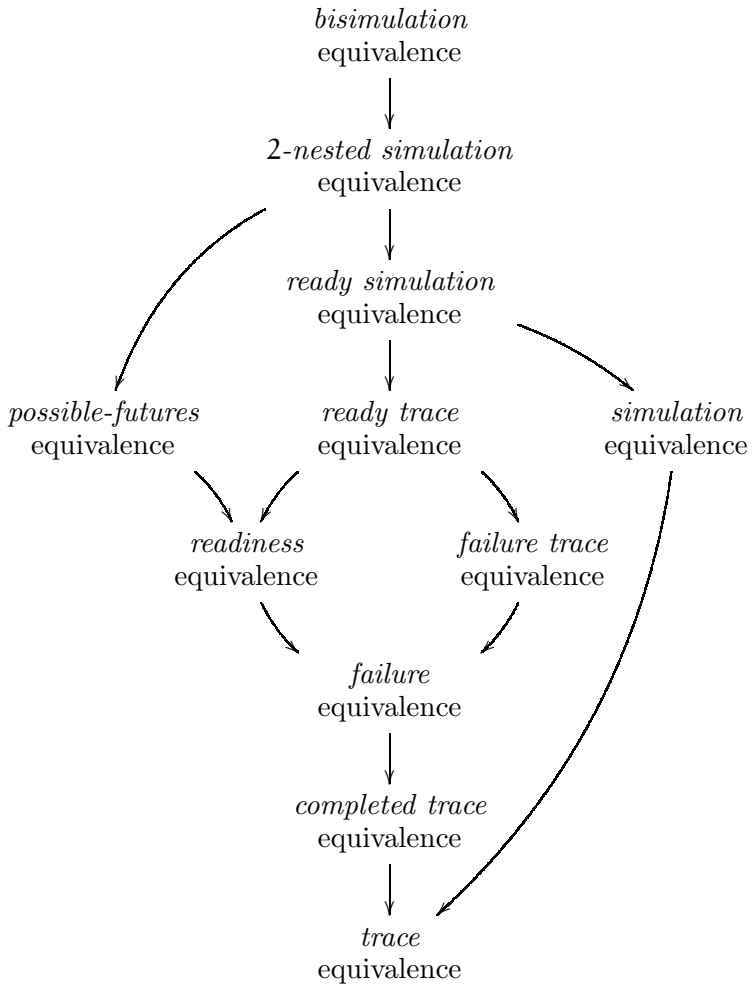


Figure 1.1: Linear/branching time spectrum

transitions leading from a state corresponding to a sequence starting with a nonterminal X are prescribed, in a one-to-one fashion, by the rules of the grammar corresponding to the nonterminal X. Formally, $X\beta \xrightarrow{a} \alpha\beta$ if $X \to a\alpha$ is a rule of the grammar. In concurrency theory such a transition is read as "process $X\beta$ performs the action a and evolves into the process $\alpha\beta$".

A wide range of semantic equivalences was classified by van Glabbeek [75,76] in his linear time/branching time spectrum (see Figure 1.1). The coarsest (least discriminating) equivalence in this hierarchy is *trace equivalence*, as defined by Hoare [28]. A (partial) trace of a process is a finite sequence of actions that can be performed by the process. Two processes are trace equivalent if their sets of traces are equal.

A variant of trace equivalence is called *completed trace equivalence*, which in the theory of formal languages and automata is known as *language equivalence* (provided that infinite computations are disregarded). A completed trace is maximal in the sense that it cannot be extended to a longer one. Two processes

are completed trace equivalent if they are trace equivalent and moreover they have the same set of completed traces.

As we go further up in van Glabbeek's spectrum, the equivalences distinguish more and more branching features. The finest (most discriminating) equivalence is *bisimulation equivalence*, or *bisimilarity*. This is perhaps *the* equivalence that has attracted the most attention in concurrency theory, and is also the main focus of this paper. As originally introduced by Park [54] and Milner [45], bisimilarity appeared to play a prominent rôle due to many pleasant properties it possesses.

Bisimulation equivalence is the cornerstone of a number of theories of concurrent and distributed computing, most notably Robin Milner's Calculus of Communicating Systems (CCS) [47] and the π-calculus [49]. Milner received the 1991 Turing Award, and bisimulation figured prominently in his Turing Award lecture [48].

The idea underlying bisimulation equivalence had also drawn the attention of modal logicians already 30 years ago, in the guise of p-morphisms [58], and later zig-zag relations [74]. It is intimately related to the distinguishing powers of general branching-time temporal logics, in particular the modal mu-calculus. Set theorists have been attracted to bisimulation, as it forms the basis of Peter Aczel's Anti Foundation Axiom for non-well-founded set theory [7]. The functional programming community has also shown interest in bisimulation, as evidenced by Samson Abramsky's notion of *applicative bisimulation* for relating terms of the lazy lambda calculus [1].

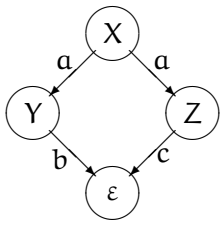
The essence of bisimilarity, quoting Hennessy and Milner [23], "is that the behaviour of a program is determined by how it communicates with an observer." Therefore, the notion of bisimilarity for different models is defined in terms of their *behaviours* and *observable behaviours*. For example for rooted labelled transition systems it seems natural to identify their behaviours with (possibly infinite) synchronization trees [45] into which they unfold, and to take sequences of actions as observations. The abstract definition of bisimilarity for arbitrary categories of models due to Joyal, Nielsen and Winskel [38] formalizes this idea. Given a category of models where objects are behaviours and morphisms indicate how one behaviour extends the other, and given a subcategory of observable behaviours, the abstract definition yields a notion of bisimilarity for all behaviours with respect to observable behaviours. For example, for rooted labelled transition systems, taking synchronization trees as their behaviours, and sequences of actions as the observable behaviours, we recover the standard notion of strong bisimilarity.

Another abstract definition of bisimilarity is that based on coalgebras. Transition systems of various kinds can be viewed as coalgebras for appropriate endofunctors. This approach gives rise to a definition of bisimulation as a span of coalgebras [57, 72].

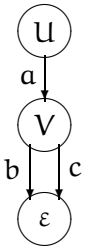
A remarkable property of bisimilarity is its computational feasibility. Bisimilarity is widely regarded as the "most decidable" behavioural equivalence, and this aspect will be demonstrated in the rest of this paper.

Finally, as we shall show, bisimulation has an elegant game-theoretic interpretation as promulgated by Stirling [66] and Thomas [71].

To motivate this study, consider the two regular expressions $\mathsf{ab} + \mathsf{ac}$ and $\mathsf{a}(\mathsf{b} + \mathsf{c})$ which are represented by the following nondeterministic finite-state automata (NFA) and corresponding regular grammars.



$\mathsf{X} \to \mathsf{aY}$
$\mathsf{X} \to \mathsf{aZ}$
$\mathsf{Y} \to \mathsf{b}$
$\mathsf{Z} \to \mathsf{c}$



$\mathsf{U} \to \mathsf{aV}$
$\mathsf{V} \to \mathsf{b}$
$\mathsf{V} \to \mathsf{c}$

These expressions, as well as their corresponding automata, are clearly language equivalent, as they both describe the language $\{\, \mathsf{ab},\ \mathsf{ac} \,\}$; as language generators, they are indistinguishable. However, viewed as process generators they may be distinguished. The first automaton in its initial state X may perform an a-transition and evolve into either state Y—from which only a b-transition is possible—or state Z—from which only a c-transition is possible; on the other hand, the second automaton in its initial state U performs the a-transition and evolves into state V from which both the b- and c-transitions are possible.

If we interpret these automata as representing the behaviours of processes, with the transitions being potential communications with the environment in which these processes reside, then it behooves us to consider them as behaviourally *in*equivalent. For example after the initial communication involving the a-transition, the second process would be in state V from which it is willing to participate in a communication involving a b-transition, whereas the first process may be in state Z from which it will refuse to participate in a communication involving a b-transition. In the terminology of concurrency theory, the first process may *deadlock* in an instance in which the second will not.

Milner [45] proposed bisimilarity to formally capture the notion of behavioural equivalence, and gave it, along with Park [54], a simple and elegant mathematical definition in terms of bisimulations. A *(strong) bisimulation* is a binary relation $\mathcal{R}$ on processes such that whenever $\mathcal{R}(\mathsf{P}, \mathsf{Q})$: if P can perform an a-transition to become P' (for any a and P'), then Q can also perform an a-transition to become some Q' such that $\mathcal{R}(\mathsf{P}', \mathsf{Q}')$; and conversely, if Q can perform an a-transition to become Q', then P can also perform an a-transition to become some P' such that $\mathcal{R}(\mathsf{P}', \mathsf{Q}')$. Note the recursive nature of the definition. Now, two processes P and Q are *bisimilar* if there exists a bisimulation $\mathcal{R}$ such that $\mathcal{R}(\mathsf{P}, \mathsf{Q})$. It is well-known that bisimulations are closed under union and that the largest bisimulation, under set inclusion, exists. In fact, this largest bisimulation, $\sim$, is an equivalence relation and taken to be bisimulation equivalence, or simply (strong) bisimilarity.

It is instructive to view bisimulation equivalence in terms of particular two-player games. A *game* is provided by a pair of processes (P, Q), with the players alternating moves as follows: Player I (also called Attacker or Spoiler) chooses a sequence of transitions of one of the processes, and in response, Player II (also called Defender or Duplicator) must choose an identically labelled sequence of transitions of the other process. The game then continues starting from the

resulting pair of processes. If Player II ever finds that she cannot respond to a move made by Player I, she loses the game. On the other hand, if Player I cannot perform any transition from either of the processes, the Player II wins. If the game is infinite, Player II is the winner. A moment's reflection then leads to the realization that Player II has a defending strategy exactly when the two processes are bisimilar: if there is a bisimulation relating the processes, then a defending strategy for Player II consists of merely matching transitions made by Player I which lead to a resulting pair which is also contained in the bisimulation relation. Conversely Player I has a winning strategy exactly when the two processes are not bisimilar.

As an example, the two processes X and U pictured above are not bisimilar. (There is no bisimulation relating X and U.) This nonbisimilarity is evidenced by the existence of an obvious winning strategy for Player I in the game defined by the pair (X, U). After one exchange of moves consisting of a single a-transition, the game must be in either the configuration (Y, V) or (Z, V). In the first instance, Player I may win by choosing the single c-transition from process V, while in the second instance she may win by choosing the single b-transition from process V. Thus, bisimilarity is a strictly more discriminating equivalence relation than language equivalence, and is intrinsically sensitive (unlike language equivalence) to the nondeterministic branching structure of processes.

For some applications the notion of bisimilarity is too strong because it reflects not only the *visible (observable) actions* but also the *internal (unobservable) actions*. This means that two processes have to exhibit bisimilar behaviours including, e.g., internal synchronization. It is, however, often not desirable to observe such internal events (e.g. implementation hiding in software-engineering). For this reason a special *silent action*, generally denoted by τ, is introduced with the intention that the action τ should be undetectable by an external observer.

Several semantic approaches can differ in the way they treat the unobservable action τ. One possibility is to disregard τ actions and agree that only the visible actions are observable. Citing Milner [47]: "... we merely require that each τ action is matched by *zero* or more τ actions ...". The notion of bisimilarity achieved this way is called *weak bisimilarity*.

In order to define weak bisimilarity one usually introduces the so-called *weak transition relation*. The idea is that a process P performs under the weak transition relation an action a and evolves into a process Q whenever it is possible to perform from P zero or more τ actions and then the action a, followed again by zero or more τ actions, finally resulting in Q. We also allow that P under the weak transition relation performs the τ action and evolves into P again.

Weak bisimilarity then corresponds to the bisimilarity notion defined above where instead of the basic transition relations we use the weak transition relations. In the same manner one can also generalize the bisimulation game described above.

Let us also mention that in [77] van Glabbeek and Weijland introduced a finer notion of behavioural equivalence than weak bisimilarity called *branch-*

ing bisimilarity. Their approach builds on the ideas of weak bisimilarity but it moreover distinguishes between processes that change their branching properties after the performance of individual τ-actions. This in particular means that if a τ-action is performed by one of the processes then the other process not only has to match this move by a sequence of τ actions but also all the intermediate states reached during this sequence have to be branching bisimilar to the first process.

For completeness let us mention that at least two other behavioural equivalences that abstract away from unobservable actions, called *eta bisimilarity* [3] and *delay bisimilarity* [46], have been proposed. They treat abstraction from unobservable actions in a slightly different way than branching bisimilarity and are positioned between weak and branching bisimilarity, mutually incomparable.

Since this paper deals only with strong and weak bisimilarity, we shall not provide further details about the other notions of bisimilarity. The interested reader is referred to [78].

We now have in place the three main ingredients of a formal language theory in a new setting: automata and grammars (processes), and equivalence (bisimilarity). The computational complexity of bisimulation in this formal-language framework, however, differs greatly from its classical counterpart, with a number of surprising twists and turns worth mentioning. We concentrate here on the inner layers of the Chomsky hierarchy, viz. *regular* and *context-free* processes, and note in passing that a language like CCS is easily shown to be Turing-powerful.

1.2 Regular Processes

In the case of regular processes, that is, those given by right-linear GNF grammars such as the two depicted above, the main complexity result is as follows. Let P,Q be regular processes whose underlying NFA have a total of n states and m transitions. Then, as was shown by Kanellakis and Smolka [39], whether or not P and Q are bisimilar can be decided in polynomial time, $O(nm)$ time to be exact. This algorithm was subsequently improved upon by Paige and Tarjan who devised one that runs in $O(m \log n)$ time [53]. This is in stark contrast to the equivalence problem for regular expressions, which was shown to be PSPACE-complete [29]. The class of regular processes is usually denoted by FS, to emphasize the intrinsically finite-state nature of these processes.

Moreover, bisimulation was originally defined by Milner as the limit of a sequence of successively finer equivalence relations, $\sim_k$, where $\sim_1$ is trace equivalence. In terms of our game-theoretic characterisation, two processes are related by $\sim_k$ exactly when Player II has a winning strategy if the game is redefined to declare her the winner after the exchange of k moves. So, for example, the above processes X and U are related by $\sim_1$ but not by $\sim_2$ as Player I has a strategy for guaranteeing a win within the exchange of two moves. Kanellakis and Smolka showed that, for each fixed k, deciding $\sim_k$ is PSPACE-complete, a complexity that disappears in the limit; i.e., upon reaching $\sim$.

As for weak bisimilarity on regular processes, one can first pre-compute the weak transition relation (which simply amounts to computing of the transitive closure) and construct new regular processes where transitions are replaced with weak transitions. On these new regular systems the algorithms for (strong) bisimilarity checking can be used. Hence the problem for weak bisimilarity can also be decided in polynomial time.

1.3 Context-Free Processes

The situation is even more dramatic in the context-free case, where the resulting processes are no longer regular. In the concurrency theory community, context-free processes are referred to as BPA (Basic Process Algebra) processes. In the classical setting, Bar-Hillel, Perles, and Shamir [6] showed that the equivalence problem for languages generated by nondeterministic context-free grammars is undecidable. In fact all the equivalences in van Glabbeek's spectrum (apart from bisimilarity) are undecidable for BPA [21, 31].

Taking advantage of the periodic structure exhibited by bisimilar processes, Baeten, Bergstra, and Klop [2] were able to show that bisimilarity of *normed* BPA—those context-free processes in which the underlying GNF grammar contains no redundant nonterminals—is decidable. (Being normed means that there is a sequence of transitions leading from any state to the state ε. The norm of a state is defined as the length of the shortest such sequence.) In fact, Hirshfeld, Jerrum and Moller [26] showed that, in this case, bisimilarity can be decided in polynomial time. Restricting to simple (i.e., deterministic) normed grammars, where language equivalence and bisimilarity coincide, this gives that language equivalence is polynomially decidable, improving vastly on the doubly-exponential algorithm of Korenjak and Hopcroft [41].

For arbitrary (unnormed) BPA processes, Christensen, Hüttel, and Stirling [16] showed that bisimilarity is still decidable. However, the complexity in this general case is now known to be PSPACE-hard [63], yet no worse than doubly-exponential [12].

Decidability of weak bisimilarity checking for BPA is still an open problem. It is generally conjectured that the problem is decidable but so far only a partial positive result for a restricted subclass of totally normed BPA was achieved by Hirshfeld in [25]. However, the problem was very recently shown to be at least EXPTIME-hard by Mayr [43], even for normed BPA.

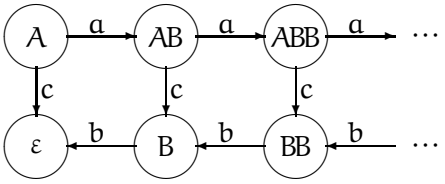
1.4 Commutative Context-Free Processes

Of course, in studying concurrent processes one would like to consider processes composed not merely sequentially as with context-free processes, but concurrently as well. A simple form of concurrent composition can be modelled by considering commutative context-free processes; that is, where we now interpret concatenation of nonterminals modulo commutativity. In this way, any nonterminal in a sequence can be used to provide the next transition from the state associated with that sequence. In the concurrency theory community, the

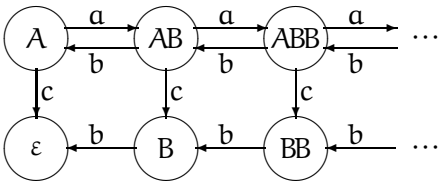
resulting process is referred to as BPP (Basic Parallel Process). For example, the grammar

$$A \to aAB \qquad A \to c \qquad B \to b$$

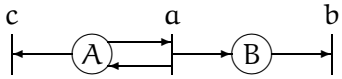
gives rise to the BPA (context-free) process



and to the BPP (commutative context-free) process



BPP processes correspond to communication-free Petri nets, in other words those (place/transition) Petri nets in which each transition has a unique input place. For example, the above BPP process corresponds to the following Petri net:



The results regarding deciding bisimulation in this case are similar to those for BPA processes. Hirshfeld [24] showed that once again language equivalence is undecidable (in fact none of the equivalences from van Glabbeek's spectrum below bisimilarity are decidable [30]), while Christensen, Hirshfeld and Moller [14] showed that bisimilarity is decidable in general, and Hirshfeld, Jerrum and Moller [27] showed that it is decidable in polynomial time for normed processes. The problem for unnormed BPP is known to be PSPACE-hard [62] and Jančar [34] has recently demonstrated that it can indeed be decided in polynomial space. PSPACE-completeness of the problem is hence established.

One noteworthy corollary from Jančar's paper is the resolution of an intriguing long-standing conjecture. In the case of BPA processes, if X is an unnormed variable it is easily confirmed that $X \sim X\alpha$ for any α; being unnormed, the process represented by the nonterminal X can never terminate, so the behaviour of $X\alpha$ will be the same as that of X. Also, if $XX \sim XXX$ then the variable X must be unnormed; this follows from the fact that the norm is additive, and bisimilar processes must have the same norm. Thus it is clear that the identity $X \sim XX$ follows immediately from $XX \sim XXX$. However, this is by no means obvious in the case of BPP processes. This conjecture was put forward more than a decade ago (a stronger version appears in [15]), and since then many clever researchers have failed at every attempt to prove this cancellation law; none of the standard bisimulation proof techniques could be applied to this question.

Jančar finally provided a proof which is unexpectedly complicated for such a simple-looking conjecture.

Even though the weak bisimilarity problem for BPP is still open, Jančar conjectured that his new technique from [34] might be extended to prove decidability of this problem. His conjecture is also partially confirmed by positive decidability results of weak bisimilarity for several subclasses of BPP [25, 69].

1.5 State-Extended Processes

A common extension to context-free and commutative context-free processes is provided by including a finite-state control unit. With such an extension, the grammar rules are no longer based solely on a nonterminal from the sequence representing the state, but are dictated as well by the finite-state control. State-extended BPA naturally correspond to pushdown automata (PDA), with the nonterminal sequence representing the stack, while state-extended BPP correspond to multiset automata (MSA), which are sometimes referred to as PPDA for parallel pushdown automata, and represent a subclass of Petri nets.

Sénizergues [59] and Stirling [67] both showed the decidability of bisimulation equivalence over state-extended BPA. Another noteworthy result is the result of Stirling [68] that bisimilarity is decidable over strict deterministic grammars. This result reinforces (and gives a shorter proof for) Sénizergues's solution [60] to the long-standing equivalence problem for deterministic pushdown automata. Recently Stirling proved that this problem is primitive recursive [70].

For the case of state-extended BPP, the result differs from the sequential case; here bisimilarity was proved undecidable [50] using Jančar's technique for the undecidability of bisimilarity of Petri nets [33].

Very recently the weak bisimilarity problem for PDA was shown to be undecidable [64], and it was proved that weak bisimilarity of Petri nets and MSA is significantly harder than strong bisimilarity. In fact, the weak bisimilarity problems are highly undecidable both for PDA and MSA [65] (Σ^1_1-complete in the analytical hierarchy), which contrasts to decidability of strong bisimilarity for PDA [59, 67] and to Π^0_1-completeness (first level of the arithmetical hierarchy) of strong bisimilarity for Petri nets and MSA (see [32]).

1.6 Process Rewrite Systems

Based on the models of infinite-state systems introduced above, a successful effort to provide a common framework for their analysis was started by Moller in [50], and a slightly generalized and simplified version of this formalism was presented by Mayr [42] in the form of *Process Rewrite Systems* (PRS).

In the PRS formalism processes are identified with *process expressions* which consist of atomic *process constants* combined into larger process expressions by means of *sequential* and *parallel* operators. Formally the class of general process expressions $\mathcal{G}$ is defined by the following abstract syntax

$$\mathsf{E} ::= \varepsilon \mid \mathsf{X} \mid \mathsf{E}.\mathsf{E} \mid \mathsf{E}|\mathsf{E}$$

where 'ε' denotes the *empty process*, X ranges over a given set of process constants, and '.' and '|' are the operators of sequential and parallel composition, respectively. Moreover, we assume that '.' is associative, '|' is associative and commutative, and 'ε' is a unit for '.' and '|'.

A *process rewrite system* is a finite set Δ of rewrite rules of the form $\mathsf{E} \xrightarrow{a} \mathsf{E}'$ such that E and E' are from $\mathcal{G}$, and a is from a given set of actions. This finite set of rewrite rules generates an infinite-state process by means of the following inference rules (recall that '|' is commutative).

$$\frac{(\mathsf{E} \xrightarrow{a} \mathsf{E}') \in \Delta}{\mathsf{E} \xrightarrow{a} \mathsf{E}'} \qquad \frac{\mathsf{E} \xrightarrow{a} \mathsf{E}'}{\mathsf{E}.\mathsf{F} \xrightarrow{a} \mathsf{E}'.\mathsf{F}} \qquad \frac{\mathsf{E} \xrightarrow{a} \mathsf{E}'}{\mathsf{E}|\mathsf{F} \xrightarrow{a} \mathsf{E}'|\mathsf{F}}$$

The intuition behind the combinations of parallel and sequential operators on the left- and right-hand sides of the rewrite rules is as follows. A rewrite rule of the form $\mathsf{X} \xrightarrow{a} \mathsf{Y}$ can be interpreted as "process X performs the action a and becomes process Y." Similarly a rewrite rule of the form $\mathsf{X} \xrightarrow{a} \varepsilon$ means that "process X performs the action a and terminates." The interpretation of the rewrite rule $\mathsf{X} \xrightarrow{a} \mathsf{Y}.\mathsf{Z}$ is "process X calls a procedure Y and then continues as process Z". When the sequential operator is present on the left-hand side of a rewrite rule as in $\mathsf{X}.\mathsf{Y} \xrightarrow{a} \mathsf{Z}$, the intuition is that we enable value passing: X represents a value returned by some previous computation and the behaviour of the process Y is affected by this value.

We now proceed to discuss the parallel operator '|'. A rewrite rule of the form $\mathsf{X} \xrightarrow{a} \mathsf{Y}|\mathsf{Z}$ stands for "process X performs the action a and becomes a parallel composition of processes Y and Z." In other words, process X forks into Y and Z. Rewrite rules of the form $\mathsf{X}|\mathsf{Y} \xrightarrow{a} \mathsf{Z}$ are interpreted as "processes X and Y synchronize by jointly executing the action a and becoming the process Z."

Finally, in the most general cases, we allow process expressions to contain a mixture of sequential and parallel operators on both sides of the rewrite rules, as in the rule

$$\mathsf{X}.\mathsf{Y} \xrightarrow{a} (\mathsf{U}|\mathsf{V}).\mathsf{Z}.$$

This particular rule can be interpreted as follows: "process Y receives a return value X and performs the action a; after this, a parallel execution of U and V is initiated; finally, when both of these parallel components terminate, the computation continues with the execution of the process Z."

These simple examples demonstrate why process rewrite systems find a natural application in the interprocedural control-flow analysis of programs [40]. More details can be found, e.g., in [19].

By restricting the general form of the rewrite rules we obtain several subclasses of process rewrite systems. Let $\mathcal{S}$ (sequential process expressions) represent the subclass of general process expressions $\mathcal{G}$ that contains no parallel operator. Also, let $\mathcal{P}$ (parallel process expressions) represent the subclass of general process expressions $\mathcal{G}$ that contains no sequential operator. Let $\mathsf{1}$ be the class that contains only process constants and the empty process. Then, e.g., $(\mathsf{1}, \mathcal{P})$-PRS is the subclass of PRS where every rule $\mathsf{E} \xrightarrow{a} \mathsf{E}'$ is restricted such that E is a process constant only and E' is an arbitrary parallel expression.

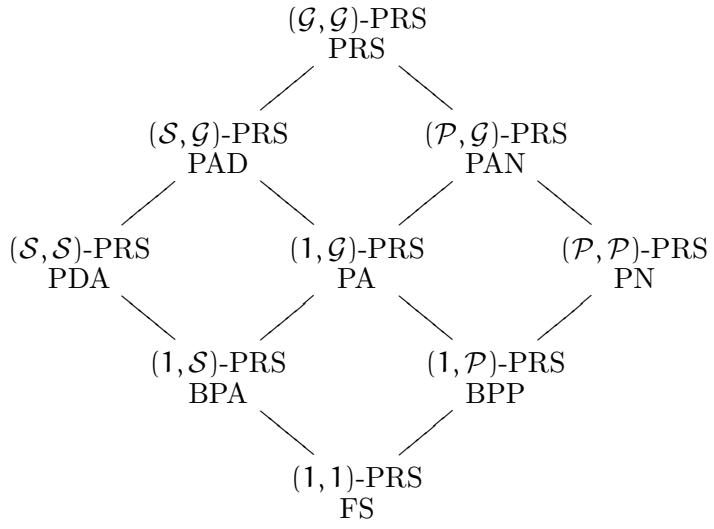


Figure 1.2: The PRS-Hierarchy

The complete PRS-hierarchy is depicted in Figure 1.2. This hierarchy is known to be strict with respect to strong bisimilarity and none of the classes in the hierarchy is Turing powerful since, e.g., the reachability problem is decidable even for the whole PRS class [42].

The reader may wonder what is the connection with the models of infinite-state systems introduced in previous sections? The answer is surprisingly simple. Most of the classes in Figure 1.2 correspond naturally to the well-known classes of processes like context-free processes, pushdown automata and Petri nets. Rules of the type $\mathsf{X} \stackrel{a}{\longrightarrow} \mathsf{Y}.\mathsf{Z}$ define the BPA class, rules of the type $\mathsf{X} \stackrel{a}{\longrightarrow} \mathsf{Y}|\mathsf{Z}$ correspond to BPP processes, rules like $\mathsf{X}.\mathsf{Y} \stackrel{a}{\longrightarrow} \mathsf{Z}.\mathsf{U}$ characterize the PDA class (the correspondence is not straightforward and was proved in [13] by Caucal), rules of the type $\mathsf{X}|\mathsf{Y} \stackrel{a}{\longrightarrow} \mathsf{Z}|\mathsf{U}$ are Petri net rules, and $\mathsf{X} \stackrel{a}{\longrightarrow} (\mathsf{Y}|\mathsf{Z}).\mathsf{U}$ is a characteristic rule for the PA-processes of Baeten and Weijland [4].

Even the class of state-extended BPP has its place in the hierarchy. It lies between basic parallel processes (BPP) and Petri nets (PN) and its position is strict in both directions.

The study of decidability and complexity of bisimilarity checking problems for the classes from the PRS-hierarchy represents an active field of research; a summary of recent results is provided in [61].

1.7 Parallel Complexity

An intriguing question to ask about bisimulation is does it have an efficient *parallel* solution? The class NC contains those problems that can be solved in polylogarithmic time using a polynomial number of processors (in the size of the input). NC is generally regarded as the class of problems that have fast parallel solutions.

It is believed that P-complete problems cannot be in NC. A problem is in

	strong/weak trace equivalence	*strong* bisimilarity	*weak* bisimilarity
FS	PSPACE-complete	P-complete	P-complete
BPA	Π_1^0-complete	$\in$2-EXPTIME	???
PDA	Π_1^0-complete	decidable	Σ_1^1-complete
BPP	Π_1^0-complete	PSPACE-complete	???
MSA	Π_1^0-complete	Π_1^0-complete	Σ_1^1-complete
PN	Π_1^0-complete	Π_1^0-complete	Σ_1^1-complete

Figure 1.3: Summary of Complexity Results

P if it can be solved by a deterministic Turing machine in polynomial time. A problem is P-complete if it is in the class P, and it is P-hard in the sense that any other problem in P is log-space reducible to it. A reduction is log-space if it uses at most a logarithmic amount of intermediate storage space.

Balcazar et al. [5] established the P-completeness of bisimulation checking on regular processes via a log-space reduction from the *Monotone Alternating Circuit Value Problem*. Despite this negative result, several parallel and distributed algorithms for deciding bisimulation equivalence of regular processes have been proposed that achieve non-trivial speedups in practice [8, 37, 56, 80].

1.8 Conclusions

We have offered a brief history of the computational complexity of bisimulation. Several comprehensive surveys about the subject, focusing on *infinite-state processes*, have been written (e.g., [35, 36, 50]), including a handbook chapter [11]. There is even now a project devoted to maintaining an up-to-date overview of the state of the art in this dynamic research topic [61].

The reader may have noticed the following trend about bisimulation equivalence: it is computationally easier to decide than language equivalence, regardless of the nature of the underlying process model, be it finite-state or infinite-state. It is interesting to search for an explanation to this computational dichotomy. Some insight can be gained by again noting that bisimulation is a much more discriminating equivalence than language equivalence, to the point where it is easier to decide. In particular, bisimilar states, for any symbol $\mathtt{a}$, must lead to bisimilar states. The absence of this restriction on language-equivalent states in some sense forces one to determinize the automata in question to decide equivalence, a costly proposition indeed.

On the other hand weak bisimilarity is much harder to decide on infinite-state processes compared not only to (strong) bisimilarity but also to other (even weak) equivalences from van Glabbeek's spectrum such as the trace equivalences. Whenever a process formalism allows for a finite-state control unit, weak bisimilarity becomes highly undecidable (Σ_1^1-complete in the analytical hierarchy) [65] whereas, e.g., strong and weak trace equivalence remain on the first level of the arithmetical hierarchy (see [32]). A summary of these results

is provided in Figure 1.3.

Finally, we ask what are the *practical ramifications* of the computational dichotomy? Happily, the answer is a positive one for computer scientists interested in bisimilarity, such as concurrency theorists and verification tool builders. In this case, one is confronted (at least for strong bisimilarity) with a tractable problem even for processes of a fairly expressive nature.

Bibliography

[1] S. Abramsky. The lazy lambda calculus. In D. A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Welsey, Reading, MA, 1990.

[2] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM*, 40(3):653–682, July 1993.

[3] J.C.M. Baeten and R.J. van Glabbeek. Another look at abstraction in process algebra. In *Proceedings of the 14th International Colloquium on Automata, Languages and Programming (ICALP'87)*, volume 267 of *Lecture Notes in Computer Science*, pages 84–94. Springer-Verlag, 1987.

[4] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[5] J. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4:638–648, 1992.

[6] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft, und Kommunikationsforschung*, 14:143–177, 1961.

[7] J. Barwise and L. Moss. *Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena*. CSLI Lecture Notes, No. 60, Stanford, CA, 1996.

[8] S. Blom and S. Orzan. A distributed algorithm for strong bisimulation reduction of state spaces. In L. Brim and O. Grumberg, editors, *Parallel and Distributed Model Checking (PDMC 2002)*. Elsevier Science, Electronic Notes in Theoretical Computer Science 68 No. 4, 2002.

[9] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[10] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[11] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.

[12] O. Burkart, D. Caucal, and B. Steffen. An elementary decision procedure for arbitrary context-free processes. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, pages 423–433. Volume 969 of *Lecture Notes in Computer Science*, Springer-Verlag, 1995.

[13] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, 1992.

[14] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation is decidable for basic parallel processes. In *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, 1993.

[15] S. Christensen, Y. Hirshfeld, and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pages 386–396. IEEE, 1993.

[16] S. Christensen, H. Hüttel, and C. Stirling. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Information and Computation*, 12(2):143–148, 1995.

[17] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag, 1981.

[18] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[19] J. Esparza and J.Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proceedings of the 2nd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'99)*, volume 1578 of *Lecture Notes in Computer Science*, pages 14–30. Springer-Verlag, 1999.

[20] P. Godefroid. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032 of *LNCS*. Springer-Verlag, 1996.

[21] J.F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):353–371, 1994.

[22] J. Hartmanis. Turing Award lecture: On computational complexity and the nature of computer science. *Communications of the ACM*, 37(10):37–43, October 1994.

[23] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the Association for Computing Machinery*, 32(1):137–161, 1985.

[24] Y. Hirshfeld. Petri nets and the equivalence problem. In *Proceedings of the 7th Workshop on Computer Science Logic (CSL'93)*, volume 832 of *Lecture Notes in Computer Science*, pages 165–174. Springer-Verlag, 1994.

[25] Y. Hirshfeld. Bisimulation trees and the decidability of weak bisimulations. In *Proceedings of the 1st International Workshop on Verification of Infinite State Systems (INFINITY'96)*, volume 5 of *Electronic Notes in Theoretical Computer Science*. Springer-Verlag, 1996.

[26] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158:143–159, May 1996.

[27] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimulation equivalence of normed basic parallel processes. *Mathematical Structures in Computer Science*, 6:251–259, 1996.

[28] C.A.R. Hoare. Communicating sequential processes. In *On the construction of programs – an advanced course*, pages 229–254. Cambridge University Press, 1980.

[29] H. B. Hunt, D. J. Rosenkrantz, and T. G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *Journal of Computer and System Sciences*, 12:222–268, 1976.

[30] H. Hüttel. Undecidable equivalences for basic parallel processes. In *Proceedings of the 2nd International Symposium on Theoretical Aspects of Computer Software (TACS'94)*, volume 789 of *Lecture Notes in Computer Science*, pages 454–464. Springer-Verlag, 1994.

[31] D.T. Huynh and L. Tian. On deciding readiness and failure equivalences for processes in Σ_2^{P}. *Information and Computation*, 117(2):193–205, 1995.

[32] P. Jančar. High undecidability of weak bisimilarity for Petri nets. In *Proceedings of Colloquium on Trees in Algebra and Programming (CAAP'95)*, volume 915 of *Lecture Notes in Computer Science*, pages 349–363. Springer-Verlag, 1995.

[33] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995.

[34] P. Jančar. Strong bisimilarity on basic parallel processes is PSPACE-complete. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 218–227. IEEE Computer Society Press, 2003.

[35] P. Jančar and A. Kučera. Equivalence-checking with infinite-state systems: Techniques and results. In *Proceedings of the 29th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'02)*, volume 2540 of *Lecture Notes in Computer Science*, pages 41–73. Springer-Verlag, 2002.

[36] P. Jančar and F. Moller. Techniques for decidability and undecidability of bisimilarity – an invited tutorial. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 30–45. Springer-Verlag, 1999.

[37] C. Jeong, Y. Kim, H. Kim, and Y. Oh. A faster parallel implementation of Kanellakis-Smolka algorithm for bisimilarity checking. In *Proceedings of the International Computer Symposium*, Tainan, Taiwan, 1998.

[38] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

[39] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, May 1990.

[40] J. Knoop. *Optimal Interprocedural Program Optimization: A New Framework and its Application*, volume 1428 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[41] A.J. Korenjak and J.E. Hopcroft. Simple deterministic languages. In *Proceedings of the 7th Annual IEEE Symposium on Switching and Automata Theory*, pages 36–46. IEEE, 1966.

[42] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.

[43] R. Mayr. Weak bisimilarity and regularity of BPA is EXPTIME-hard. In *Proceedings of the 10th International Workshop on Expressiveness in Concurrency (EXPRESS'03)*, pages 160–143, 2003. To appear in ENTCS.

[44] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[45] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[46] R. Milner. A modal characterization of observable machine-behaviour. In *Proceedings of the 6th Colloquium on Trees in Algebra and Programming (CAAP'81)*, volume 112 of *Lecture Notes in Computer Science*, pages 25–34. Springer-Verlag, 1981.

[47] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[48] R. Milner. Elements of interaction — Turing Award lecture. *Communications of the ACM*, 36(1):78–89, January 1993.

[49] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Parts I and II. *Information and Computation*, 100, 1992.

[50] F. Moller. Infinite results. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer-Verlag, 1996.

[51] F. Moller and S. A. Smolka. On the computational complexity of bisimulation. *ACM Computing Surveys*, 27(2):287–289, June 1995.

[52] F. Moller and S. A. Smolka. On the computational complexity of bisimulation, redux. In *PCK50: Principles of Computing and Knowledge: Paris C. Kanellakis Memorial Workshop*, San Diego, CA, June 2003. ACM.

[53] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, December 1987.

[54] D. M. R. Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th G.I. Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[55] D. Peled. All from one, one for all: On model checking using representatives. In *Proceedings of the 5th International Computer Aided Verification Conference (CAV'93)*, volume 697 of *LNCS*, pages 409–423, 1993.

[56] S. Rajasekaran and I. Lee. Parallel algorithms for relational coarsest partition problems. *IEEE Transactions on Parallel and Distributed Systems*, 9(7), July 1998.

[57] J.M. Rutten. Universal coalgebra: A theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[58] K. Segerberg. An essay in classical modal logic. *Filosofiska Studier*, 13:301–322, 1971.

[59] G. Senizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 120–129. IEEE, 1998.

[60] G. Senizergues. L(A)=L(B)? Decidability results from complete formal systems. *Theoretical Computer Science*, 251(1-2):1–166, January 2001.

[61] J. Srba. Roadmap of infinite results. *Bulletin of the European Association for Theoretical Computer Science (Columns: Concurrency)*, 78:163–175, October 2002. Updated online version: http://www.brics.dk/~srba/roadmap.

[62] J. Srba. Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard. In *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 535–546. Springer-Verlag, 2002.

[63] J. Srba. Strong bisimilarity and regularity of Basic Process Algebra is PSPACE-hard. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, pages 716–727. Volume 2380 of *Lecture Notes in Computer Science*, Springer-Verlag, 2002.

[64] J. Srba. Undecidability of weak bisimilarity for pushdown processes. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *Lecture Notes in Computer Science*, pages 579–593. Springer-Verlag, 2002.

[65] J. Srba. Completeness results for undecidable bisimilarity problems. In *Proceedings of the 5th International Workshop on Verification of Infinite-State Systems (INFINITY'03)*, pages 9–22, 2003. To appear in ENTCS.

[66] C. Stirling. Games for bisimulation and model checking. In *Notes for Mathfit Workshop on Finite Model Theory*, University of Wales, Swansea, July 1996.

[67] C. Stirling. Decidability of bisimulation equivalence for pushdown processes. Research Report EDI-INF-RR-0005, School of Informatics, Edinburgh University, January 2000.

[68] C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, 255(1-2):1–31, March 2001.

[69] C. Stirling. Decidability of weak bisimilarity for a subset of basic parallel processes. In *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'01)*, volume 2030 of *Lecture Notes in Computer Science*, pages 379–393. Springer-Verlag, 2001.

[70] C. Stirling. Deciding DPDA equivalence is primitive recursive. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *Lecture Notes in Computer Science*, pages 821–832. Springer-Verlag, 2002.

[71] W. Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science (extended abstract). In *Proceedings of the 4th International Joint Conference CAAP/FASE, Theory and Practice of Software Development (TAPSOFT'93)*, volume 668 of *Lecture Notes in Computer Science*, pages 559–568. Springer-Verlag, 1993.

[72] D. Turi and J. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Mathematical Structures in Computer Science*, 8(5):481–540, 1998.

[73] A. Valmari. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (ICATPN'90): Advances in Petri Nets*, volume 483 of *LNCS*, pages 491–515. Springer-Verlag, 1991.

[74] J. van Bentham. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, chapter II.4, pages 167–247. Kluwer Academic Publishers, 1984.

[75] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, CWI/Vrije Universiteit, 1990.

[76] R.J. van Glabbeek. The linear time—branching time spectrum. In *Proceedings of the 1st International Conference on Theories of Concurrency: Unification and Extension (CONCUR'90)*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, 1990.

[77] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Information Processing Letters*, 89:613–618, 1989.

[78] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[79] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Annual IEEE Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society Press, 1986.

[80] S. Zhang and S. A. Smolka. Efficient parallelization of equivalence checking algorithms. In M. Diaz and R. Groz, editors, *Proceedings of the 5th International Conference on Formal Description Techniques*, pages 133–146, October 1992.

Chapter 2

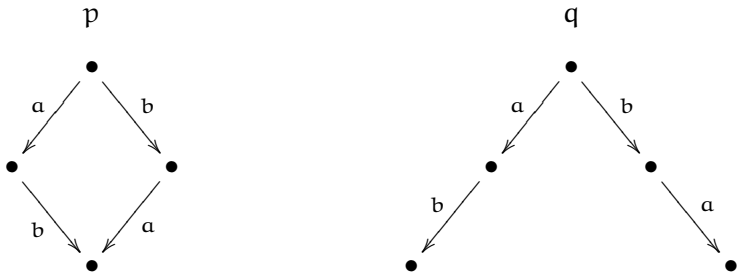
Focus Areas

We shall now have a closer look at the particular topics discussed in this thesis. We start with an intuitive description of partial order semantics and the corresponding notions of bisimilarity and continue with a short introduction to cryptographic protocols.

2.1 Partial Order Semantics of Concurrent Processes

In this section we will compare different approaches when dealing with concurrent executions of actions (events) and motivate the reader for the partial order approach adopted in Chapter 4.

Let us assume that we want to model a process $\mathtt{P}$ which can perform concurrently (independently) actions $\mathtt{a}$ and $\mathtt{b}$, and terminate afterwards. In the *interleaving approach*, concurrency is modelled using non-determinism. This means that the process $\mathtt{P}$ can be modelled by the following process $\mathtt{p}$ (we also show the corresponding unfolding $\mathtt{q}$ of the process $\mathtt{p}$).



Hence the process $\mathtt{p}$ can perform either the action $\mathtt{a}$ followed by the action $\mathtt{b}$ or vice versa. In the interleaving approach there is no apparent way to distinguish between a concurrent execution of the two actions and a nondeterministic choice between them (as shown in the process $\mathtt{p}$). This is supported also by the fact that in the unfolding $\mathtt{q}$ of the process $\mathtt{p}$, we do not reach the same state after performing the sequences $\mathtt{ab}$ and $\mathtt{ba}$, i.e., $\mathtt{q}$ is a tree.

In the *partial order approach*, we specify the distinction between concurrency and nondeterminism explicitly. Mazurkiewicz's traces were suggested as a suitable semantics for this purpose [10,11]. Modelling of concurrency is achieved by introducing an independence alphabet with an explicit independence relation.

Example 2.1 *In this example we shall explain the main idea of independence alphabet. Assume that the set of actions is equal to* $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ *such that the actions* $\mathtt{a}$ *and* $\mathtt{b}$ *are mutually independent, but dependent with the action* $\mathtt{c}$.

This can be formally defined by introducing an irreflexive and symmetric independence relation I *over the set of actions. In our example we have*

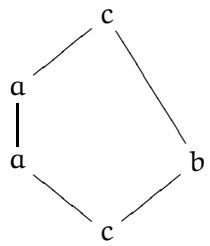
$$\mathrm{I} \stackrel{\text{def}}{=} \{(\mathtt{a}, \mathtt{b}), (\mathtt{b}, \mathtt{a})\}.$$

The intuition is that whenever during a process execution the actions $\mathtt{a}$ *and* $\mathtt{b}$ *appear after one another, the observer of such a system should not be able to determine in which order they were performed. Hence e.g. the execution of a sequence* $\mathtt{caabc}$ *should appear the same as e.g.* $\mathtt{cabac}$.

This can be formalized by saying that two action sequences are one-step related whenever one sequence can be obtained from the other by swapping two neighbouring independent actions. By taking a reflexive and transitive closure of this one-step relation we get an equivalence relation which identifies exactly those action sequences that cannot be distinguished.

In our example $\mathtt{caabc}$ *and* $\mathtt{cabac}$ *are one-step related because in* $\mathtt{caabc}$ *we have two neighbouring independent actions at positions* 3 *and* 4, *and by swapping them we get the sequence* $\mathtt{cabac}$. *Since also* $\mathtt{cabac}$ *is one-step related to* $\mathtt{cbaac}$, *we can conclude that* $\mathtt{caabc}$ *is (in two steps) equivalent to* $\mathtt{cbaac}$. *In fact the equivalence class represented by* $\mathtt{caabc}$ *contains exactly the following sequences of actions:* $\mathtt{caabc}$, $\mathtt{cabac}$ *and* $\mathtt{cbaac}$.

Another possible notation is to describe such an equivalence class as a labelled partially ordered set (trace). The labelled partial order corresponding to the sequence $\mathtt{caabc}$ *is the following one.*



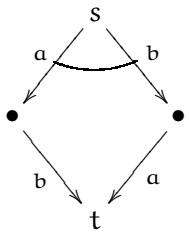
By taking all linearisations of this partial order we obtain exactly the equivalence class of the sequences represented by $\mathtt{caabc}$, *i.e.,* $\{\mathtt{caabc}, \mathtt{cabac}, \mathtt{cbaac}\}$.

Many models were suggested to fulfill these ideas, e.g. elementary net systems, event structures, trace languages and asynchronous transition systems. For surveys see [13, 18]. We will focus on the model of labelled asynchronous transition systems since it is a natural generalization of labelled transition systems frequently used in the interleaving approach.

Asynchronous transition systems were introduced by Bednarczyk [2] and by Shields [15]. The idea is that labelled transition systems are extended with an irreflexive and symmetric independence relation over the actions. Whenever two actions $\mathtt{a}$ and $\mathtt{b}$ are independent and can be performed in a sequence (i.e. from a state $\mathtt{s}$ after performing the sequence $\mathtt{ab}$ a state $\mathtt{t}$ is reached),

the asynchronous transition system must satisfy that from s it is possible to perform also the sequence ba and reach the same state t.

When modelling the process P that can perform independently the actions a and b, we get the following asynchronous transition system (to indicate explicitly the independence between a and b we connect them by an arc).



In order to define a suitable notion of unfolding of the process s above, we still require that the unfolding is an acyclic labelled transition system but we do not insist that it has to be a tree. In particular, whenever two independent actions can be performed in a sequence, the requirement on the asynchronous transition system mentioned above is reflected also in the unfolding. Hence e.g. in our picture the unfolding of the asynchronous process s is isomorphic to the process s itself. On the other hand, nondeterministic choice between dependent actions does not obey the "diamond" property and unfolds in the usual way into a tree. This also means that any labelled transition system can be considered as an asynchronous transition system with the empty independence relation.

Remark 2.1 *Let us mention an interesting observation which illustrates how interleaving and partial order approaches intuitively explain the different understanding of concurrency. First, we consider the process* p *from the beginning of this subsection and its behaviour (unfolding)* q*. As mentioned before, after the actions* a *and* b *were performed, different states in* q *are reachable depending on the order in which these actions were performed. It can be checked by going back into the history and verifying the last action that was executed (either* b *or* a*). This action is always unique. On the other hand, when considering the process* s *(defined above) which unfolds into itself, after the actions* a *and* b *were performed the current situation of the system is represented by the state* t*. However, looking back into the history we cannot say whether* a *or* b *was performed as the last action since the situation is symmetric for these two actions. This reflects the intuition that whenever two actions are truly concurrent, one cannot determine the order in which they were executed.*

When defining a notion of behavioural equivalence for asynchronous transition systems, the independence relation should be taken into account. Such equivalences are usually defined on the corresponding unfoldings of given asynchronous processes. Considering e.g. bisimilarity, we can extend the rules of bisimulation games on unfoldings to allow backward moves of the attacker (Spoiler) and the defender (Duplicator). Hence during a bisimulation game the complete history of the game is remembered and it is possible to backtrack into the history. In case where the independence relation is empty, the history is

uniquely determined since the unfolding is a tree. If it is not the case, there might be different choices for going one step back into the history.

Assume again the process $\mathtt{s}$ from the previous picture. After performing the sequence $\mathtt{ab}$ the state $\mathtt{t}$ is reached. If the attacker now decides to make the backward move, he has two choices: either to the state reachable from $\mathtt{s}$ after performing the action $\mathtt{a}$ (the left branch), or to the state reachable from $\mathtt{s}$ after performing the action $\mathtt{b}$ (the right branch).

This gives rise to the notion of *hereditary history preserving bisimilarity* (hhp-bisimilarity) that was introduced by Bednarczyk [3] and discovered independently by Joyal, Nielsen and Winskel [7] using a categorical approach to bisimilarity.

Remark 2.2 *Another equivalence called* history preserving bisimilarity *was introduced by many authors, among others by Rabinovich and Trakhtenbrot [14], and van Glabbeek and Goltz [16]. This equivalence is similar to hhp-bisimilarity but allows only a limited access into the history.*

The negative news is that these versions of bisimilarity based on partial orders are usually intractable for automatic verification. For finite-state asynchronous processes, history preserving bisimilarity remains decidable (proved by Vogler [17] for a related model of 1-safe Petri nets) but it becomes an EXPTIME-complete problem (a result by Jategaonkar and Meyer [6]). Even more, hhp-bisimilarity was shown to be undecidable for labelled finite-state asynchronous processes and 1-safe Petri nets by Jurdzinski and Nielsen [8]. In Chapter 4 we shall further extend the result and demonstrate that the problem is undecidable also for unlabelled asynchronous transition systems.

2.2 Formal Techniques for Cryptographic Protocols

The main purpose of a *cryptographic protocol* is to establish a secure and confidential communication among its participants, possibly through an open and hostile environment.

A cryptographic protocol consists of two main ingredients: a *cryptographic part* (a way how to encrypt messages using e.g. public or shared keys, together with cryptoanalysis which estimates the computational strengths of such encryptions) and a description of a *communication scheme* for exchanging the encrypted messages. In the study of formal techniques, we usually abstract from the implementation details of cryptographic primitives (i.e. we do not discuss specific algorithms of the encryption and decryption process) and we rather accept the so called *perfect encryption hypothesis*. This means that an attacker (intruder) of the protocol is assumed to be able to spy, record, modify and reply to the exchanged messages but is not able to decrypt messages without knowing the decryption key or to create encrypted messages without knowing the respective plaintext and encryption key.

On a semi-formal level, the communication schemes of cryptographic protocols are usually described by a few primitive building blocks like $\{M\}_K$ which

stands for the encryption of a message M by a key K, (M_1, M_2) which represents the pairing function and a few other operators like hashing. Probably the most famous (and studied) example is the authentication protocol by Needham and Shroeder [12] (see also [9]):

1. $A \rightarrow B$: $\{(A, N_A)\}_{K_B}$
2. $B \rightarrow A$: $\{(N_A, N_B)\}_{K_A}$
3. $A \rightarrow B$: $\{N_B\}_{K_B}$

The three steps of the protocol are interpreted as follows. First, the participant A initiates the protocol by sending to B the pair (A, N_A) consisting of the identification of A and a fresh nonce N_A, all encrypted by the public key of K_B. In the second step, B decrypts the message using his/her private key and returns the nonce N_A together with a fresh nonce N_B, both encrypted by a public key of A. In the last phase A decrypts the message using his/her private key, verifies that the same nonce N_A that had been sent was also returned and confirms this phase by forwarding the encryption of the nonce N_B to B. A more formal description of the protocol can be achieved e.g. by using a process calculi, like the spi-calculus [1].

As we (under the perfect encryption hypothesis) assume that a potential intruder listening to (or actively interfering with) the communication cannot decrypt any of these messages exchanged between A and B, it might seem impossible to gain any knowledge of the plaintext parts of the messages or to make the participants to misinterpret the parties they are communicating with. Nevertheless, even under the perfect encryption hypothesis, several attacks on the protocol are possible, as discovered e.g. by Lowe in [9].

One possible attack is the man-in-the-middle attack. Assume that an intruder I with a public key K_I initiates the protocol with A and receives $\{A, N_A\}_{K_I}$. The intruder can decrypt the message by his private key, encrypt it by the public key of B and forward $\{A, N_A\}_{K_B}$ to the participant B. As B believes that A initiated the protocol, he/she sends the answer $\{N_A, N_B\}_{K_A}$ to A. The participant A initiated the protocol with I and hence assumes the message arrived from I and hence forwards $\{N_B\}_{K_I}$ to the intruder. The intruder decrypts this message by his private key and forwards $\{N_B\}_{K_B}$ to B. At this point B believes that he/she is communicating with A but in fact the communication goes via I and moreover the intruder also knows the secret nonce N_B.

The example of Needham-Schroeder protocol shows that problems related to such simple protocols are difficult to analyze and that even under the assumption of perfect cryptography, protocols can still contain errors resulting from a poor design of the communication scheme.

In order to automatically reason about cryptographic protocols, a formal model of the protocol has to be defined. An extension of the pi-calculus with basic cryptographic primitives called *spi-calculus* has been suggested by Abadi and Gordon in [1] and is widely used to describe a variety of cryptographic protocols. Unfortunately, this formalism is easily seen to be Turing powerful and

hence the question of finding suitable subcalculi with certain decidable properties is of high interest. A detailed overview of positive and negative results for several subcalculi are described in the introduction parts of Chapters 5 and 6, with a particular focus on perhaps the simplest class of protocols called *ping-pong protocols*. These are memory-less protocols and the exchanged messages consist solely of encrypted plaintexts with no other composition operators. The secrecy of finite ping-pong protocols can be decided in polynomial time, a result by Dolev and Yao [5]. Later, Dolev, Even and Karp found a cubic-time algorithm [4] by expressing secrecy as emptiness of the intersection of a context-free language with a regular language.

In Chapters 5 and 6 we study a selection of verification problems related to recursive (and replicative) extensions of the ping-pong protocols. In particular, the impossibility results presented in these chapters are of general interest as any (recursive) formalism for cryptographic protocols should be able to describe at least the ping-pong behaviour.

Bibliography

[1] M. Abadi and A.D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.

[2] Marek A. Bednarczyk. *Categories of Asynchronous Systems*. PhD thesis, University of Sussex, 1988.

[3] Marek A. Bednarczyk. Hereditary history preserving bisimulations or what is the power of the future perfect in program logics. Technical report — http://www.ipipan.gda.pl/~marek, Polish Academy of Sc., Gdansk, 1991.

[4] D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols. *Information and Control*, 55(1–3):57–68, 1982.

[5] D. Dolev and A.C. Yao. On the security of public key protocols. *Transactions on Information Theory*, IT-29(2):198–208, 1983.

[6] Lalita Jategaonkar and Albert R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154(1):107–143, 1996.

[7] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

[8] M. Jurdzinski and M. Nielsen. Hereditary history preserving bisimilarity is undecidable. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS'00)*, volume 1770 of *LNCS*. Springer-Verlag, 2000.

[9] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of the 2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 147–166. Springer-Verlag, 1996.

[10] A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

[11] Antoni Mazurkiewicz. Trace theory. In *Petri Nets, Applications and Relationship to other Models of Concurrency*, number 255 in LNCS, pages 279–324. Springer-Verlag, 1987.

[12] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[13] M. Nielsen and G. Winskel. Petri nets and bisimulation. *Theoretical Computer Science*, 153(1–2):211–244, 1996.

[14] A. Rabinovich and B. Trakhtenbrot. Behaviour structures and nets of processes. *Fundamenta Informaticae*, 11:357–404, 1988.

[15] M.W. Shields. Concurrent machines. *Computer Journal*, 28:449–465, 1985.

[16] R.J. van Glabbeek and Ursula Goltz. Equivalence notions for concurrent systems and refinement of actions (extended abstract). In *Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science (MFCS'89)*, volume 379 of *LNCS*, pages 237–248. Springer-Verlag, 1989.

[17] Walter Vogler. Deciding history preserving bisimilarity. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming (ICALP'91)*, volume 510 of *LNCS*, pages 493–505. Springer-Verlag, 1991.

[18] Winskel and M. Nielsen. Models for concurrency. In *Handbook of Logic in Computer Science*, volume 4, *Semantic Modelling*, pages 1–148. Oxford University Press, 1995.

Chapter 3

Bibliographical Remarks and Author's Contribution

Chapter 1 of the thesis is based on a slightly modified version of the following paper.

> On the Computational Complexity of Bisimulation, Redux by F. Moller, S. Smolka and J. Srba. Information and Computation, pages 129-143, volume 192(2), Springer-Verlag, 2004.

Section 2.1 on informal introduction to partial order semantics is identical (apart from a minor typographical changes) to the respective section from the following thesis.

> Decidability and Complexity Issues for Infinite-State Processes by J. Srba. BRICS, Department of Computer Science, University of Aarhus, Denmark. PhD Thesis. 171 pages, 2003.

The original author's contribution presented in this thesis is summarized in the following list:

> On the Computational Complexity of Bisimulation, Redux by F. Moller, S. Smolka and J. Srba. Information and Computation, pages 129-143, volume 192(2), Springer-Verlag, 2004.
>
> Undecidability of Domino Games and Hhp-Bisimilarity by M. Jurdzinski, M. Nielsen and J. Srba. Information and Computation, pages 343-368, volume 184(2), Springer-Verlag, 2003.
>
> Recursive Ping-Pong Protocols by H. Hüttel and J. Srba. In Proceedings of 4th International Workshop on Issues in the Theory of Security (WITS'04), pages 129-140, 2004.
>
> Recursion vs. Replication in Simple Cryptographic Protocols by H. Hüttel and J. Srba. In Proceedings of 31st Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'05), pages 175–184, volume 3381 of LNCS, Springer-Verlag, 2005.

Detailed comments concerning the content of the following chapters is provided in the beginning of each chapter.

Part II

Papers

Chapter 4

Undecidability of Domino Games and Hhp-Bisimilarity

The paper *Undecidability of Domino Games and Hhp-Bisimilarity* presented in this chapter has been published as a journal paper.

> Undecidability of Domino Games and Hhp-Bisimilarity by M. Jurdzinski, M. Nielsen and J. Srba. Information and Computation, pages 343-368, volume 184(2), Springer-Verlag, 2003.

Except for minor typographical changes the content of this chapter is equal to the journal paper. The main contribution of the present author is the definition of *unlabelled* domino bisimilarity game and the proof of its undecidability. This result is used to demonstrate undecidability of hereditary history preserving bisimilarity on unlabelled asynchronous transition systems, which is also a part to which the author contributed.

Undecidability of Domino Games and Hhp-Bisimilarity

Marcin Jurdziński, Mogens Nielsen and Jiří Srba

Abstract. History preserving bisimilarity (hp-bisimilarity) and hereditary history preserving bisimilarity (hhp-bisimilarity) are behavioural equivalences taking into account causal relationships between events of concurrent systems. Their prominent feature is that they are preserved under action refinement, an operation important for the top-down design of concurrent systems. It is shown that, in contrast to hp-bisimilarity, checking hhp-bisimilarity for finite labelled asynchronous transition systems is undecidable, by a reduction from the halting problem of 2-counter machines. To make the proof more transparent a novel intermediate problem of checking domino bisimilarity for origin constrained tiling systems is introduced and shown undecidable. It is also shown that the unlabelled domino bisimilarity problem is undecidable, which implies undecidability of hhp-bisimilarity for unlabelled finite asynchronous systems. Moreover, it is argued that the undecidability of hhp-bisimilarity holds for finite elementary net systems and 1-safe Petri nets.

4.1 Introduction

The notion of behavioural equivalence which has attracted most attention in concurrency theory is bisimilarity, originally introduced by Park [25] and Milner [19]; concurrent programs are considered to have the same meaning if they are bisimilar. The prominent role of bisimilarity is due to many pleasant properties it enjoys; we mention a few of them here.

A process of checking whether two transition systems are bisimilar can be seen as a two player game which is in fact an Ehrenfeucht-Fraïssé type of game for modal logic. More precisely, there is a winning strategy for a player who wants to show that the systems are bisimilar if and only if the systems cannot be distinguished by the formulas of the logic; the result due to Hennessy and Milner [12].

Another notable property of bisimilarity is its computational feasibility; see for example the overview note [21]. Let us illustrate this on the examples of finite transition systems and a class of infinite-state transition systems generated by context free grammars. For finite transition systems there are very efficient polynomial time algorithms for checking bisimilarity [17, 24], in sharp contrast to PSPACE-completeness of the classical language equivalence. For transition systems generated by context free grammars, while language equivalence is undecidable, bisimilarity is decidable [4], and if the grammar has no redundant nonterminals, even in polynomial time [13]. Furthermore, as the results of Groote and Hüttel [11] indicate, bisimilarity has a very rare status of being a decidable equivalence for context free grammars: all the other equivalences

in the linear-branching time hierarchy [8] are undecidable. The algorithmic tractability makes bisimilarity especially attractive for automatic verification of concurrent systems.

The essence of bisimilarity, quoting Hennessy and Milner [12], "is that the behaviour of a program is determined by how it communicates with an observer." Therefore, the notion of what can be observed of a behaviour of a system affects the notion of bisimilarity. An abstract definition of bisimilarity for arbitrary categories of models due to Joyal et al. [15] formalizes this idea. Given a category of models where objects are behaviours and morphisms correspond to extension of behaviours, and given a subcategory of observable behaviours, the abstract definition yields a notion of bisimilarity for all behaviours with respect to observable behaviours. For example, for rooted labelled transition systems, taking synchronization trees [19] into which they unfold as their behaviours, and sequences of actions as the observable behaviours, we recover the standard strong bisimilarity of Park and Milner [15].

In order to model concurrency more faithfully several models have been introduced (see [30] for a survey) that make explicit the distinction between events that can occur concurrently, and those that are causally related. Then a natural choice is to replace sequences, i.e., linear orders as the observable behaviours, by partial orders of occurrences of events with causality as the ordering relation. For example, taking unfoldings of labelled asynchronous transition systems into event structures as the behaviours, and labelled partial orders as the observations, Joyal et al. [15] obtained from their abstract definition the hereditary history preserving bisimilarity (hhp-bisimilarity), independently introduced and studied by Bednarczyk [1].

A similar notion of bisimilarity has been studied before, namely history preserving bisimilarity (hp-bisimilarity), introduced by Rabinovich and Trakhtenbrot [26] and van Glabbeek and Goltz [9]. For the relationship between hp- and hhp-bisimilarity see for example [1, 7, 15].

One of the important motivations to study partial order based equivalences was the discovery that hp-bisimilarity has a rare status of being preserved under action refinement [9], an operation important for the top-down design of concurrent systems. Bednarczyk [1] has extended this result to hhp-bisimilarity.

There is a natural logical characterization of hhp-bisimilarity checking games as shown by Nielsen and Clausen [22]: they are characteristic games for an extension of modal logic with backwards modalities, interpreted over event structures.

Hp-bisimilarity has been shown to be decidable for 1-safe Petri nets by Vogler [29], and to be DEXPTIME-complete by Jategaonkar, and Meyer [14]; let us just mention here that 1-safe Petri nets can be regarded as a proper subclass of finite asynchronous transition systems (see [30] for details), and that decidability of hp-bisimilarity can be easily extended to all finite asynchronous transition systems using the methods of [14].

Hhp-bisimilarity appears to be only a slight strengthening of hp-bisimilarity [15] and hence many attempts have been made to extend the above mentioned algorithms to the case of hhp-bisimilarity. However, decidability of hhp-bisimilarity has remained open, despite several attempts over the years [3, 7, 22,

23]. Fröschle and Hildebrandt [7] have discovered an infinite hierarchy of bisimilarity notions refining hp-bisimilarity, and coarser than hhp-bisimilarity, such that hhp-bisimilarity is the intersection of all the bisimilarities in the hierarchy. They have shown all these bisimilarities to be decidable for 1-safe Petri nets. Fröschle [6] has proved hhp-bisimilarity to be decidable for BPP-processes, a class of infinite state systems.

In this paper we resolve the question of decidability of hhp-bisimilarity for all finite state systems by showing it to be undecidable for finite, both labelled and unlabelled, asynchronous transition systems, finite elementary net systems, and 1-safe Petri nets. In order to make the proof more transparent we first introduce an intermediate problem of domino bisimilarity and show its undecidability by a direct reduction from the undecidable halting problem for 2-counter machines [20]. The undecidability of the novel problem of checking domino bisimilarity seems to be interesting in its own right and does not follow from somewhat related results for domino snakes [5] and domino games [10], nor from the undecidability of the classical tiling problems [2].

4.2 Hereditary history preserving bisimilarity

In this section we define hereditary history preserving bisimulation for asynchronous transition systems and we introduce the algorithmic problem of checking hereditary history preserving bisimilarity. We also mention the equivalent, and sometimes technically more convenient, problem of solving hereditary history preserving bisimilarity checking games.

Definition 4.1 *(Labelled/unlabelled asynchronous transition system).*
A labelled asynchronous transition system *is a tuple* $\mathsf{A} = (\mathsf{S}, \mathsf{s}^{\mathrm{ini}}, \mathsf{E}, \rightarrow, \mathsf{L}, \lambda, \mathrm{I})$*, where* S *is its set of* states*,* $\mathsf{s}^{\mathrm{ini}} \in \mathsf{S}$ *is the* initial state*,* E *is the set of* events*,* $\rightarrow \subseteq \mathsf{S} \times \mathsf{E} \times \mathsf{S}$ *is the set of* transitions*,* L *is the set of labels, and* $\lambda : \mathsf{E} \rightarrow \mathsf{L}$ *is the* labelling function*, and* $\mathrm{I} \subseteq \mathsf{E}^2$ *is the* independence relation *which is irreflexive and symmetric. We often write* $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}'$*, instead of* $(\mathsf{s}, \mathsf{e}, \mathsf{s}') \in \rightarrow$*. Moreover, the following conditions have to be satisfied:*

1. *if* $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}'$ *and* $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}''$ *then* $\mathsf{s}' = \mathsf{s}''$*,*
2. *if* $(\mathsf{e}, \mathsf{e}') \in \mathrm{I}$*,* $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}'$*, and* $\mathsf{s}' \xrightarrow{\mathsf{e}'} \mathsf{t}$*, then* $\mathsf{s} \xrightarrow{\mathsf{e}'} \mathsf{s}''$*, and* $\mathsf{s}'' \xrightarrow{\mathsf{e}} \mathsf{t}$ *for some* $\mathsf{s}'' \in \mathsf{S}$*.*

An asynchronous transition system is coherent *if it satisfies the following condition:*

3. *if* $(\mathsf{e}, \mathsf{e}') \in \mathrm{I}$*,* $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}'$*, and* $\mathsf{s} \xrightarrow{\mathsf{e}'} \mathsf{s}''$*, then* $\mathsf{s}' \xrightarrow{\mathsf{e}'} \mathsf{t}$*, and* $\mathsf{s}'' \xrightarrow{\mathsf{e}} \mathsf{t}$ *for some* $\mathsf{t} \in \mathsf{S}$*.*

An asynchronous transition system is prime *if it is acyclic and satisfies the following condition:*

4. *if* $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{t}$ *and* $\mathsf{s}' \xrightarrow{\mathsf{e}'} \mathsf{t}$ *then* $(\mathsf{e}, \mathsf{e}') \in \mathrm{I}$*.*

We say that an asynchronous transition system is unlabelled *if the set of labels* L *is a singleton set.*

Winskel and Nielsen [23, 30] give a thorough survey and establish formal relationships between asynchronous transition systems and other models for concurrency, such as Petri nets, and event structures. The independence relation is meant to model concurrency: independent events can occur concurrently, while those that are not independent are causally related or in conflict.

Let $A = (S, s^{\mathrm{ini}}, E, \rightarrow, L, \lambda, I)$ be a labelled asynchronous transition system. A sequence of events $\overline{e} = \langle e_1, e_2, \ldots, e_n \rangle \in E^*$ is a *run* of A if there are states $s_1, s_2, \ldots, s_{n+1} \in S$, such that $s_1 = s^{\mathrm{ini}}$, and for all $i \in \{1, 2, \ldots, n\}$, we have $s_i \xrightarrow{e_i} s_{i+1}$. We write $\mathtt{Runs}(A)$ to denote the set of runs of A. We extend the labelling function λ to runs in the standard way.

Let $\overline{e} = \langle e_1, e_2, \ldots, e_n \rangle \in \mathtt{Runs}(A)$. We say that the k-th event, $1 \leq k < n$, is *swappable* in $\overline{e}$ if $(e_k, e_{k+1}) \in I$. We define $\mathrm{Swap}(\overline{e})$ to be the set of numbers of swappable events in $\overline{e}$. We write $\overline{e} \otimes k$ to denote the result of *swapping* the k-th event of $\overline{e}$ with the $(k+1)$-st, i.e., the sequence $\langle e_1, \ldots, e_{k-1}, e_{k+1}, e_k, \ldots, e_n \rangle$. Note that if $k \in \mathrm{Swap}(\overline{e})$ then $\overline{e} \otimes k \in \mathtt{Runs}(A)$; it follows from condition 2. of the definition of an asynchronous transition system.

A run of a transition system models a finite *sequential* behaviour of a system: a sequence of occurrences of events. In order to model *concurrent* behaviours of a system we define an equivalence relation on the set of runs of an asynchronous transition system. We define the equivalence relation $\cong_A$ on $\mathtt{Runs}(A)$ to be the reflexive, symmetric, and transitive closure of

$$\big\{ \, (\overline{e}, \overline{e} \otimes k) \, : \, \overline{e} \in \mathtt{Runs}(A) \text{ and } k \in \mathrm{Swap}(\overline{e}) \, \big\}.$$

In other words, we have that $\overline{e_1} \cong_A \overline{e_2}$, for $\overline{e_1}, \overline{e_2} \in \mathtt{Runs}(A)$, if and only if $\overline{e_2}$ can be obtained from $\overline{e_1}$ by a finite number of swaps of swappable events.

We define an unfolding operation on asynchronous transition systems into prime asynchronous transition systems. The states of the unfolding of an asynchronous transition system A are meant to represent all concurrent behaviours of a system, just like the states of a synchronization tree represent all sequential behaviours of a system.

Definition 4.2 *(Unfolding).*
Let $A = (S, s^{\mathrm{ini}}, E, \rightarrow, L, \lambda, I)$ *be an asynchronous transition system. The unfolding* $\mathtt{Unf}(A)$ *of* A *is an asynchronous transition system with the same set of events, the labelling function, and the independence relation as* A*. The set of states, the initial state, and the transition relation of* $\mathtt{Unf}(A)$ *are defined as follows:*

- *the set of states* $S_{\mathtt{Unf}(A)}$ *of* $\mathtt{Unf}(A)$ *is defined to be* $\mathtt{Runs}(A)/_{\cong_A}$*, i.e., the set of concurrent behaviours of* A*,*

- *the initial state* $s^{\mathrm{ini}}_{\mathtt{Unf}(A)}$ *of* $\mathtt{Unf}(A)$ *is* $[\varepsilon]_{\cong_A}$*, i.e., the* $\cong_A$*-equivalence class of the empty run,*

- *the set of transitions* $\rightarrow_{\mathtt{Unf}(A)}$ *of* $\mathtt{Unf}(A)$ *consists of transitions of the form* $\big([\overline{e}]_{\cong_A}, e, [\overline{e} \cdot e]_{\cong_A}\big)$*, for all* $\overline{e} \in E^*$*, and* $e \in E$*, such that* $\overline{e} \cdot e \in \mathtt{Runs}(A)$*.*

The following proposition follows easily from the definition of $\mathtt{Unf}(A)$.

Proposition 4.1 *If* A *is an asynchronous transition system then its unfolding* $\mathtt{Unf}(A)$ *is a prime asynchronous transition system.*

Let $\overline{e} = \langle e_1, e_2, \ldots, e_n \rangle \in \mathtt{Runs}(A)$. We say that the k-th event, $1 \leq k \leq n$, is *most recent* in $\overline{e}$ if and only if $(e_k, e_\ell) \in I$, for all ℓ, such that $k < \ell \leq n$. We define $\mathrm{MR}(\overline{e})$ to be the set of numbers of most recent events in $\overline{e}$. We write $\overline{e} \ominus k$ to denote the result of removing the k-th event from $\overline{e}$, i.e., the sequence $\langle e_1, \ldots, e_{k-1}, e_{k+1}, \ldots, e_n \rangle$. Note that if $k \in \mathrm{MR}(\overline{e})$ then $\overline{e} \ominus k \in \mathtt{Runs}(A)$; it follows from condition 2. of the definition of an asynchronous transition system.

Definition 4.3 *(Hereditary history preserving bisimulation).*
Let $A_i = (S_i, s_i^{\mathrm{ini}}, E_i, \rightarrow_i, L, \lambda_i, I_i)$*, for* $i \in \{1, 2\}$*, be labelled asynchronous transition systems. A relation* $B \subseteq \mathtt{Runs}(A_1) \times \mathtt{Runs}(A_2)$ *is a* hereditary history preserving *(hhp-) bisimulation relating* A_1 *and* A_2 *if the following conditions are satisfied:*

1. $(\varepsilon, \varepsilon) \in B$*,*

and if $(\overline{e_1}, \overline{e_2}) \in B$ *then* $\lambda_1(\overline{e_1}) = \lambda_2(\overline{e_2})$*, and:*

2. *for all* $i \in \{1, 2\}$*, and* $e_i \in E_i$*, if* $\overline{e_i} \cdot e_i \in \mathtt{Runs}(A_i)$ *then there exists* $e_{3-i} \in E_{3-i}$*, such that* $\overline{e_{3-i}} \cdot e_{3-i} \in \mathtt{Runs}(A_{3-i})$*,* $\lambda_1(e_1) = \lambda_2(e_2)$*, and* $(\overline{e_1} \cdot e_1, \overline{e_2} \cdot e_2) \in B$*,*
3. $\mathrm{MR}(\overline{e_1}) = \mathrm{MR}(\overline{e_2})$*,*
4. *if* $k \in \mathrm{MR}(\overline{e_1}) = \mathrm{MR}(\overline{e_2})$ *then* $(\overline{e_1} \ominus k, \overline{e_2} \ominus k) \in B$*.*

Two asynchronous transition systems A_1, and A_2 are hereditary history preserving (hhp-) *bisimilar*, if there is an hhp-bisimulation relating them. For alternative and slightly varying definitions of hhp-bisimulation that all give rise to the same notion of hhp-bisimilarity see, e.g., the papers by Bednarczyk [1], Joyal et al. [15], Nielsen and Clausen [22], Nielsen and Winskel [23], or Fröschle and Hildebrandt [7].

The following proposition is straightforward since every asynchronous transition system A and its unfolding $\mathtt{Unf}(A)$ have the same set of runs and the same independence relation.

Proposition 4.2 *Asynchronous transition systems* A_1 *and* A_2 *are hhp-bisimilar if and only if their unfoldings* $\mathtt{Unf}(A_1)$ *and* $\mathtt{Unf}(A_2)$ *are hhp-bisimilar.*

The main results of this paper are the following theorems proved in Section 4.4.

Theorem 4.1 (Undecidability of hhp-bisimilarity)
Hhp-bisimilarity is undecidable for finite labelled asynchronous transition systems.

Theorem 4.2 (Undecidability of unlabelled hhp-bisimilarity)
Hhp-bisimilarity is undecidable for finite unlabelled asynchronous transition systems.

The process of checking hhp-bisimilarity of asynchronous transition systems is conveniently viewed as a game played on runs of the systems by two players: *Spoiler* and *Duplicator*. Duplicator aims to prove the systems to be bisimilar while Spoiler tries to show the opposite [22, 27, 28].

Definition 4.4 *(Hhp-bisimilarity checking game).*
Let $\mathsf{A}_\mathsf{i} = (\mathsf{S}_\mathsf{i}, \mathsf{s}_\mathsf{i}^{\mathrm{ini}}, \mathsf{E}_\mathsf{i}, \rightarrow_\mathsf{i}, \mathsf{L}, \lambda_\mathsf{i}, \mathsf{I}_\mathsf{i})$ *for* $\mathsf{i} \in \{1,2\}$ *be labelled asynchronous transition systems.* Configurations *of the* hhp-bisimilarity checking game $\mathcal{B}_{\mathrm{hhp}}(\mathsf{A}_1, \mathsf{A}_2)$ *are elements of the set* $\mathtt{Runs}(\mathsf{A}_1) \times \mathtt{Runs}(\mathsf{A}_2)$. *Game* $\mathcal{B}_{\mathrm{hhp}}(\mathsf{A}_1, \mathsf{A}_2)$ *is played by two players:* Spoiler *and* Duplicator. *The* initial configuration *is the pair of empty runs* $(\varepsilon, \varepsilon)$. *In each* move *the players change the current configuration* $(\overline{\mathsf{e}_1}, \overline{\mathsf{e}_2})$ *of* $\mathcal{B}_{\mathrm{hhp}}(\mathsf{A}_1, \mathsf{A}_2)$ *in one of the following ways chosen by Spoiler.*

- *Forward move:*

  1. *Spoiler chooses an* $\mathsf{i} \in \{1,2\}$ *and an event* $\mathsf{e}_\mathsf{i} \in \mathsf{E}_\mathsf{i}$, *such that* $\overline{\mathsf{e}_\mathsf{i}} \cdot \mathsf{e}_\mathsf{i} \in \mathtt{Runs}(\mathsf{A}_\mathsf{i})$;
  2. *Duplicator responds by choosing an event* $\mathsf{e}_{3-\mathsf{i}} \in \mathsf{E}_{3-\mathsf{i}}$, *such that* $\overline{\mathsf{e}_{3-\mathsf{i}}} \cdot \mathsf{e}_{3-\mathsf{i}} \in \mathtt{Runs}(\mathsf{A}_{3-\mathsf{i}})$, *and* $\lambda_1(\mathsf{e}_1) = \lambda_2(\mathsf{e}_2)$;

  the pair $(\overline{\mathsf{e}_1} \cdot \mathsf{e}_1, \overline{\mathsf{e}_2} \cdot \mathsf{e}_2)$ *becomes the current configuration.*

- *Backward move:*

  1. *Spoiler chooses an* $\mathsf{i} \in \{1,2\}$ *and a* $\mathsf{k} \in \mathrm{MR}(\overline{\mathsf{e}_\mathsf{i}})$;
  2. *Duplicator can only respond if* $\mathsf{k} \in \mathrm{MR}(\overline{\mathsf{e}_{3-\mathsf{i}}})$; *otherwise Duplicator gets stuck;*

  if $\mathsf{k} \in \mathrm{MR}(\overline{\mathsf{e}_1})$, *and* $\mathsf{k} \in \mathrm{MR}(\overline{\mathsf{e}_2})$ *then* $(\overline{\mathsf{e}_1} \oslash \mathsf{k}, \overline{\mathsf{e}_2} \oslash \mathsf{k})$ *becomes the current configuration.*

A play *of* $\mathcal{B}_{\mathrm{hhp}}(\mathsf{A}_1, \mathsf{A}_2)$ *if a* maximal *sequence of configurations formed by players making moves in the fashion described above. Duplicator is the winner in every infinite play; a finite play is lost by the player who is stuck. Note that Spoiler gets stuck only if both transition systems have no transitions going out from their initial states.*

We skip the tedious details of formalizing notions of strategies and winning strategies for either of the players. The following standard fact is proved by arguing that an hhp-bisimulation is a good formalization of the notion of a winning strategy for Duplicator in an hhp-bisimilarity checking game [22].

Proposition 4.3 *Asynchronous transition systems* A_1 *and* A_2 *are hhp-bisimilar if and only if there is a winning strategy for Duplicator in hhp-bisimilarity checking game* $\mathcal{B}_{\mathrm{hhp}}(\mathsf{A}_1, \mathsf{A}_2)$.

It is easy to see how an hhp-bisimulation $B \subseteq \mathtt{Runs}(A_1) \times \mathtt{Runs}(A_2)$ can serve as a winning strategy for Duplicator in $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$. Intuitively, the hhp-bisimulation B contains all configurations of $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$ which can become current configurations when Duplicator is following the strategy determined by B. The strategy is defined as follows. Let $(\overline{e_1}, \overline{e_2})$ be the current configuration of $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$. If Spoiler chooses event e_i of A_i in a forward move, then Duplicator responds by choosing an event e_{3-i} of A_{3-i}, such that $(\overline{e_1}\cdot e_1, \overline{e_2}\cdot e_2) \in B$. If Spoiler makes a backward move then response of Duplicator is unique. This strategy contains the initial configuration $(\varepsilon, \varepsilon)$ by condition 1. of the definition of an hhp-bisimulation, and it is well defined by conditions 2.–4.

It is not very hard to argue that bisimilarity checking games are determined.

Proposition 4.4 (Determinacy) *Bisimilarity checking games are determined, i.e., in every game exactly one of the players has a winning strategy.*

It follows that if one of the players does not have a winning strategy in a bisimilarity checking game then the other player does.

4.3 Domino bisimilarity is undecidable

In this section we introduce a novel algorithmic problem of checking domino bisimilarity for labelled origin constrained tiling systems and an equivalent problem of solving domino bisimilarity checking games. The main results are the undecidability of the problem and its extension to the unlabelled case. These results serve crucially in Section 4.4 to establish the main result of the paper, i.e., the undecidability of hhp-bisimilarity for finite, both labelled and unlabelled, asynchronous transition systems and elementary net systems.

In Subsection 4.3.1 we define the algorithmic problem of checking domino bisimilarity. In Subsection 4.3.2 we recall 2-counter machines [20] and in Subsection 4.3.3 we prove the undecidability of labelled domino bisimilarity by a reduction from the halting problem for 2-counter machines. Finally, in Subsection 4.3.4 we extend the undecidability result to the unlabelled case.

It is worthwhile to compare our domino bisimilarity, or equivalently domino bisimilarity checking games, to domino snakes studied by Etzion-Petruschka et al. [5] and domino games of Grädel [10]. Despite apparent similarities of our domino bisimilarity checking games to the latter, our games seem to be of quite a different flavour and indeed the proofs of undecidability are very different.

4.3.1 Domino bisimilarity

Definition 4.5 *(Origin constrained tiling system).*
An origin constrained tiling system $T = \big(D, D^{\mathrm{ori}}, (H, H^0), (V, V^0), L, \lambda\big)$ *consists of a set* D *of* dominoes*, its subset* $D^{\mathrm{ori}} \subseteq D$ *called the* origin constraint*, two* horizontal compatibility *relations* $H, H^0 \subseteq D^2$*, two* vertical compatibility *relations* $V, V^0 \subseteq D^2$*, a set* L *of* labels*, and a* labelling function $\lambda : D \to L$.

A *configuration* of T is a triple $(d, x, y) \in D \times \mathbb{N} \times \mathbb{N}$, such that if $x = y = 0$ then $d \in D^{\mathrm{ori}}$. In other words, in the "origin" position $(x, y) = (0, 0)$ of the

non-negative integer grid only dominoes from the origin constraint D^{ori} are allowed.

Let (d, x, y), and (d', x', y') be configurations of T such that $|x' - x| + |y' - y| = 1$, i.e., the positions (x, y), and (x', y') are neighbours in the non-negative integer grid. Without loss of generality we may assume that $x + y < x' + y'$. We say that configurations (d, x, y), and (d', x', y') are *compatible* if either of the two conditions below holds:

- $x' = x$, and $y' = y + 1$, and
  if $y = 0$, then $(d, d') \in V^0$, and if $y > 0$, then $(d, d') \in V$, or

- $x' = x + 1$, and $y' = y$, and
  if $x = 0$, then $(d, d') \in H^0$, and if $x > 0$, then $(d, d') \in H$.

Definition 4.6 *(Domino bisimulation).*
Let $T_i = \left(D_i, D_i^{ori}, (H_i, H_i^0), (V_i, V_i^0), L_i, \lambda_i\right)$ *for* $i \in \{1, 2\}$ *be origin constrained tiling systems. A relation* $B \subseteq D_1 \times D_2 \times \mathbb{N} \times \mathbb{N}$ *is a* domino bisimulation *relating* T_1 *and* T_2*, if* $(d_1, d_2, x, y) \in B$ *implies that* $\lambda_1(d_1) = \lambda_2(d_2)$*, and the following conditions are satisfied for all* $i \in \{1, 2\}$*:*

1. *for all* $d_i \in D_i^{ori}$*, there is* $d_{3-i} \in D_{3-i}^{ori}$*, such that* $\lambda_1(d_1) = \lambda_2(d_2)$*, and* $(d_1, d_2, 0, 0) \in B$*,*

2. *for all* $x, y \in \mathbb{N}$*, such that* $(x, y) \neq (0, 0)$*, and* $d_i \in D_i$*, there is* $d_{3-i} \in D_{3-i}$*, such that* $\lambda_1(d_1) = \lambda_2(d_2)$*, and* $(d_1, d_2, x, y) \in B$*,*

3. *if* $(d_1, d_2, x, y) \in B$*, then for all neighbours* $(x', y') \in \mathbb{N} \times \mathbb{N}$ *of* (x, y)*, and* $d_i' \in D_i$*, if configurations* (d_i, x, y)*, and* (d_i', x', y') *of* T_i *are compatible, then there exists* $d_{3-i}' \in D_{3-i}$*, such that* $\lambda_1(d_1') = \lambda_2(d_2')$*, and configurations* (d_{3-i}, x, y)*, and* (d_{3-i}', x', y') *of* T_{3-i} *are compatible, and* $(d_1', d_2', x', y') \in B$*.*

We say that two tiling systems are *domino bisimilar* if and only if there is a domino bisimulation relating them.

The main result of this section is the following theorem proved in Subsection 4.3.3.

Theorem 4.3 (Undecidability of domino bisimilarity)
Domino bisimilarity is undecidable for origin constrained tiling systems.

The proof is a reduction from the halting problem for deterministic 2-counter machines. For a deterministic 2-counter machine M we define in Subsection 4.3.3 two origin constrained tiling systems T_1, and T_2, such that the following holds.

Lemma 4.1 *Machine* M *does not halt if and only if there is a domino bisimulation relating* T_1 *and* T_2*.*

The process of checking domino bisimilarity of origin constrained tiling systems is conveniently viewed as a game played on an infinite grid by two players: Spoiler and Duplicator. As in the case of hhp-bisimilarity checking games Duplicator aims to prove the tiling systems to be bisimilar while Spoiler tries to show the opposite.

Definition 4.7 *(Origin constrained domino bisimilarity checking game). Let* T_1 *and* T_2 *be origin constrained tiling systems.* Configurations *of the* origin constrained domino bisimilarity checking game $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$ *are elements of the set* $\mathsf{D}_1 \times \mathsf{D}_2 \times \mathbb{N} \times \mathbb{N}$. *Game* $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$ *is played by two players* Spoiler *and* Duplicator.

- *First the players fix an* initial configuration*:*
  1. *Spoiler chooses an* $\mathfrak{i} \in \{1, 2\}$*, and a configuration* $(\mathsf{d}_\mathfrak{i}, \mathsf{x}, \mathsf{y})$ *of* $\mathsf{T}_\mathfrak{i}$*,*
  2. *Duplicator responds by choosing a domino* $\mathsf{d}_{3-\mathfrak{i}} \in \mathsf{D}_{3-\mathfrak{i}}$*, such that* $(\mathsf{d}_{3-\mathfrak{i}}, \mathsf{x}, \mathsf{y})$ *is a configuration of* $\mathsf{T}_{3-\mathfrak{i}}$*, and* $\lambda_1(\mathsf{d}_1) = \lambda_2(\mathsf{d}_2)$*;*

  if both players were able to make their choices then the tuple $(\mathsf{d}_1, \mathsf{d}_2, \mathsf{x}, \mathsf{y})$ *becomes the* current configuration *of* $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$*.*
- *In each* move *of the domino bisimilarity checking game the players change the current configuration* $(\mathsf{d}_1, \mathsf{d}_2, \mathsf{x}, \mathsf{y})$*:*
  1. *Spoiler chooses an* $\mathfrak{i} \in \{1, 2\}$*, and a configuration* $(\mathsf{d}'_\mathfrak{i}, \mathsf{x}', \mathsf{y}')$ *of* $\mathsf{T}_\mathfrak{i}$ *compatible with configuration* $(\mathsf{d}_\mathfrak{i}, \mathsf{x}, \mathsf{y})$*,*
  2. *Duplicator responds by choosing* $\mathsf{d}'_{3-\mathfrak{i}} \in \mathsf{D}_{3-\mathfrak{i}}$ *so that* $(\mathsf{d}'_{3-\mathfrak{i}}, \mathsf{x}', \mathsf{y}')$ *is a configuration of* $\mathsf{T}_{3-\mathfrak{i}}$*, and* $\lambda_1(\mathsf{d}_1) = \lambda_2(\mathsf{d}_2)$*, and configurations* $(\mathsf{d}_{3-\mathfrak{i}}, \mathsf{x}, \mathsf{y})$ *and* $(\mathsf{d}'_{3-\mathfrak{i}}, \mathsf{x}', \mathsf{y}')$ *of* $\mathsf{T}_{3-\mathfrak{i}}$ *are compatible;*

  if both players were able to make their choices then the tuple $(\mathsf{d}'_1, \mathsf{d}'_2, \mathsf{x}', \mathsf{y}')$ *becomes the* current configuration *of* $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$*.*

A play *of* $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$ *is a* maximal *sequence of configurations formed by players making moves in the fashion described above. Duplicator is the winner in every infinite play; a finite play is lost by the player who is stuck.*

We avoid tedious details of formalizing notions of strategies and winning strategies for either of the players. The following simple fact is proved by arguing that a domino bisimulation is a good formalization of a winning strategy for Duplicator in a domino bisimilarity checking game.

Proposition 4.5 *Origin constrained tiling systems* T_1 *and* T_2 *are domino bisimilar if and only if Duplicator has a winning strategy in the domino bisimilarity game* $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$*.*

As in the case of hhp-bisimilarity checking games it is easy to argue that domino bisimilarity checking games are determined; in other words Proposition 4.4 holds also for domino bisimilarity checking games.

4.3.2 Counter machines

A 2-counter machine M consists of a finite program with the set L of instruction labels, and instructions of the form:

- ℓ: $\mathtt{c_i := c_i + 1; goto\ m}$
- ℓ: $\mathtt{if\ c_i = 0\ then\ c_i := c_i + 1; goto\ m}$ $\mathtt{else\ c_i := c_i - 1; goto\ n}$
- $\mathtt{halt}$:

where $\mathtt{i} = 1, 2$; $\ell, \mathtt{m}, \mathtt{n} \in \mathsf{L}$, and $\{\mathtt{start}, \mathtt{halt}\} \subseteq \mathsf{L}$. A configuration of M is a triple $(\ell, \mathtt{x}, \mathtt{y}) \in \mathsf{L} \times \mathbb{N} \times \mathbb{N}$, where ℓ is the label of the current instruction, and $\mathtt{x}$ and $\mathtt{y}$ are the values stored in counters $\mathtt{c_1}$ and $\mathtt{c_2}$, respectively; we denote the set of configurations of M by $\mathtt{Confs}(\mathsf{M})$. The semantics of 2-counter machines is standard: let $\vdash_{\mathsf{M}} \subseteq \mathtt{Confs}(\mathsf{M}) \times \mathtt{Confs}(\mathsf{M})$ be the usual one-step derivation relation on configurations of M; by $\vdash_{\mathsf{M}}^{+}$ we denote the reachability (in at least one step) relation for configurations, i.e., the transitive closure of $\vdash_{\mathsf{M}}$.

Before we give a reduction from the halting problem of 2-counter machines to origin constrained domino bisimilarity let us take a look at the directed graph $(\mathtt{Confs}(\mathsf{M}), \vdash_{\mathsf{M}})$, with configurations of M as vertices, and edges denoting derivation in one step. Since machine M is deterministic, for each configuration there is at most one outgoing edge; moreover only halting configurations have no outgoing edges. It follows that connected components of the graph $(\mathtt{Confs}(\mathsf{M}), \vdash_{\mathsf{M}})$ are either trees with edges going to the root which is the unique halting configuration in the component, or have no halting configuration at all. This observation is formalized in the following proposition.

Proposition 4.6 *Let M be a 2-counter machine. The following conditions are equivalent:*

1. *machine M halts on input $(0, 0)$, i.e., $(\mathtt{start}, 0, 0) \vdash_{\mathsf{M}}^{+} (\mathtt{halt}, \mathtt{x}, \mathtt{y})$ for some $\mathtt{x}, \mathtt{y} \in \mathbb{N}$,*

2. *$(\mathtt{start}, 0, 0) \sim_{\mathsf{M}} (\mathtt{halt}, \mathtt{x}, \mathtt{y})$ for some $\mathtt{x}, \mathtt{y} \in \mathbb{N}$, where the relation $\sim_{\mathsf{M}} \subseteq \mathtt{Confs}(\mathsf{M}) \times \mathtt{Confs}(\mathsf{M})$ is the symmetric and transitive closure of $\vdash_{\mathsf{M}}$.*

4.3.3 The reduction

In this subsection we give a proof of Theorem 4.3 by proving Lemma 4.1. The idea is to design a tiling system which "simulates" the behaviour of a 2-counter machine.

Let M be a 2-counter machine. We construct a tiling system T_{M} with the set L of instruction labels of M as the set of dominoes, and the identity function on L as the labelling function. Note that this implies that all tuples belonging to a domino bisimulation relating copies of T_{M} are of the form $(\ell, \ell, \mathtt{x}, \mathtt{y})$, so we can identify them with configurations of M, i.e., sometimes we will make no distinction between $(\ell, \ell, \mathtt{x}, \mathtt{y})$ and $(\ell, \mathtt{x}, \mathtt{y}) \in \mathtt{Confs}(\mathsf{M})$ for $\ell \in \mathsf{L}$.

We define the horizontal compatibility relations $\mathsf{H}_{\mathsf{M}}, \mathsf{H}_{\mathsf{M}}^{0} \subseteq \mathsf{L} \times \mathsf{L}$ of the tiling system T_{M} as follows:

- $(\ell, m) \in H_M$ if and only if either of the instructions below is an instruction of machine M:
  - ℓ: `c1 := c1 + 1; goto` m
  - m: `if c1 = 0 then c1 := c1 + 1; goto` n `else c1 := c1 - 1; goto` ℓ
- $(\ell, m) \in H^0_M$ if and only if $(\ell, m) \in H_M$, or the instruction below is an instruction of machine M:
  - ℓ: `if c1 = 0 then c1 := c1 + 1; goto` m `else c1 := c1 - 1; goto` n

Vertical compatibility relations V_M, and V^0_M are defined in the same way, with c_1 instructions replaced with c_2 instructions. We also take $D^{ori}_M = L$, i.e., all dominoes are allowed in position $(0, 0)$. Note that the identity function is a 1-1 correspondence between configurations of M, and configurations of the tiling system T_M; from now on we will hence identify configurations of M and T_M. It follows immediately from the construction of T_M, that two configurations $c, c' \in \mathtt{Confs}(M)$ are compatible as configurations of T_M, if and only if $c \vdash_M c'$, or $c' \vdash_M c$, i.e., compatibility relation of T_M coincides with the symmetric closure of $\vdash_M$. By $\approx_M$ we denote the symmetric and transitive closure of the compatibility relation of configurations of T_M. The following proposition is then straightforward.

Proposition 4.7 *The two relations $\sim_M$ and $\approx_M$ coincide.*

Now we are ready to define the two origin constrained tiling systems T_1, and T_2, postulated in Lemma 4.1. The idea is to have two independent and slightly pruned copies of T_M in T_2: one without the initial configuration $(\mathtt{start}, 0, 0)$, and the other without any halting configurations $(\mathtt{halt}, x, y)$. The other tiling system T_1 is going to have three independent copies of T_M: the two of T_2, and moreover, another full copy of T_M.

More formally we define $D_2 = \big(L \times \{1, 2\}\big) \setminus \big\{(\mathtt{halt}, 2)\big\}$, and $D^{ori}_2 = D_2 \setminus \big\{(\mathtt{start}, 1)\big\}$, and $V_2 = \big((V_M \otimes 1) \cup (V_M \otimes 2)\big) \cap (D_2 \times D_2)$, where for a binary relation R we define $R \otimes i$ to be the relation $\big\{\, \big((a, i), (b, i)\big) : (a, b) \in R \,\big\}$. The other compatibility relations V^0_2, H_2, and H^0_2 are defined analogously from the respective compatibility relations of T_M.

The tiling system T_1 is obtained from T_2 by adding yet another independent copy of T_M, this time a complete one: $D_1 = D_2 \cup (L \times \{3\})$, and $D^{ori}_1 = D^{ori}_2 \cup (L \times \{3\})$, and $V_1 = V_2 \cup (V_M \otimes 3)$, etc.. The labelling functions of T_1, and T_2 are defined as $\lambda_i\big((\ell, i)\big) = \ell$.

In order to show Lemma 4.1, and hence conclude the proof of Theorem 4.3, it suffices to establish the following two lemmas.

Lemma 4.2 *If machine M halts on input $(0, 0)$ then origin constrained tiling systems T_1 and T_2 are not domino bisimilar.*

Proof. By Proposition 4.5 it suffices to show that if machine M halts on input $(0,0)$ then Spoiler has a winning strategy in the game $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1,\mathsf{T}_2)$. Spoiler starts by choosing the configuration $\big((\mathtt{start},3),0,0\big)$ of T_1. Duplicator has to respond with domino $(\mathtt{start},2)$ of T_2 since $(\mathtt{start},1)\notin \mathsf{D}_2^{\mathrm{ori}}$. Then Spoiler "simulates" the finite computation of M on input $(0,0)$ in the following way. If $\big((\ell,3),(\ell,2),x,y\big)$ is the current configuration of the game then Spoiler chooses the configuration $\big((\ell',3),x',y'\big)$ of T_1, such that $\big(\ell,x,y\big)\vdash_{\mathsf{M}}(\ell',x',y')$. This move is allowed thanks to Proposition 4.7. Then Duplicator can only respond with domino $(\ell',2)$ of T_2, and $\big((\ell',3),(\ell',2),x',y'\big)$ becomes the current configuration of the game. In the last step of the simulation Spoiler chooses a configuration $\big((\mathtt{halt},3),x',y'\big)$ for some $x',y'\in\mathbb{N}$ which makes Duplicator stuck because $(\mathtt{halt},2)\notin \mathsf{D}_2$. □

Lemma 4.3 *If machine* M *does not halt on input* $(0,0)$ *then origin constrained tiling systems* T_1 *and* T_2 *are domino bisimilar.*

Proof. By Proposition 4.5 it suffices to show that if machine M does not halt on input $(0,0)$ then Duplicator has a winning strategy in the game $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1,\mathsf{T}_2)$. We claim that the following is a winning strategy for Duplicator.

If in the first step Spoiler chooses a configuration $\big((\ell,j),x,y\big)$ of T_1 or T_2 for $j\in\{1,2\}$, then Duplicator responds with the domino (ℓ,j) of the other tiling system. It is obvious that then Duplicator can respond to all moves of Spoiler because both players play on identical pruned copies of T_{M}.

If instead Spoiler chooses a configuration $\big((\ell,3),x,y\big)$ of T_1 in the first step then Duplicator responds with:

- domino $(\ell,1)$ of T_2 if $(\ell,x,y)\sim_{\mathsf{M}}(\mathtt{halt},x',y')$ or $(\ell,x,y)=(\mathtt{halt},x',y')$ for some $x',y'\in\mathbb{N}$, and
- domino $(\ell,2)$ of T_2 if $(\ell,x,y)\not\sim_{\mathsf{M}}(\mathtt{halt},x',y')$ for all $x',y'\in\mathbb{N}$.

In the first case the only way Spoiler can make Duplicator stuck is to be able to choose configuration $\big((\mathtt{start},3),0,0\big)$ of T_1 since the only difference between copy 3 of T_{M} in T_1 and copy 1 of T_{M} in T_2 is that the latter does not have the triple $(\mathtt{start},0,0)$ as a configuration. Hence in order to prove that Duplicator has a winning strategy from the initial configuration $\big((\ell,3),(\ell,1),x,y\big)$, it suffices to show that $(\ell,x,y)\not\approx_{\mathsf{M}}(\mathtt{start},0,0)$. Assume for the sake of contradiction that $(\ell,x,y)\approx_{\mathsf{M}}(\mathtt{start},0,0)$. By Proposition 4.7 we then have $(\ell,x,y)\sim_{\mathsf{M}}(\mathtt{start},0,0)$. This, by our assumption that $(\ell,x,y)\sim_{\mathsf{M}}(\mathtt{halt},x',y')$ for some $x',y'\in\mathbb{N}$, implies that $(\mathtt{start},0,0)\sim_{\mathsf{M}}(\mathtt{halt},x',y')$ for some $x',y'\in\mathbb{N}$. Then Proposition 4.6 implies that $(\mathtt{start},0,0)\vdash^{+}_{\mathsf{M}}(\mathtt{halt},x',y')$, which contradicts the assumption of the lemma that machine M does not halt on input $(0,0)$.

The argument in the other case is similar. It suffices to show that $(\ell,x,y)\not\approx_{\mathsf{M}}(\mathtt{halt},x',y')$ for all $x',y'\in\mathbb{N}$, because the only difference between copy 3 of T_{M} in T_1 and copy 2 of T_{M} in T_2 is that the latter has no triple $(\mathtt{halt},x',y')$ as a configuration. By applying Proposition 4.7 to our assumption that $(\ell,x,y)\not\sim_{\mathsf{M}}(\mathtt{halt},x',y')$ for all $x',y'\in\mathbb{N}$, we immediately get that $(\ell,x,y)\not\approx_{\mathsf{M}}(\mathtt{halt},x',y')$ for all $x',y'\in\mathbb{N}$. □

4.3.4 Undecidability of unlabelled domino bisimilarity

In this section we argue that the problem of deciding domino bisimilarity is undecidable even for unlabelled origin constrained tiling systems or, equivalently, for origin constrained tiling systems with a singleton set of labels.

Theorem 4.4 (Undecidability of unlabelled domino bisimilarity) *Unlabelled domino bisimilarity is undecidable for origin constraint tiling systems.*

Proof. We show that the unlabelled origin constrained domino bisimilarity checking game is undecidable and then we use Proposition 4.5.

Let $\mathsf{T}_i = \big(\mathsf{D}_i, \mathsf{D}_i^{\mathrm{ori}}, (\mathsf{H}_i, \mathsf{H}_i^0), (\mathsf{V}_i, \mathsf{V}_i^0), \mathsf{L}_i, \lambda_i\big)$, for $i \in \{1,2\}$, be origin constrained tiling systems. Without loss of generality we may assume that $\mathsf{L}_1 = \mathsf{L}_2 = \{1, \ldots, n\}$, for some $n \geq 1$. We give an effective construction of unlabelled origin constrained tiling systems $\overline{\mathsf{T}}_i = \big(\overline{\mathsf{D}}_i, \mathsf{D}_i^{\mathrm{ori}}, (\overline{\mathsf{H}}_i, \overline{\mathsf{H}}_i^0), (\overline{\mathsf{V}}_i, \overline{\mathsf{V}}_i^0)\big)$, for $i \in \{1,2\}$, such that the following holds.

Lemma 4.4 (The reduction) *Duplicator has a winning strategy in the labelled domino bisimilarity checking game $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$ if and only if he has a winning strategy in the unlabelled game $\mathcal{B}_{\mathrm{d}}(\overline{\mathsf{T}}_1, \overline{\mathsf{T}}_2)$.*

Establishing this lemma will complete the proof of Theorem 4.4. Tiling systems $\overline{\mathsf{T}}_i$, for $i \in \{1,2\}$, are defined as follows:

- $\overline{\mathsf{D}}_i = \mathsf{D}_i \cup \{c_d^1, \ldots, c_d^{\lambda_i(d)} : d \in \mathsf{D}_i\} \cup \{b_d : d \in \mathsf{D}_i\}$,
  the dominoes in D_i are called *plain dominoes* and the ones in $\overline{\mathsf{D}}_i \setminus \mathsf{D}_i$ are called *auxiliary dominoes*,
- $\overline{\mathsf{H}}_i = \mathsf{H}_i \cup \big\{(d, c_d^1) : d \in \mathsf{D}_i\big\} \cup \big\{(c_d^i, c_d^{i+1}) : d \in \mathsf{D}_i \text{ and } 1 \leq i < \lambda_i(d)\big\}$,
- $\overline{\mathsf{H}}_i^0 = \mathsf{H}_i^0 \cup \big\{(d, c_d^1) : d \in \mathsf{D}_i\big\} \cup \big\{(c_d^i, c_d^{i+1}) : d \in \mathsf{D}_i \text{ and } 1 \leq i < \lambda_i(d)\big\}$,
- $\overline{\mathsf{V}}_i = \mathsf{V}_i \cup \big\{(d, b_d) : d \in \mathsf{D}_i\big\}$,
- $\overline{\mathsf{V}}_i^0 = \mathsf{V}_i^0 \cup \big\{(d, b_d) : d \in \mathsf{D}_i\big\}$.

For the proof of the "if" part of Lemma 4.4 the following two facts will be instrumental.

Lemma 4.5 *Let (d_1, d_2, x, y) be a configuration of the unlabelled domino bisimulation checking game $\mathcal{B}_{\mathrm{d}}(\overline{\mathsf{T}}_1, \overline{\mathsf{T}}_2)$, such that d_i is a plain domino and d_{3-i} is an auxiliary domino, for some $i \in \{1,2\}$. Then Spoiler has a winning strategy from this configuration in the unlabelled game $\mathcal{B}_{\mathrm{d}}(\overline{\mathsf{T}}_1, \overline{\mathsf{T}}_2)$.*

Proof. Consider the case $i = 1$; the other case is symmetric. Spoiler wins immediately by making the move $(b_{d_1}, x, y+1)$: Duplicator has no response to this move. □

Lemma 4.6 *Let (d_1, d_2, x, y) be a configuration of the unlabelled domino bisimilarity checking game $\mathcal{B}_d(T_1, T_2)$, such that $\lambda_1(d_1) \neq \lambda_2(d_2)$. Then Spoiler has a winning strategy from this configuration in the unlabelled game $\mathcal{B}_d(\overline{T}_1, \overline{T}_2)$.*

Proof. Without loss of generality assume that $\lambda_1(d_1) > \lambda_2(d_2)$. Let Spoiler play the following sequence of moves from the configuration (d_1, d_2, x, y) in the unlabelled game $\mathcal{B}_d(\overline{T}_1, \overline{T}_2)$:

$$(c^1_{d_1}, x+1, y), (c^2_{d_1}, x+2, y), \ldots, \left(c^{\lambda_2(d_2)}_{d_1}, x+\lambda_2(d_2), y\right).$$

The only way for Duplicator to avoid losing immediately as in Lemma 4.5 is to respond with the following sequence of moves:

$$(c^1_{d_2}, x+1, y), (c^2_{d_2}, x+2, y), \ldots, \left(c^{\lambda_2(d_2)}_{d_2}, x+\lambda_2(d_2), y\right).$$

From the configuration $\left(c^{\lambda_2(d_2)}_{d_1}, c^{\lambda_2(d_2)}_{d_2}, x+\lambda_2(d_2), y\right)$ Spoiler wins immediately by playing the move $(c^{\lambda_2(d_2)+1}_{d_1}, x+\lambda_2(d_2)+1, y)$: Duplicator has no response to this move. □

We are now ready to establish the "if" part of Lemma 4.4.

Lemma 4.7 ("if") *If Duplicator has a winning strategy in the unlabelled domino bisimilarity checking game $\mathcal{B}_d(\overline{T}_1, \overline{T}_2)$ then he has a winning strategy in the labelled game $\mathcal{B}_d(T_1, T_2)$.*

Proof. Note that every configuration of the labelled domino bisimilarity checking game $\mathcal{B}_d(T_1, T_2)$ is also a configuration of the unlabelled domino bisimilarity checking game $\mathcal{B}_d(\overline{T}_1, \overline{T}_2)$. Given a winning strategy for Duplicator in the unlabelled game $\mathcal{B}_d(\overline{T}_1, \overline{T}_2)$ we define a strategy for Duplicator in the labelled game $\mathcal{B}_d(T_1, T_2)$ to be equal to the restriction of the former strategy to configurations and moves of the labelled game $\mathcal{B}_d(T_1, T_2)$. This strategy is well defined because of Lemmas 4.5 and 4.6; it is clearly a winning strategy. □

Finally, we conclude the proof of Theorem 4.4 by sketching a proof of the "only if" part of Lemma 4.4.

Lemma 4.8 ("only if") *If Duplicator has a winning strategy in the labelled domino bisimilarity checking game $\mathcal{B}_d(T_1, T_2)$ then he has a winning strategy in the unlabelled game $\mathcal{B}_d(\overline{T}_1, \overline{T}_2)$.*

Proof. Suppose that Duplicator has a winning strategy in the labelled domino bisimilarity checking game $\mathcal{B}_d(T_1, T_2)$. A configuration of the unlabelled game $\mathcal{B}_d(\overline{T_1}, \overline{T_2})$ is called *admissible* if it is also a configuration of the labelled game $\mathcal{B}_d(T_1, T_2)$ and it belongs to the winning strategy for Duplicator there, i.e., if

it is reachable by Duplicator playing the strategy. We define the strategy for Duplicator in the unlabelled game $\mathcal{B}_d(\overline{T_1}, \overline{T_2})$ to be equal to the strategy in the labelled game for all admissible configurations and moves in which Spoiler chooses a plain domino. We need to define the strategy for Duplicator for the configurations that are not admissible or moves in which Spoiler chooses an auxiliary domino.

The strategy we define for Duplicator in the unlabelled game $\mathcal{B}_d(\overline{T_1}, \overline{T_2})$ has the property that every configuration $C = (d_1, d_2, x, y)$ that belongs to the strategy, i.e., that can be reached when Duplicator follows the strategy, is one of the following forms:

1. the configuration C belongs to the winning strategy for Duplicator in the labelled game $\mathcal{B}_d(T_1, T_2)$;
2. $d_1 = c^k_{d_3}$ and $d_2 = c^k_{d_4}$, for some plain dominoes $d_3 \in D_1$ and $d_4 \in D_2$, and $\lambda_1(d_3) = \lambda_2(d_4)$, and if $x \geq k$ then the configuration $(d_3, d_4, x - k, y)$ belongs to the winning strategy for Duplicator in the labelled game $\mathcal{B}_d(T_1, T_2)$;
3. $d_1 = b_{d_3}$ and $d_2 = b_{d_4}$, for some plain dominoes $d_3 \in D_1$ and $d_4 \in D_2$, and $\lambda_1(d_3) = \lambda_2(d_4)$, and if $y > 0$ then the configuration $(d_3, d_4, x, y - 1)$ belongs to the winning strategy for Duplicator in the labelled game $\mathcal{B}_d(T_1, T_2)$.

It is a tedious but routine exercise to inspect that if Spoiler plays from a configuration of one of the forms 1.–3. listed above then Duplicator can always respond so as to make the next configuration fall into one of the categories 1.–3. □ □

4.4 Hhp-bisimilarity is undecidable

The main result of this section is a proof of Theorem 4.1, i.e., the undecidability of hhp-bisimilarity for finite state asynchronous transition systems. We also give a few extensions of this result: undecidability of hhp-bisimilarity for finite unlabelled asynchronous transition systems, for finite (unlabelled) elementary net systems and 1-safe Petri nets.

The proof of our main result is a reduction from the problem of deciding domino bisimilarity for origin constrained tiling systems shown to be undecidable in the previous section. The method of encoding a tiling system on an infinite grid in the unfolding of a finite asynchronous transition system is due to Madhusudan and Thiagarajan [18]. For each origin constrained tiling system T, we give an effective definition a finite asynchronous transition system $A(T)$, such that the following holds.

Lemma 4.9 (The reduction) *Origin constrained tiling systems* T_1 *and* T_2 *are domino bisimilar if and only if asynchronous transition systems* $A(T_1)$ *and* $A(T_2)$ *are hhp-bisimilar.*

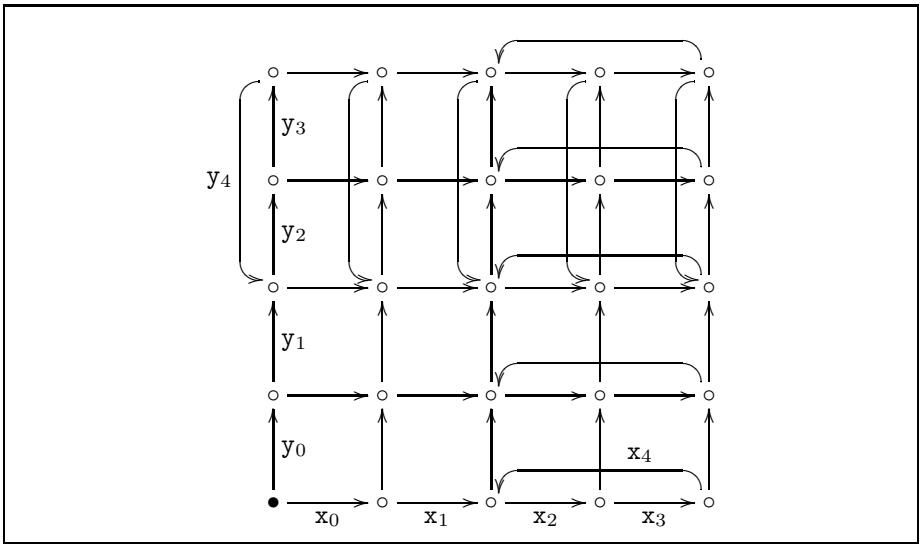


Figure 4.1: Modelling the infinite grid.

In Subsections 4.4.1–4.4.3 we define the finite asynchronous transition system $A(T)$ and prove Lemma 4.9 thus completing the proof of Theorem 4.1. In Subsection 4.4.4 we prove Theorem 4.2, i.e., we show that hhp-bisimilarity is undecidable even for unlabelled finite asynchronous transition systems. Finally, in Subsection 4.4.5 we argue that the hhp-bisimilarity undecidability results hold for the class of asynchronous transition systems induced by elementary net systems and 1-safe Petri nets.

4.4.1 Asynchronous transition system $A(T)$

Let $T = \big(D, D^{ori}, (H, H^0), (V, V^0), L, \lambda\big)$ be an origin constrained tiling system. The infinite grid structure is modelled by the unfolding of the asynchronous transition system shown in Figure 4.1. The set of events of this asynchronous transition system is $E = \{x_0, x_1, x_2, x_3, x_4, y_0, y_1, y_2, y_3, y_4\}$. The independence relation I is the symmetric closure of $\big\{\ (x_i, y_j)\ :\ i, j \in \{0,1,2,3,4\}\ \big\}$.

We identify the states of the asynchronous transition system in Figure 4.1 with pairs of numbers $(i, j) \in \{0,1,2,3,4\}^2$, where i is the horizontal coordinate and j is the vertical coordinate. The state in the bottom-left corner in Figure 4.1 is $(0,0)$; it is the initial state. For all $n \in \mathbb{N}$, define:

$$\widehat{n} = \begin{cases} n & \text{if } n \leq 4, \\ 2 + \big((n-2) \bmod 3\big) & \text{if } n > 4. \end{cases}$$

A position $(n, m) \in \mathbb{N}^2$ of the infinite grid is represented by state $(\widehat{n}, \widehat{m})$ in the asynchronous transition system $A(T)$.

Configurations of the tiling system T are modelled by extra transitions going out of states of the grid structure in Figure 4.1, and labelled by events of the form d_{ij}, for $d \in D$, and $i, j \in \{0,1,2,3\}$. We define a set of events E_D as follows:

$$E_D = \{\ d_{ij}\ :\ d \in D;\ \text{and } i, j \in \{0,1,2,3\};\ \text{and } i = j = 0 \text{ implies } d \in D^{ori}\ \}.$$

The idea is, for every $d \in D$, to have a transition going out of each state $(i,j) \in \{0,1,2,3,4\}^2$ labelled with d_{ij}, provided that (d,i,j) is a configuration of T. In fact, for a technical reason we need to use events d_{i1} and d_{1j} at states $(i,4)$ and $(4,j)$, for $i,j \in \{0,1,2,3\}$, respectively, instead of d_{i4} and d_{4j}. In order to avoid special treatment of this case throughout the rest of the paper we adopt the following notation, for all $n \in \mathbb{N}$:

$$\widetilde{n} = \begin{cases} n & \text{if } n \leq 3, \\ 1 + \big((n-1) \bmod 3\big) & \text{if } n > 3. \end{cases}$$

Horizontal and vertical compatibility relations for configurations of the tiling system T are modelled by an independence relation I_D on E_D, according to which events d_{ij} and $e_{k\ell}$ corresponding to "neighbouring" configurations are independent if and only if the configurations are compatible. More precisely, we define I_D to be the symmetric closure of the following set:

$$\begin{array}{l} \big\{\, (d_{0j}, e_{1j}) \,:\, j \in \{0,1,2,3\} \text{ and } (d,e) \in H_0 \,\big\} \,\cup \\ \big\{\, (d_{ij}, e_{\widetilde{(i+1)}j}) \,:\, i \in \{1,2,3\}, j \in \{0,1,2,3\}, \text{ and } (d,e) \in H \,\big\} \,\cup \\ \big\{\, (d_{i0}, e_{i1}) \,:\, i \in \{0,1,2,3\} \text{ and } (d,e) \in V_0 \,\big\} \,\cup \\ \big\{\, (d_{ij}, e_{i\widetilde{(j+1)}}) \,:\, i \in \{0,1,2,3\}, j \in \{1,2,3\}, \text{ and } (d,e) \in V \,\big\}. \end{array}$$

For all $i,j \in \{0,1,2,3,4\}$, and $d \in D$, we have up to four transitions going out of state (i,j) and labelled by the following events in E_D: $d_{\widetilde{i}\widetilde{j}}, d_{(i-1)\widetilde{j}}$, if $i > 0$, $d_{\widetilde{i}(j-1)}$ if $j > 0$, and $d_{(i-1)(j-1)}$ if $i,j > 0$. We write $(i,j,\{d_{i'j'}\})$ to denote the state reached by the transition labelled by the event $d_{i'j'}$ going out of state (i,j). In other words, for all $i,j \in \{0,1,2,3,4\}$ we have the following transitions:

- $(i,j) \xrightarrow{d_{\widetilde{i}\,\widetilde{j}}} \big(i,j,\{d_{\widetilde{i}\,\widetilde{j}}\}\big)$,
- $(i,j) \xrightarrow{d_{(i-1)\widetilde{j}}} \big(i,j,\{d_{(i-1)\widetilde{j}}\}\big)$ if $i > 0$,
- $(i,j) \xrightarrow{d_{\widetilde{i}(j-1)}} \big(i,j,\{d_{\widetilde{i}(j-1)}\}\big)$ if $j > 0$,
- $(i,j) \xrightarrow{d_{(i-1)(j-1)}} \big(i,j,\{d_{(i-1)(j-1)}\}\big)$ if $i,j > 0$.

Moreover, if there are transitions:

- $(i,j) \xrightarrow{d_{k\ell}} \big(i,j,\{d_{k\ell}\}\big)$, and
- $(i,j) \xrightarrow{e_{k'\ell'}} \big(i,j,\{e_{k'\ell'}\}\big)$,

and $(d_{k\ell}, e_{k'\ell'}) \in I_D$, then there is also a state $\big(i,j,\{d_{k\ell}, e_{k'\ell'}\}\big)$ and transitions:

- $\big(i,j,\{d_{k\ell}\}\big) \xrightarrow{e_{k'\ell'}} \big(i,j,\{d_{k\ell}, e_{k'\ell'}\}\big)$, and
- $\big(i,j,\{e_{k'\ell'}\}\big) \xrightarrow{d_{k\ell}} \big(i,j,\{d_{k\ell}, e_{k'\ell'}\}\big)$.

Finally, there are transitions:

- $\big(i,j,\{d_{\widetilde{i\,j}}\}\big) \xrightarrow{x_i} \big(\widehat{i+1},j,\{d_{\widetilde{i\,j}}\}\big)$, if $\big(i,j,\{d_{\widetilde{i\,j}}\}\big)$ is a state, and
- $\big(i,j,\{d_{\widetilde{i(j-1)}}\}\big) \xrightarrow{x_i} \big(\widehat{i+1},j,\{d_{\widetilde{i(j-1)}}\}\big)$, if $\big(i,j,\{d_{\widetilde{i(j-1)}}\}\big)$ is a state,

and transitions:

- $\big(i,j,\{d_{\widetilde{i\,j}}\}\big) \xrightarrow{y_j} \big(i,\widehat{j+1},\{d_{\widetilde{i\,j}}\}\big)$, if $\big(i,j,\{d_{\widetilde{i\,j}}\}\big)$ is a state, and
- $\big(i,j,\{d_{\widetilde{(i-1)j}}\}\big) \xrightarrow{y_j} \big(i,\widehat{j+1},\{d_{\widetilde{(i-1)j}}\}\big)$, if $\big(i,j,\{d_{\widetilde{(i-1)j}}\}\big)$ is a state.

The sets of states $S_{A(T)}$ and transitions $\rightarrow_{A(T)}$ of the asynchronous transition system $A(T) = (S_{A(T)}, s^{ini}_{A(T)}, E_{A(T)}, \rightarrow_{A(T)}, \lambda_{A(T)}, I_{A(T)})$ are as described above. The set of events is defined by $E_{A(T)} = E \cup E_D$. The initial state is $s^{ini}_{A(T)} = (0,0)$. The independence relation $I_{A(T)}$ is defined as the symmetric closure of the set:

$$I \cup I_D \cup \big\{ (x_i, d_{\widetilde{i\,j}}), (y_j, d_{\widetilde{i\,j}}) : i,j \in \{0,1,2,3,4\} \text{ and } d_{\widetilde{i\,j}} \in E_D \big\}.$$

Finally, the labelling function $\lambda_{A(T)}$ is an identity on E, and for elements of E_D it replaces the dominoes with their labels in the tiling system T, i.e.,

$$\lambda_{A(T)}(e) = \begin{cases} e & \text{if } e \in E, \\ \big(\lambda(d)\big)_{ij} & \text{if } e \in E_D \text{ and } e = d_{ij}. \end{cases}$$

Proposition 4.8 *The labelled transition systems $A(T)$ is a labelled asynchronous transition system.*

4.4.2 The unfolding of $A(T)$

In this subsection we sketch the structure of the unfolding $\mathtt{Unf}\big(A(T)\big)$ of asynchronous transition system $A(T)$ defined in the previous subsection.

For notational convenience we will write $(i,j,\emptyset)$ for a state (i,j) of $A(T)$. In order to avoid heavy use of notations $\widehat{n}$ and $\widetilde{m}$ we adopt the following conventions:

- we write x_n and y_m, for all $n,m \in \mathbb{N}$, to denote events $x_{\widehat{n}}, y_{\widehat{m}} \in E$, respectively;
- we write d_{nm} to denote an event $d_{\widetilde{n}\widetilde{m}} \in E_D$, for all $n,m \in \mathbb{N}$.

Proposition 4.9 *The set of states of $\mathtt{Unf}\big(A(T)\big)$ reachable from the initial state $(0,0,\emptyset)$ consists of triples $(n,m,C) \in \mathbb{N} \times \mathbb{N} \times \wp(E_D)$, such that either:*

- $C = \emptyset$; *or*
- $C = \{d_{n'm'}\}$ *such that* $d_{n'm'} \in E_D$, *and* $n' \in \{n-1, n\}$, *and* $m' \in \{m-1, m\}$; *or*
- $C = \{d_{(n-1)m'}, e_{nm'}\}$ *such that* $d_{(n-1)m'}, e_{nm'} \in E_D$, *and* $m' \in \{m-1, m\}$, *and configurations* $(d, n-1, m')$ *and* (e, n, m') *of* T *are compatible; or*

- $C = \{d_{n'(m-1)}, e_{n'm}\}$ *such that* $d_{n'(m-1)}, e_{n'm} \in E_D$, *and* $n' \in \{n-1, n\}$, *and configurations* $(d, n', m-1)$ *and* (e, n', m) *of* T *are compatible.*

States of the first category above represent positions on the infinite grid; in particular the state $(n, m, \emptyset)$ represents the position $(n, m) \in \mathbb{N} \times \mathbb{N}$. States of the second category above represent configurations of the tiling system T; in particular configuration $(d, n, m) \in D \times \mathbb{N} \times \mathbb{N}$ is represented by states $\big(n', m', \{d_{nm}\}\big)$ for $n' \in \{n, n+1\}$ and $m' \in \{m, m+1\}$. States of the third and forth categories above are used to "check compatibility" of neighbouring configurations of the tiling system T.

Proposition 4.10 *The set of transitions of* $\mathtt{Unf}\big(A(T)\big)$ *consists of the following:*

- $(n, m, C) \xrightarrow{x_n}_{\mathtt{Unf}(A(T))} (n+1, m, C)$
  for $C = \emptyset$, *or for* $C = \{d_{nm'}\}$, *where* $m' \in \{m-1, m\}$,

- $(n, m, C) \xrightarrow{y_m}_{\mathtt{Unf}(A(T))} (n, m+1, C)$
  for $C = \emptyset$, *or for* $C = \{d_{n'm}\}$, *where* $n' \in \{n-1, n\}$,

- $(n, m, \emptyset) \xrightarrow{d_{n'm'}}_{\mathtt{Unf}(A(T))} \big(n, m, \{d_{n'm'}\}\big)$
  for $n' \in \{n-1, n\}$, *and* $m' \in \{m-1, m\}$, *and* $d_{n'm'} \in E_D$,

- $\big(n, m, \{d_{(n-1)m'}\}\big) \xrightarrow{e_{nm'}}_{\mathtt{Unf}(A(T))} \big(n, m, \{d_{(n-1)m'}, e_{nm'}\}\big)$ *and*
  $\big(n, m, \{e_{nm'}\}\big) \xrightarrow{d_{(n-1)m'}}_{\mathtt{Unf}(A(T))} \big(n, m, \{d_{(n-1)m'}, e_{nm'}\}\big)$,
  for $m' \in \{m-1, m\}$ *if configurations* $(d, n-1, m')$ *and* (e, n, m') *of* T *are compatible,*

- $\big(n, m, \{d_{n'(m-1)}\}\big) \xrightarrow{e_{n'm}}_{\mathtt{Unf}(A(T))} \big(n, m, \{d_{n'(m-1)}, e_{n'm}\}\big)$ *and*
  $\big(n, m, \{e_{n'm}\}\big) \xrightarrow{d_{n'(m-1)}}_{\mathtt{Unf}(A(T))} \big(n, m, \{d_{n'(m-1)}, e_{n'm}\}\big)$
  for $n' \in \{n-1, n\}$, *if configurations* $(d, n', m-1)$ *and* (e, n', m) *of* T *are compatible.*

4.4.3 Translations between hhp- and domino bisimulations

By Proposition 4.2 it follows that in order to prove Lemma 4.9 it suffices to demonstrate that a domino bisimulation relating T_1 and T_2 gives rise to an hhp-bisimulation relating $\mathtt{Unf}\big(A(T_1)\big)$ and $\mathtt{Unf}\big(A(T_2)\big)$, and vice versa.

In other words, by Propositions 4.3 and 4.5, it suffices to argue that a winning strategy for Duplicator in $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$ can be translated to a winning strategy for him in $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2))\big)$, and vice versa. In what follows, in order to keep the arguments from becoming too dull or cumbersome, we are mixing freely at our convenience the two ways of talking about bisimulations: as relations, and as winning strategies in bisimilarity checking games.

For notational convenience we introduce the following convention for writing elements of an hhp-bisimulation relating $\mathtt{Unf}\big(A(T_1)\big)$ and $\mathtt{Unf}\big(A(T_2)\big)$, or equivalently, for configurations of game $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2))\big)$. Note

that if a pair of runs $(\overline{e_1}, \overline{e_2}) \in \mathtt{Runs}\big(\mathtt{Unf}(A(T_1))\big) \times \mathtt{Runs}\big(\mathtt{Unf}(A(T_2))\big)$ belongs to an hhp-bisimulation then the states reached by these runs are of the forms (n, m, C_1) and (n, m, C_2) for some $n, m \in \mathbb{N}$, respectively. In what follows we write (n, m, C_1, C_2) to denote such a pair $(\overline{e_1}, \overline{e_2})$. This notation is a bit sloppy because it is not 1-1. For example, $(1, 1, \emptyset)$ is used to denote both (x_0y_0, x_0y_0) and (y_0x_0, y_0x_0). It is not hard to see that this sloppiness is not a problem here.

From domino to hhp-bisimulation. Let $B \subseteq D_1 \times D_2 \times \mathbb{N} \times \mathbb{N}$ be a domino bisimulation relating T_1 and T_2. We define a winning strategy for Duplicator in game $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2))\big)$ in the following way.

If Spoiler makes a backward move then the response of Duplicator is determined uniquely. Moreover, this response can always be performed because asynchronous transition systems $\mathtt{Unf}\big(A(T_1)\big)$ and $\mathtt{Unf}\big(A(T_2)\big)$ have the property that every pair of runs with equal labelling sequences has equal sets of most recent events. If Spoiler makes a forward move by choosing an event x_n or y_m, for $n, m \in \mathbb{N}$, then Duplicator responds with the same event in the other transition system.

The only non-trivial responses of Duplicator are the ones to be made when Spoiler makes a forward move by choosing an event of the form d_{nm}, where d is a domino, and $n, m \in \mathbb{N}$. We define these responses by referring to the domino bisimulation B. The strategy for Duplicator we define below has the following property.

Property 4.1 *Suppose that a configuration* (n, m, C_1, C_2) *of an hhp-bisimilarity checking game* $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}\big(A(T_1)\big), \mathtt{Unf}\big(A(T_2)\big)\big)$ *can be reached from the initial configuration while Duplicator is playing according to the strategy. Then* $d_{n'm'} \in C_1$ *and* $e_{n'm'} \in C_2$*, for* $n' \in \{n-1, n\}$ *and* $m' \in \{m-1, m\}$*, imply that* $(d, e, n', m') \in B$.

Suppose without loss of generality that Spoiler makes a move in $\mathtt{Unf}\big(A(T_1)\big)$; the other case is symmetric. We consider several cases depending on the current configuration of $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}\big(A(T_1)\big), \mathtt{Unf}\big(A(T_2)\big)\big)$.

- The current configuration of the game $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2))\big)$ is

  $$(n, m, \emptyset, \emptyset)$$

  for some $n, m \in \mathbb{N}$. Spoiler can choose an event $d_{n'm'}$, such that $n' \in \{n-1, n\}$ and $m' \in \{m-1, m\}$. Then Duplicator responds with an event $e_{n'm'}$ in $\mathtt{Unf}\big(A(T_2)\big)$, such that $(d, e, n', m') \in B$.

- The current configuration of the game $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2))\big)$ is

  $$\big(n, m, \{d_{n'm'}\}, \{e_{n'm'}\}\big)$$

  such that $n' \in \{n-1, n\}$ and $m' \in \{m-1, m\}$. Spoiler can choose an event $d'_{k\ell}$, such that either $k = n'$ and $\{m', \ell\} = \{m-1, m\}$, or $\ell = m'$ and $\{n', k\} = \{n-1, n\}$. In both cases Duplicator responds with an event $e'_{k\ell}$, such that configurations (e, n', m') and (e', k, ℓ) of T_2 are compatible, and $(d', e', k, \ell) \in B$.

Note that all the responses we have defined above are indeed possible due to Property 4.1 and the definition of a domino bisimulation, and moreover, they maintain Property 4.1.

From hhp- to domino bisimulation. Let B be an hhp-bisimulation relating $\mathtt{Unf}\big(\mathsf{A}(\mathsf{T}_1)\big)$ and $\mathtt{Unf}\big(\mathsf{A}(\mathsf{T}_2)\big)$. We define a winning strategy for Duplicator in game $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$. The strategy for Duplicator we define below has the following property.

Property 4.2 *If configuration* $(\mathsf{d}, \mathsf{e}, n, m)$ *of* $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$ *can be reached while Duplicator is playing according to the strategy then* $\big(n, m, \{\mathsf{d}_{nm}\}, \{\mathsf{e}_{nm}\}\big) \in \mathsf{B}$.

Suppose without loss of generality that Spoiler makes a move in T_1; the other case is symmetric. We consider the two kinds of moves possible in a domino bisimilarity game.

- In order to fix an initial configuration of $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$ Spoiler chooses a configuration (d, n, m) of T_1. Note that for all $n, m \in \mathbb{N}$, we have that $(n, m, \emptyset, \emptyset) \in \mathsf{B}$. Let e_{nm} be Duplicator's response if Spoiler makes a forward move in the game $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}\big(\mathsf{A}(\mathsf{T}_1)\big), \mathtt{Unf}\big(\mathsf{A}(\mathsf{T}_2)\big)\big)$ by choosing event d_{nm} in configuration $(n, m, \emptyset, \emptyset)$. Then we take e to be Duplicator's response to Spoiler's choice of configuration (d, n, m).

- Let $(\mathsf{d}, \mathsf{e}, n, m)$ be the current configuration of $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$. In a next move Spoiler can choose a configuration (d', n', m') of T_1 compatible with (d, n, m). We consider cases when $(n', m') = (n-1, m)$ and $(n', m') = n, m+1)$; the other two cases are analogous. Note that by Property 4.2 we have that
$$\big(n, m, \{\mathsf{d}_{nm}\}, \{\mathsf{e}_{nm}\}\big) \in \mathsf{B}. \tag{4.1}$$

  - Let $(n', m') = (n-1, m)$. Since configurations (d, n, m) and $(\mathsf{d}', n-1, m)$ of T_1 are compatible, by applying condition 2. of the definition of an hhp-bisimulation to (4.1) we get that there is a domino e' of T_2, such that configurations (e, n, m) and $(\mathsf{e}', n-1, m)$ of T_2 are compatible, and
$$\big(n, m, \{\mathsf{d}'_{(n-1)m}, \mathsf{d}_{nm}\}, \{\mathsf{e}'_{(n-1)m}, \mathsf{e}_{nm}\}\big) \in \mathsf{B}. \tag{4.2}$$
    We define event e' to be Duplicator's response in $\mathcal{B}_{\mathrm{d}}(\mathsf{T}_1, \mathsf{T}_2)$ for Spoiler's move consisting of choosing configuration $(\mathsf{d}', n-1, m)$ of T_1. By applying condition 4. of the definition of an hhp-bisimulation to (4.2) twice we get that
$$\big(n-1, m, \{\mathsf{d}'_{(n-1)m}\}, \{\mathsf{e}'_{(n-1)m}\}\big) \in \mathsf{B}.$$

  - Let $(n', m') = (n, m+1)$. By applying condition 2. of the definition of an hhp-bisimulation to (4.1) we get that
$$\big(n, m+1, \{\mathsf{d}_{nm}\}, \{\mathsf{e}_{nm}\}\big) \in \mathsf{B}. \tag{4.3}$$

Since configurations (d, n, m) and $(d', n, m+1)$ of T_1 are compatible, by applying condition 2. of the definition of an hhp-bisimulation to (4.3) we get that there is a domino e' of T_2, such that configurations (e, n, m) and $(e', n, m+1)$ of T_2 are compatible, and

$$\big(n, m+1, \{d_{nm}, d'_{n(m+1)}\}, \{e_{nm}, e'_{n(m+1)}\}\big) \in B. \tag{4.4}$$

We define event e' to be Duplicator's response in $\mathcal{B}_d(T_1, T_2)$ for Spoiler's move consisting of choosing configuration $(d', n, m+1)$ of T_1. By applying condition 4. of the definition of an hhp-bisimulation to (4.4) we get

$$\big(n, m+1, \{d'_{n(m+1)}\}, \{e'_{n(m+1)}\}\big) \in B.$$

Note that all the responses we have defined above are indeed possible due to Property 4.2 and the definition of a domino bisimulation, and moreover, they maintain Property 4.2.

4.4.4 Unlabelled hhp-bisimilarity

In this subsection we prove Theorem 4.2, i.e., we show that undecidability of hhp-bisimilarity holds also for unlabelled asynchronous transition systems. For an unlabelled origin constrained tiling system T we give an effective definition of an asynchronous transition system $A'(T)$, which is obtained by only a slight modification of the asynchronous transition system $A(T)$: we add new states s_1, s_2, s_3, s_4, s_5, and new transitions $(0,1) \xrightarrow{c} s_1 \xrightarrow{c} s_2 \xrightarrow{c} s_3 \xrightarrow{c} s_4 \xrightarrow{c} s_5$, where c is a new event. The independence relation of $A'(T)$ is the same as in $A(T)$. There is no labelling function in $A'(T)$.

Obviously, if Duplicator has a winning strategy in the labelled hhp-bisimilarity checking game $\mathcal{B}_{hhp}\big(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2))\big)$ then he has also a winning strategy in the unlabelled game $\mathcal{B}_{hhp}\big(\mathtt{Unf}(A'(T_1)), \mathtt{Unf}(A'(T_2))\big)$. By determinacy of bisimilarity checking games the following lemma is the converse of the previous statement.

Lemma 4.10 (The reduction) *If Spoiler has a winning strategy in the labelled hhp-bisimilarity checking game $\mathcal{B}_{hhp}\big(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2))\big)$ then he has a winning strategy in the unlabelled game $\mathcal{B}_{hhp}\big(\mathtt{Unf}(A'(T_1)), \mathtt{Unf}(A'(T_2))\big)$.*

The rest of this section is devoted to proving the above lemma from which Theorem 4.2 follows by Theorem 4.4 and the proof of Theorem 4.1.

For notational convenience, we sometimes identify configurations of the games $\mathcal{B}_{hhp}\big(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2))\big)$ or $\mathcal{B}_{hhp}\big(\mathtt{Unf}(A'(T_1)), \mathtt{Unf}(A'(T_2))\big)$, respectively, with pairs of states of the transition systems $\mathtt{Unf}\big(A(T_1)\big)$ and $\mathtt{Unf}\big(A(T_2)\big)$, or from the transition systems $\mathtt{Unf}\big(A'(T_1)\big)$ and $\mathtt{Unf}\big(A'(T_2)\big)$, respectively. The following observation will be useful.

Lemma 4.11 *Let (n_1, m_1, C_1) and (n_2, m_2, C_2) be states of $\mathtt{Unf}\big(A'(T_1)\big)$ and $\mathtt{Unf}\big(A'(T_2)\big)$, respectively, such that $|C_1| \neq |C_2|$. Then Spoiler has a winning strategy in $\mathcal{B}_{hhp}\big(\mathtt{Unf}(A'(T_1)), \mathtt{Unf}(A'(T_2))\big)$ from these states.*

Proof. Note that $|C_1|, |C_2| \in \{0, 1, 2\}$. If $|C_i| = 0$ and $|C_{3-i}| \neq 0$, for $i \in \{1, 2\}$, then an infinite number of forward moves is possible from (n_i, m_i, C_i), and only finitely many from the state $(n_{3-i}, m_{3-i}, C_{3-i})$. Hence Spoiler has a winning strategy in this case.

If $|C_1| > 0$, $|C_2| > 0$, and $|C_1| \neq |C_2|$ then a winning strategy for Spoiler is as follows. Assume without loss of generality that $|C_1| = 1$; then of course $|C_2| = 2$. Spoiler makes a backward move in the first component, such that the next configuration consists of states $(n_1, m_1, \emptyset)$ and (n_2', m_2', C_2'), where $|C_2'| \neq 0$. By the preceding argument it follows that Spoiler has a winning strategy from this configuration. □

The argument to prove Lemma 4.10 is as follows. Consider a winning strategy for Spoiler in the game $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2))\big)$. A configuration of the unlabelled hhp-bisimilarity checking game $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A'(T_1)), \mathtt{Unf}(A'(T_2))\big)$ is called *admissible* if the corresponding configuration in the labelled hhp-bisimilarity checking game $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2))\big)$ belongs to the winning strategy for Spoiler, i.e., if this configuration is reachable by playing the strategy from the initial configuration $(0, 0, \emptyset, \emptyset)$. Obviously, the initial configuration is admissible. We define a strategy for Spoiler in the unlabelled game to be equal to the strategy in the labelled game in all admissible configurations. In a sequence of lemmas (Lemmas 4.12–4.15 below) we argue that in all admissible configurations, every response of Duplicator which leads to a non-admissible configuration enables Spoiler to win in a finite number of steps.

Lemma 4.12 *Let* $(0, 0, \emptyset)$ *and* $(0, 0, \emptyset)$ *be a pair of states of* $\mathtt{Unf}\big(A'(T_1)\big)$ *and* $\mathtt{Unf}\big(A'(T_2)\big)$*, respectively. If Spoiler chooses the move* $(0, 0, \emptyset) \xrightarrow{x_0} (1, 0, \emptyset)$ *then Duplicator must answer with the same move in the other system; otherwise Spoiler can win in a finite number of steps. The similar holds if Spoiler chooses the move* $(0, 0, \emptyset) \xrightarrow{y_0} (0, 1, \emptyset)$*.*

Proof. By Lemma 4.11 we know that Duplicator cannot choose any transition $(0, 0, \emptyset) \xrightarrow{d_{00}} \big(0, 0, \{d_{00}\}\big)$, for $d \in D$. Suppose then that his choice was $(0, 0, \emptyset) \xrightarrow{y_0} (0, 1, \emptyset)$. We argue that Spoiler has a winning strategy now. He plays $(0, 1, \emptyset) \xrightarrow{c} s_1$. If Duplicator answers with some $(n, m, \emptyset)$ then he loses because of similar arguments as in the proof of Lemma 4.11. In any other case (after performing this move) there are at most three possible forward moves for Duplicator, whereas Spoiler can perform another four forward moves. Therefore, Spoiler has a winning strategy. □

Lemma 4.13 *Note that* $(n, m, \emptyset, \emptyset)$ *is an admissible configuration in the unlabelled hhp-bisimilarity checking game* $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A'(T_1)), \mathtt{Unf}(A'(T_2))\big)$*, for all* $n, m \in \mathbb{N}$*. If Spoiler chooses the move* $(n, m, \emptyset) \xrightarrow{x_n} (n + 1, m, \emptyset)$ *then Duplicator must answer with the same move in the other system; otherwise Spoiler can win in a finite number of steps. The similar holds if Spoiler chooses the move* $(n, m, \emptyset) \xrightarrow{y_m} (n, m + 1, \emptyset)$*.*

Proof. The special case when $n = m = 0$ is proved in Lemma 4.12. By Lemma 4.11 it follows that Duplicator cannot choose any transition of the form $(n, m, \emptyset) \xrightarrow{d_{n'm'}} \big(n, m, \{d_{n'm'}\}\big)$. Suppose that the response of Duplicator is $(n, m, \emptyset) \xrightarrow{y_m} (n, m+1, \emptyset)$. Assume without loss of generality that the last event in the run $\overline{e}$ leading to the state $(n, m, \emptyset)$ was y_{m-1}. Now, Spoiler can win immediately by taking a backwards move since $n + m \in \mathrm{MR}(\overline{e} \cdot x_n)$ and $n + m \notin \mathrm{MR}(\overline{e} \cdot y_m)$. □

Lemma 4.14 *Note that* $(n, m, \emptyset, \emptyset)$ *is an admissible configuration in the unlabelled hhp-bisimilarity checking game* $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A'(T_1)), \mathtt{Unf}(A'(T_2))\big)$*, for all* $n, m \in \mathbb{N}$*. If Spoiler chooses a move*

$$(n, m, \emptyset) \xrightarrow{d_{n'm}} \big(n, m, \{d_{n'm'}\}\big),$$

for $n' \in \{n-1, n\}$*,* $m' \in \{m-1, m\}$*, and* $d_{n'm'} \in E_{D_1}$*, then Duplicator must answer with*

$$(n, m, \emptyset) \xrightarrow{e_{n'm}} \big(n, m, \{e_{n'm'}\}\big)$$

in the other system, for some $e_{n'm'} \in E_{D_2}$*; otherwise Spoiler can win in a finite number of steps.*

Proof. By Lemma 4.11 Duplicator must respond with a move of the form

$$(n, m, \emptyset) \xrightarrow{e_{n''m}} \big(n, m, \{e_{n''m''}\}\big).$$

We show that Spoiler has a winning strategy in all cases where $n' \neq n''$ or $m' \neq m''$. We use Lemma 4.11 without explicitly mentioning it.

- If $n' = n-1$ and $m' = m-1$ then Duplicator looses for any other choice of indices n'' and m''. The reason is that from the state $\big(n, m, \{d_{n'm'}\}\big)$ no forward move that preserves the cardinality of the set $\{d_{n'm'}\}$ is available, and instead from the state $\big(n, m, \{e_{n''m''}\}\big)$, a forward move can be made by choosing either the event x_n or y_m.

- If $n' = n$ and $m' = m$ then two forward moves are possible for Spoiler from the state $\big(n, m, \{d_{n'm'}\}\big)$ that preserve the cardinality of the set $\{d_{n'm'}\}$. If Duplicator chooses a different index n'' or m'' then he can perform at most one forward move preserving the cardinality of the set $\{e_{n''m''}\}$.

- If $n' = n - 1$ and $m' = m$ then the only choice for Duplicator, that does not respect indices n' and m', is $n'' = n$ and $m'' = m - 1$ since for the other choices where $n'' = n$ and $m'' = m$, or $n'' = n - 1$ and $m'' = m-1$, Duplicator looses because of the previous arguments. Consider the states $\big(n, m, \{d_{(n-1)m}\}\big)$ and $\big(n, m, \{e_{n(m-1)}\}\big)$. We show that Spoiler has a winning strategy starting from these states. Spoiler chooses

$$\big(n, m, \{d_{(n-1)m}\}\big) \xrightarrow{y_m} \big(n, m+1, \{d_{(n-1)m}\}\big)$$

and Duplicator's only response is

$$\big(n, m, \{e_{n(m-1)}\}\big) \xrightarrow{x_n} \big(n+1, m, \{e_{n(m-1)}\}\big).$$

However, Spoiler can now perform a backwards move reaching the state $\big(n, m+1, \emptyset\big)$ and Duplicator's only response is by the same backwards move, reaching the state $\big(n+1, m, \emptyset\big)$. Spoiler has a winning strategy now, by the arguments from Lemma 4.13.

- The case where $n' = n$ and $m' = m - 1$ is similar to the previous one.

□

Lemma 4.15 *Let* $\big(n, m, \{d_{n'm'}\}, \{e_{n'm'}\}\big)$ *be an admissible configuration in the unlabelled hhp-bisimilarity checking game* $\mathcal{B}_{\mathrm{hhp}}\big(\mathtt{Unf}(A'(T_1)), \mathtt{Unf}(A'(T_2))\big)$. *If Spoiler chooses a move*

$$(n, m, \{d_{n'm'}\}) \xrightarrow{d'_{n''m''}} \big(n, m, \{d_{n'm'}, d'_{n''m''}\}\big),$$

for some $n' \in \{n-1, n\}$, $m' \in \{m-1, m\}$, *and* $d'_{n''m''} \in E_{D_1}$, *then Duplicator must answer with*

$$(n, m, \{e_{n'm'}\}) \xrightarrow{e'_{n''m''}} \big(n, m, \{e_{n'm'}, e'_{n''m''}\}\big)$$

in the other system, for some $e'_{n''m''} \in E_{D_2}$; *otherwise Spoiler can win in a finite number of steps.*

We omit the proof of this lemma; similar arguments can be used as in the proof of Lemma 4.14.

Observe that all the E_D-events in the labelled asynchronous transition system $A(T)$ have the same label since T is by our assumption an unlabelled origin constrained tiling system. Therefore, Lemmas 4.12–4.15 cover all the relevant cases in the argument for Lemma 4.10 sketched before Lemma 4.12. This concludes the reduction of unlabelled domino bisimilarity to unlabelled hhp-bisimilarity, and hence Theorem 4.2 follows from Theorem 4.4.

4.4.5 Finite elementary net system $N(T)$

In this subsection we argue that undecidability of hhp-bisimilarity for finite elementary net systems and 1-safe Petri nets follows as a corollary of our proof for finite asynchronous transition systems.

Given a tiling system T we define an elementary net system $N(T)$ and we argue that $A(T)$ is isomorphic to the asynchronous transition system $na\big(N(T)\big)$ corresponding to the net $N(T)$; see the articles by Nielsen and Winskel [23,30] for the definition of the asynchronous transition system $na\big(N(T)\big)$. The elementary net system $N(T)$ is safe, i.e., all of its reachable markings are contact-free and hence it is a 1-safe Petri net as well. This immediately implies the following facts.

Theorem 4.5 *Hhp-bisimilarity is undecidable for finite labelled elementary net systems and 1-safe Petri nets.*

Theorem 4.6 *Hhp-bisimilarity is undecidable for finite unlabelled elementary net systems and 1-safe Petri nets.*

The elementary net system $\mathsf{N}(\mathsf{T}) = (\mathsf{P}_{\mathsf{N}(\mathsf{T})}, \mathsf{E}_{\mathsf{N}(\mathsf{T})}, \mathtt{pre}_{\mathsf{N}(\mathsf{T})}, \mathtt{post}_{\mathsf{N}(\mathsf{T})}, \mathsf{M}_{\mathsf{N}(\mathsf{T})})$ is shown in Figure 4.2 and it consists of the following:

- the set of conditions

$$\begin{aligned} \mathsf{P}_{\mathsf{N}(\mathsf{T})} &= \big\{\, \mathtt{a}_i, \mathtt{b}_i \,:\, i \in \{0,1,2,3,4\} \,\big\} \\ &\cup \big\{\, \mathtt{a}^j_{i(i+1)}, \mathtt{b}^i_{j(j+1)} \,:\, i,j \in \{0,1,2,3\} \,\big\} \cup \mathsf{E}_{\mathsf{D}}; \end{aligned}$$

- the set of events $\mathsf{E}_{\mathsf{N}(\mathsf{T})} = \mathsf{E}_{\mathsf{A}(\mathsf{T})}$;

- the function $\mathtt{pre}_{\mathsf{N}(\mathsf{T})} : \mathsf{E}_{\mathsf{N}(\mathsf{T})} \to \wp(\mathsf{P}_{\mathsf{N}(\mathsf{T})})$ specifying the set of places in the pre-condition of an event:

$$\mathtt{pre}_{\mathsf{N}(\mathsf{T})}(e) = \begin{cases} \{\mathtt{a}_0\} & \text{if } e = \mathtt{x}_0, \\ \{\mathtt{b}_0\} & \text{if } e = \mathtt{y}_0, \\ \{\mathtt{a}_i\} \cup \mathtt{A}_{(i-1)i} & \text{if } e = \mathtt{x}_i \text{ for } i \in \{1,2,3,4\}, \\ \{\mathtt{b}_j\} \cup \mathtt{B}_{(j-1)j} & \text{if } e = \mathtt{y}_j \text{ for } j \in \{1,2,3,4\}, \\ \{\mathtt{a}^j_{i(i+1)}, \mathtt{b}^i_{j(j+1)}\} \cup \mathtt{Incmpt}(\mathtt{d}_{ij}) & \text{if } e = \mathtt{d}_{ij} \in \mathsf{E}_{\mathsf{D}}, \end{cases}$$

  where for $i,j \in \{0,1,2,3\}$, we define $\mathtt{A}_{i(i+1)} = \big\{\, \mathtt{a}^k_{i(i+1)} \,:\, k \in \{0,1,2,3\} \,\big\}$ and $\mathtt{B}_{j(j+1)} = \big\{\, \mathtt{b}^k_{j(j+1)} \,:\, k \in \{0,1,2,3\} \,\big\}$, and for $\mathtt{d}_{ij} \in \mathsf{E}_{\mathsf{D}}$, we define the set of dominoes *incompatible* with $\mathtt{d}_{ij}$ by:

$$\mathtt{Incmpt}(\mathtt{d}_{ij}) = \{\, e_{k\ell} \in \mathsf{E}_{\mathsf{D}} \,:\, (\mathtt{d}_{ij}, e_{k\ell}) \notin \mathsf{I}_{\mathsf{D}} \,\};$$

- the function $\mathtt{post}_{\mathsf{N}(\mathsf{T})} : \mathsf{E}_{\mathsf{N}(\mathsf{T})} \to \wp(\mathsf{P}_{\mathsf{N}(\mathsf{T})})$ specifying the set of places in the post-condition of an event:

$$\mathtt{post}_{\mathsf{N}(\mathsf{T})}(e) = \begin{cases} \{\mathtt{a}_{i+1}\} \cup \mathtt{A}_{(i+1)(i+2)} & \text{if } e = \mathtt{x}_i \text{ for } i \in \{0,1,2\}, \\ \{\mathtt{a}_4\} \cup \mathtt{A}_{12} & \text{if } e = \mathtt{x}_3, \\ \{\mathtt{a}_2\} \cup \mathtt{A}_{23} & \text{if } e = \mathtt{x}_4, \\ \{\mathtt{b}_{j+1}\} \cup \mathtt{B}_{(j+1)(j+2)} & \text{if } e = \mathtt{y}_j \text{ for } j \in \{0,1,2\}, \\ \{\mathtt{b}_4\} \cup \mathtt{B}_{12} & \text{if } e = \mathtt{y}_3, \\ \{\mathtt{b}_2\} \cup \mathtt{B}_{23} & \text{if } e = \mathtt{y}_4, \\ \emptyset & \text{if } e \in \mathsf{E}_{\mathsf{D}}; \end{cases}$$

- the initial marking $\mathsf{M}_{\mathsf{N}(\mathsf{T})} = \{\mathtt{a}_0, \mathtt{b}_0\} \cup \mathtt{A}_{01} \cup \mathtt{B}_{01} \cup \mathsf{E}_{\mathsf{D}}$.

Proposition 4.11 *The asynchronous transition system* $\mathsf{A}(\mathsf{T})$ *is isomorphic to* $\mathtt{na}\big(\mathsf{N}(\mathsf{T})\big)$.

Proof. We define a function $\Xi : S_{A(T)} \to \wp(P_{N(T)})$ as follows:

$$\Xi\big((i,j,C)\big) = \{a_i, b_j\} \cup X_i \cup Y_j \cup E_D \setminus \bigcup_{c \in C} \mathbf{pre}_{N(T)}(c),$$

where

$$X_i = \begin{cases} A_{01} & \text{if } i = 0, \\ A_{(i-1)i} \cup A_{i(i+1)} & \text{if } i \in \{1,2,3\}, \\ A_{34} \cup A_{12} & \text{if } i = 4, \end{cases}$$

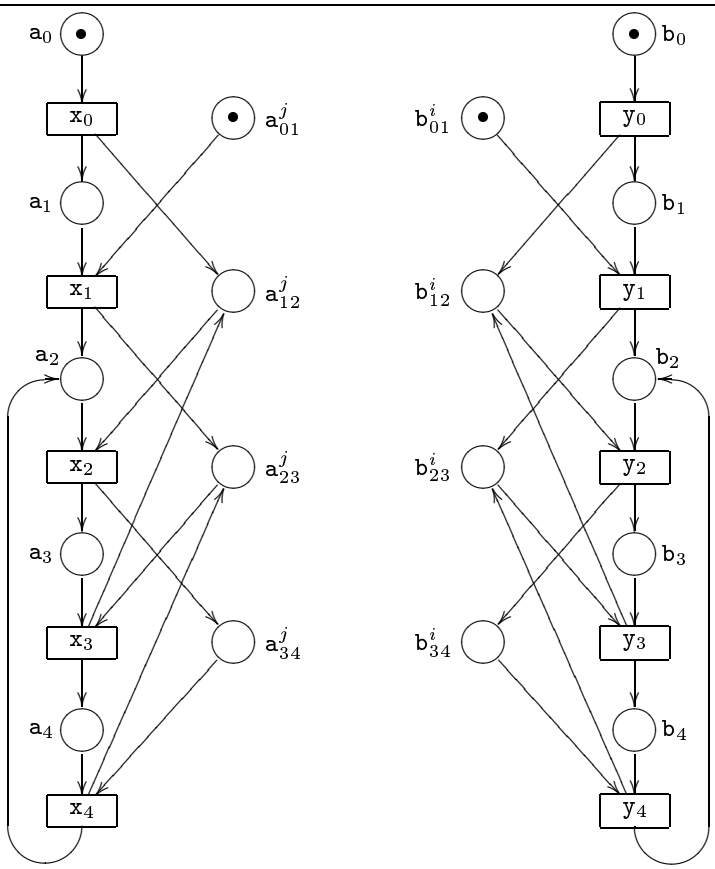
and similarly

$$Y_j = \begin{cases} B_{01} & \text{if } j = 0, \\ B_{(j-1)j} \cup B_{j(j+1)} & \text{if } j \in \{1,2,3\}, \\ B_{34} \cup B_{12} & \text{if } j = 4. \end{cases}$$

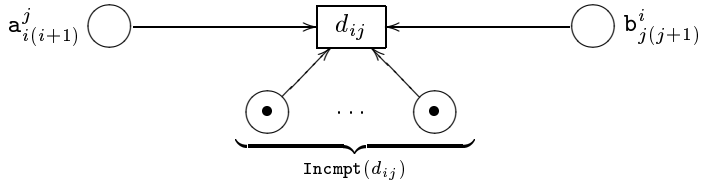
In order to argue that Ξ is an isomorphism of asynchronous transition systems $A(T)$ and $\mathbf{na}\big(N(T)\big)$ it suffices to establish the following:

1. $\Xi(s^{\mathrm{ini}}_{A(T)}) = M_{N(T)}$, i.e., the initial state of $A(T)$ is mapped by Ξ to the initial marking of $N(T)$,
2. for all $s \in S_{A(T)}$,
   (a) if $s \xrightarrow{e}_{A(T)} t$ then $\Xi(s) \xrightarrow{e}_{\mathbf{na}(N(T))} \Xi(t)$,
   (b) if $\Xi(s) \xrightarrow{e}_{\mathbf{na}(N(T))} M$ then there is $t \in S_{A(T)}$, such that $s \xrightarrow{e}_{A(T)} t$ and $M = \Xi(t)$,
3. $(e,f) \in I_{A(T)}$ if and only if ${}^\bullet e^\bullet \cap {}^\bullet f^\bullet = \emptyset$.

It is a tedious but routine exercise to verify that clauses 1.–3. hold. □



Variables i and j range over $\{0,1,2,3\}$. Therefore we have four copies of each place in the second column, for $j \in \{0,1,2,3\}$, and four copies of each place in the third column, for $i \in \{0,1,2,3\}$.



The preconditions of each $d_{ij} \in E_D$, are the two places $\mathtt{a}^j_{i(i+1)}$ and $\mathtt{b}^i_{j(j+1)}$, and all the places in $\mathtt{Incmpt}(d_{ij}) = \{\, e_{k\ell} \in E_D \;:\; (d_{ij}, e_{k\ell}) \notin I_D \,\}$.

Figure 4.2: The elementary net system $\mathsf{N}(\mathsf{T})$.

Bibliography

[1] Marek A. Bednarczyk. Hereditary history preserving bisimulations or what is the power of the future perfect in program logics. Technical report, Polish Academy of Sciences, Gdańsk, April 1991. Available at `http://www.ipipan.gda.pl/~marek`.

[2] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.

[3] Gian Luca Cattani and Vladimiro Sassone. Higher dimensional transition systems. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 55–62, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.

[4] Søren Christensen, Hans Hüttel, and Colin Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121(2):143–148, 1995.

[5] Yael Etzion-Petruschka, David Harel, and Dale Myers. On the solvability of domino snake problems. *Theoretical Computer Science*, 131:243–269, 1994.

[6] Sibylle Fröschle. Decidability of plain and hereditary history-preserving bisimilarity for BPP. In Ilaria Castellani and Björn Victor, editors, *Proceedings of EXPRESS'99 the 6th International Workshop on Expressiveness in Concurrency*, volume 27 of *Electronic Notes in Theoretical Computer Science*, 2000.

[7] Sibylle Fröschle and Thomas Hildebrandt. On plain and hereditary history-preserving bisimulation. In Mirosław Kutyłowski, Leszek Pacholski, and Tomasz Wierzbicki, editors, *Mathematical Foundations of Computer Science 1999, 24th International Symposium, MFCS'99*, volume 1672 of *LNCS*, pages 354–365, Szklarska Poreba, Poland, 6–10 September 1999. Springer-Verlag.

[8] R. J. van Glabbeek. The linear time-branching time spectrum (Extended abstract). In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension*, volume 458 of *LNCS*, pages 278–297, Amsterdam, The Netherlands, 27–30 August 1990. Springer-Verlag.

[9] Rob van Glabbeek and Ursula Goltz. Equivalence notions for concurrent systems and refinement of actions (Extended abstract). In A. Kreczmar and G. Mirkowska, editors, *Mathematical Foundations of Computer Science 1989*, volume 379 of *LNCS*, pages 237–248, Porabka-Kozubnik, Poland, August/September 1989. Springer-Verlag.

[10] Erich Grädel. Domino games and complexity. *SIAM Journal on Computing*, 19(5):787–804, 1990.

[11] Jan Friso Groote and Hans Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):354–371, 1994.

[12] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[13] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158(1–2):143–159, 1996.

[14] Lalita Jategaonkar and Albert R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154:107–143, 1996.

[15] André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996. A preliminary version appeared in *Proceedings of Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 418–427, Montreal, Canada, June 1993. IEEE Computer Society Press.

[16] Marcin Jurdziński and Mogens Nielsen. Hereditary history preserving bisimilarity is undecidable. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 358–369, Lille, France, February 2000. Springer-Verlag.

[17] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[18] P. Madhusudan and P. S. Thiagarajan. Controllers for discrete event systems via morphisms. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR'98, Concurrency Theory, 9th International Conference, Proceedings*, volume 1466 of *LNCS*, pages 18–33, Nice, France, September 1998. Springer-Verlag.

[19] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag, 1980.

[20] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.

[21] Faron Moller and Scott A. Smolka. On the computational complexity of bisimulation. *ACM Computing Surveys*, 27(2):287–289, 1995.

[22] Mogens Nielsen and Christian Clausen. Games and logics for a noninterleaving bisimulation. *Nordic Journal of Computing*, 2(2):221–249, 1995.

[23] Mogens Nielsen and Glynn Winskel. Petri nets and bisimulation. *Theoretical Computer Science*, 153(1–2):211–244, 1996.

[24] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[25] D. M. R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Theoretical Computer Science: 5th GI-Conference*, volume 104 of *LNCS*, pages 167–183. Springer-Verlag, 1981.

[26] A. Rabinovich and B. Trakhtenbrot. Behaviour structures and nets of processes. *Fundamenta Informaticae*, 11:357–404, 1988.

[27] Colin Stirling. Bisimulation, model checking and other games. Notes for Mathfit Instructional Meeting on Games and Computation, available at `http://www.dcs.ed.ac.uk/home/cps`, 1997.

[28] Wolfgang Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science. In M.-C. Gaudel and J.-P. Jouannaud, editors, *TAPSOFT'93: Theory and Practice of Software Development, 4th International Joint Conference CAAP/FASE*, volume 668 of *LNCS*, pages 559–568, Orsay, France, April 1993. Springer-Verlag.

[29] Walter Vogler. Deciding history preserving bisimilarity. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium, ICALP'91*, volume 510 of *LNCS*, pages 493–505, Madrid, Spain, 8–12 July 1991. Springer-Verlag.

[30] Glynn Winskel and Mogens Nielsen. Models for concurrency. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, Semantic Modelling, pages 1–148. Oxford University Press, 1995.

Chapter 5

Recursive Ping-Pong Protocols

The paper *Recursive Ping-Pong Protocols* presented in this chapter has been published at the following workshop.

> Recursive Ping-Pong Protocols by H. Hüttel and J. Srba. In Proceedings of 4th International Workshop on Issues in the Theory of Security (WITS'04), pages 129-140, 2004.

An extended version appeared as a technical report.

> Recursive Ping-Pong Protocols by H. Hüttel and J. Srba. Technical report RS-03-47, BRICS Research Series, 21 pages, 2004.

The technical report extends the workshop paper by including full proofs and the reduction in Theorem 5.3 together with the discussion about global/local knowledge functions and the respective constructions presented in the appendix.

Except for minor typographical changes the content of this chapter is equal to the technical report.

Recursive Ping-Pong Protocols

Hans Hüttel and Jiří Srba

Abstract. This paper introduces a process calculus with recursion which allows us to express an unbounded number of runs of the ping-pong protocols introduced by Dolev and Yao. We study the decidability issues associated with two common approaches to checking security properties, namely reachability analysis and bisimulation checking. Our main result is that our channel-free and memory-less calculus is Turing powerful, assuming that at least three principals are involved. We also investigate the expressive power of the calculus in the case of two participants. Here, our main results are that reachability and, under certain conditions, also strong bisimilarity become decidable.

5.1 Introduction

The study of correctness properties of cryptographic protocols has become an increasingly important research topic. Today, research on cryptographic protocols is often conducted using methods from program semantics together with the so-called Dolev-Yao assumptions about protocol principals and intruders introduced in [11]. In the Dolev-Yao model, all communications of a protocol may be visible to the hostile environment which is capable of interfering with the protocol by altering or blocking any message and by creating new messages. Moreover, these are the only kinds of attacks — an intruder cannot exploit weaknesses of the encryption algorithm itself (the 'perfect encryption hypothesis').

Process calculi have been suggested as a natural vehicle for reasoning about cryptographic protocols. In [1], Abadi and Gordon introduced the spi-calculus (a variant of the π-calculus) and described how properties such as secrecy and authenticity can be expressed via observational equivalence. Alternatively, security properties can be expressed and examined using reachability analysis [4, 6, 14]. An important question is: Given the Dolev-Yao assumptions, to which extent are the properties of cryptographic protocols decidable?

A number of security properties are decidable for the class of finite protocols [4, 16]. In the case of an unbounded number of protocol configurations, the picture is more complex. Durgin et al. showed in [12] that security properties are undecidable in a restricted class of so-called bounded protocols (that still allows for infinitely many reachable configurations). In [3] Amadio and Charatonik consider a language of tail-recursive protocols with bounded encryption depth and name generation; they show that, whenever certain restrictions on decryption are violated, one can encode two-counter machines in the process language. On the other hand, Amadio, Lugiez and Vanackère show in [5] that the reachability problem is in PTIME for a class of protocols with replication (as opposed to recursion).

Another contribution by Dolev and Yao in [11] is the study of *ping-pong protocols*. These are memory-less protocols which may be subjected to arbitrarily long attacks. Here, the secrecy of a finite ping-pong protocol can be decided in polynomial time. Later, Dolev, Even and Karp found a cubic-time algorithm [10] by expressing secrecy as emptiness of the intersection of a context-free language with a regular language. The class of protocols studied by Amadio et al. in [5] contains iterative ping-pong protocols and, as a consequence, secrecy properties remain polynomially decidable even in this case.

In this paper we examine decision problems for a process calculus capable of describing exactly the class of *recursive* ping-pong protocols. We study the calculus from the perspective of equivalence checking (for a general scheme see e.g. [15]) and consider no explicit model of the environment. The reason for considering this tiny calculus is that all negative results for it carry over to richer calculi capable of expressing a wider class of protocols (it is possible to describe an active intruder in the setting of bisimilarity/reachability checking of ping-pong protocols and this will be treated in our forthcoming paper).

The considered calculus is a channel-free process calculus, reminiscent of the tail-recursive processes studied by Amadio and Charatonik [3]. In the case of three principals, we can encode any Turing machine as a ping-pong protocol. Hence even this very restricted formalism is sufficiently expressive to encode universal computations. This implies that any richer calculus (and indeed any reasonable cryptographic calculus should subsume the ping-pong behaviour) is beyond the reach of automatic verification. The underlying idea of our construction is to encode the content of the tape as a series of encryptions and to express the transition function as a series of protocol steps. As a consequence, both reachability and bisimilarity are undecidable.

On the other hand, if the protocol is restricted to two principals, many properties become decidable. In particular, we show that the reachability problem is in PTIME, and under a certain natural observational condition also strong bisimilarity becomes decidable.

5.2 Basic Definitions

5.2.1 Transition Systems and Bisimilarity

We describe the semantics of our process calculi using unlabelled transition systems where every state has an associated set of its *knowledge*, which is an element of a given domain $\mathcal{D}$. Intuitively, knowledge represents observations visible to the environment.

A *transition system (with knowledge)* is a triple $(\mathsf{S}, \longrightarrow, kn)$ where S is a set of *states* (or *processes*), $\longrightarrow \subseteq \mathsf{S} \times \mathsf{S}$ is a *transition relation*, written $\alpha \longrightarrow \beta$, for $(\alpha, \beta) \in \longrightarrow$, and $\mathtt{kn} : \mathsf{S} \mapsto \mathcal{D}$ is a *knowledge function* from the set of states to the given knowledge domain $\mathcal{D}$.

Let $\mathcal{T} = (\mathsf{S}, \longrightarrow, kn)$ be a transition system. A binary relation $\mathsf{R} \subseteq \mathsf{S} \times \mathsf{S}$ is a *(strong) bisimulation* iff whenever $(\alpha, \beta) \in \mathsf{R}$ then $kn(\alpha) = kn(\beta)$ and if $\alpha \longrightarrow \alpha'$ then $\beta \longrightarrow \beta'$ for some β' such that $(\alpha', \beta') \in \mathsf{R}$, and if $\beta \longrightarrow \beta'$ then $\alpha \longrightarrow \alpha'$ for some α' such that $(\alpha', \beta') \in \mathsf{R}$.

Processes $\alpha_1, \alpha_2 \in \mathsf{S}$ are *(strongly) bisimilar* in $\mathcal{T}$, written $(\alpha_1, \mathcal{T}) \sim (\alpha_2, \mathcal{T})$ (or simply $\alpha_1 \sim \alpha_2$ if $\mathcal{T}$ is clear from the context), iff there is a (strong) bisimulation R such that $(\alpha_1, \alpha_2) \in \mathsf{R}$.

Given a pair of processes α_1 in a transition system $\mathcal{T}_1 = (\mathsf{S}_1, \longrightarrow_1, kn)$ and α_2 in $\mathcal{T}_2 = (\mathsf{S}_2, \longrightarrow_2, kn)$ such that $\mathsf{S}_1 \cap \mathsf{S}_2 = \emptyset$ and $kn : \mathsf{S}_1 \cup \mathsf{S}_2 \mapsto \mathcal{D}$, we write $(\alpha_1, \mathcal{T}_1) \sim (\alpha_2, \mathcal{T}_2)$ iff $(\alpha_1, \mathcal{T}) \sim (\alpha_2, \mathcal{T})$ such that $\mathcal{T} \stackrel{\text{def}}{=} (\mathsf{S}_1 \cup \mathsf{S}_2, \longrightarrow, kn)$ where $\alpha \longrightarrow \beta$ iff $\alpha, \beta \in \mathsf{S}_1$ and $\alpha \longrightarrow_1 \beta$, or $\alpha, \beta \in \mathsf{S}_2$ and $\alpha \longrightarrow_2 \beta$. Similarly if S_1 and S_2 are not disjoint, we simply rename the states of one of the transition systems and use the same notation $(\alpha_1, \mathcal{T}_1) \sim (\alpha_2, \mathcal{T}_2)$.

Bisimilarity has an elegant characterization in terms of *bisimulation games* [18,19]. A bisimulation game on a pair of processes $(\alpha_1, \mathcal{T})$ and $(\alpha_2, \mathcal{T})$ is a two-player game of an 'attacker' and a 'defender'. The game is played in rounds. In each round the attacker chooses one of the processes and performs a transition in the selected process; the defender must respond by performing a transition in the other process. Now the game repeats, starting from the new processes. If a pair of processes α_1 and α_2 such that $kn(\alpha_1) \neq kn(\alpha_2)$ is reached during the game, the attacker wins. If a player cannot perform a transition, the other player wins. If the game is infinite, the defender wins.

Processes $(\alpha_1, \mathcal{T})$ and $(\alpha_2, \mathcal{T})$ are bisimilar iff the defender has a winning strategy (and non-bisimilar iff the attacker has a winning strategy).

Two main decidability problems we shall investigate are (strong) bisimilarity checking and reachability analysis. The first problem asks the question (i) Are two given states α and β in a transition system (strongly) bisimilar ($\alpha \sim \beta$) ? and the second problem asks the question (ii) Is a given state β reachable from a state α, i.e., $\alpha \longrightarrow^* \beta$?

5.2.2 Ping-Pong Protocols

Let T be a set of *plain-text* messages and let K be a set of *encryption keys*. We let t range over T and k range over K. The set of messages over T encrypted with the keys from K is denoted by $\mathcal{M}(\mathsf{T}, \mathsf{K})$ and given by the following abstract syntax.

$$\mathsf{m} ::= \quad \mathsf{t} \quad | \quad \{\mathsf{m}\}_{\mathsf{k}}$$

Hence a message m (either a plain-text or an already encrypted message) can be encrypted with a key k, and such a message is written as $\{\mathsf{m}\}_{\mathsf{k}}$.

Let *Const* be a finite set of *process constants*. A *specification of a ping-pong protocol* is a finite set of *process definitions* Δ such that every process constant $\mathsf{P} \in \mathit{Const}$ has exactly one process definition of the form

$$\mathsf{P} \stackrel{\text{def}}{=} \sum_{\mathsf{i} \in \mathsf{I}} \nu_{\mathsf{i}}.\overline{w_{\mathsf{i}}}.\mathsf{P}_{\mathsf{i}}$$

where I is a finite set of indices, ν_{i} and w_{i} are messages over a fixed variable name x encrypted with the keys from K (i.e. $\nu_{\mathsf{i}}, w_{\mathsf{i}} \in \mathcal{M}(\{\mathsf{x}\}, \mathsf{K})$), and P_{i} is either a process constant or the *empty process* '$\mathbf{0}$' (i.e. $\mathsf{P}_{\mathsf{i}} \in \mathit{Const} \cup \{\mathbf{0}\}$). We call ν_{i} (resp. $\overline{w_{\mathsf{i}}}$) the *input* (resp. *output*) prefix.

Process definitions like these will often be written in their unfolded forms $\mathsf{P} \stackrel{\text{def}}{=} \nu_1.\overline{w_1}.\mathsf{P}_1 + \nu_2.\overline{w_2}.\mathsf{P}_2 + \cdots + \nu_n.\overline{w_n}.\mathsf{P}_n$ and whenever P_i is the empty process $\mathbf{0}$ then instead of $\nu_i.\overline{w_i}.\mathbf{0}$ we write only $\nu_i.\overline{w_i}$. Also, let $keys(\mathsf{u})$ denote the set of keys used in u for $\mathsf{u} \in \mathcal{M}(\mathsf{T},\mathsf{K})$, formally $keys(\mathsf{t}) \stackrel{\text{def}}{=} \emptyset$ and $keys(\{\mathsf{m}\}_\mathsf{k}) \stackrel{\text{def}}{=} \{\mathsf{k}\} \cup keys(\mathsf{m})$.

Example 5.1 *A simple protocol specification may look as follows:*

$$\Delta \stackrel{\text{def}}{=} \{\mathsf{P} \stackrel{\text{def}}{=} \{\mathsf{x}\}_\mathsf{k}.\overline{\{\{\mathsf{x}\}_{\mathsf{k}'}\}_\mathsf{k}}.\mathsf{P}\}$$

where $\mathsf{K} \stackrel{\text{def}}{=} \{\mathsf{k},\mathsf{k}'\}$. *The cyclic behaviour of the process constant* P *is described as follows:* P *receives a message and decrypts it by using the key* k*; the decrypted message is encrypted (using first the key* k' *and then* k*), and sent off.*

Let us finally also recall that for the prefixes $\{\mathsf{x}\}_\mathsf{k}$ *and* $\{\{\mathsf{x}\}_{\mathsf{k}'}\}_\mathsf{k}$ *we have* $keys(\{\mathsf{x}\}_\mathsf{k}) = \{\mathsf{k}\}$ *and* $keys(\{\{\mathsf{x}\}_{\mathsf{k}'}\}_\mathsf{k}) = \{\mathsf{k},\mathsf{k}'\}$.

A *configuration* of a ping-pong protocol specification Δ is a parallel composition of process constants, possibly preceded by output prefixes. Formally the set *Conf* of configurations is given by the following abstract syntax

$$\mathsf{C} ::= \quad \mathsf{P} \quad | \quad \overline{w}.\mathsf{P} \quad | \quad \mathsf{C}|\mathsf{C}$$

where $\mathsf{P} \in \mathcal{C}onst \cup \{\mathbf{0}\}$ ranges over process constants including the empty process, $w \in \mathcal{M}(\mathsf{T},\mathsf{K})$ ranges over the set of messages, and '|' is the operator of the parallel composition.

We introduce a structural congruence which identifies configurations that represent the same state of the protocol. $\equiv$ is defined as the least congruence over configurations ($\equiv \subseteq \mathcal{C}onf \times \mathcal{C}onf$) such that $(\mathcal{C}onf, |, \mathbf{0})$ is a commutative monoid. We identify configurations up to structural congruence.

The *width* of a configuration C is the minimal number of the parallel components in the $\equiv$-equivalence class represented by C. Hence e.g. the width of $\mathbf{0}$ is 0 and the width of $\mathsf{P}|\mathbf{0}|\mathsf{Q}$ is 2.

We shall now impose two natural restrictions on knowledge functions. We say that a knowledge function $kn : \mathcal{C}onf \mapsto \mathcal{D}$ for a given set $\mathcal{D}$ *respects structural equivalence* if $\mathsf{C}_1 \equiv \mathsf{C}_2$ implies $kn(\mathsf{C}_1) = kn(\mathsf{C}_2)$ for any $\mathsf{C}_1, \mathsf{C}_2 \in \mathcal{C}onf$. Let $\mathsf{C} \in \mathcal{C}onf$ be a configuration with the corresponding specification Δ. The set $prefix(\mathsf{C})$ of available prefixes of C is defined by: $prefix(\mathbf{0}) \stackrel{\text{def}}{=} \emptyset$; $prefix(\mathsf{C}) \stackrel{\text{def}}{=} \cup_{i\in I}\{\nu_i, \overline{w_i}\}$ if $\mathsf{C} \in \mathcal{C}onst$ such that $\left(\mathsf{C} \stackrel{\text{def}}{=} \sum_{i\in I} \nu_i.\overline{w_i}.\mathsf{P}_i\right) \in \Delta$; $prefix(\mathsf{C}) \stackrel{\text{def}}{=} \{\overline{w}\}$ if $\mathsf{C} = \overline{w}.\mathsf{P}$ for some $\mathsf{P} \in \mathcal{C}onst$; and $prefix(\mathsf{C}) \stackrel{\text{def}}{=} prefix(\mathsf{C}_1) \cup prefix(\mathsf{C}_2)$ if $\mathsf{C} = \mathsf{C}_1|\mathsf{C}_2$. The intuition is that the input/output capabilities of a configuration depend only on the available prefixes and not on the names of process constants. If we consistently rename the process constants in C and Δ and obtain a new configuration C' and a new specification Δ', it is the case that $prefix(\mathsf{C}) = prefix(\mathsf{C}')$. Hence we say that a knowledge function *kn respects renaming of process constants* if $prefix(\mathsf{C}_1) = prefix(\mathsf{C}_2)$ implies $kn(\mathsf{C}_1) = kn(\mathsf{C}_2)$ for any $\mathsf{C}_1, \mathsf{C}_2 \in \mathcal{C}onf$.

We only consider knowledge functions that respect structural congruence and renaming of process constants; we will call such functions *respecting*.

Example 5.2 *Define the following knowledge functions $kn_\emptyset$, kn_{priv} and kn_{plain}:*

- *The* empty knowledge function $kn_\emptyset : \mathcal{C}onf \mapsto \{\bot\}$ *is defined by* $kn_\emptyset(\mathsf{C}) \stackrel{\text{def}}{=} \bot$ *for all* $\mathsf{C} \in \mathcal{C}onf$.
- *The* private-keys knowledge function $kn_{priv} : \mathcal{C}onf \mapsto 2^{\mathsf{K}}$ *(where* K *is the set of keys) is defined by* $kn_{priv}(\mathsf{C}) \stackrel{\text{def}}{=} \bigcup_{\mathsf{i}\in\mathsf{I}} keys(\nu_\mathsf{i})$ *if* $\mathsf{C} \in \mathcal{C}onst$ *and* $\mathsf{C} \stackrel{\text{def}}{=} \sum_{\mathsf{i}\in\mathsf{I}} \nu_\mathsf{i}.\overline{w_\mathsf{i}}.\mathsf{P}_\mathsf{i}$, $kn_{priv}(\mathsf{C}) \stackrel{\text{def}}{=} kn_{priv}(\mathsf{C}_1) \cup kn_{priv}(\mathsf{C}_2)$ *if* $\mathsf{C} = \mathsf{C}_1|\mathsf{C}_2$ *and* $kn_{priv}(\mathsf{C}) \stackrel{\text{def}}{=} \emptyset$ *otherwise.*
- *The* plain-text knowledge function $kn_{plain} : \mathcal{C}onf \mapsto 2^{\mathsf{T}}$ *(where* T *is the set of plain-text messages), is defined by* $kn_{plain}(\mathsf{C}) \stackrel{\text{def}}{=} \{\mathsf{t}\}$ *if* $\mathsf{C} = \overline{\mathsf{t}}.\mathsf{P}$ *for some* $\mathsf{t} \in \mathsf{T}$ *and* $\mathsf{P} \in \mathcal{C}onst$, $kn_{plain}(\mathsf{C}) \stackrel{\text{def}}{=} kn_{plain}(\mathsf{C}_1) \cup kn_{plain}(\mathsf{C}_2)$ *if* $\mathsf{C} = \mathsf{C}_1|\mathsf{C}_2$ *and* $kn_{plain}(\mathsf{C}) \stackrel{\text{def}}{=} \emptyset$ *otherwise.*

It is easy to see that these knowledge functions are respecting. The intuition is that $kn_\emptyset$ does not take the knowledge of configurations into account and hence in bisimilarity checking only the branching structure induced by $\longrightarrow$ is relevant. The knowledge function kn_{priv} makes sure that any two bisimilar configurations have the same decryption keys currently available. Finally, the function kn_{plain} identifies configurations which are currently capable of communicating the same set of plain-text messages.

Example 5.3 *Consider a ping-pong protocol motivated by a simple example mentioned e.g. in [10] and [11]. A participant* X *wants to send a message* $\mathsf{m} \in \{0,1\}^*$ *to a participant* Y *and get a confirmation that the message was received. The protocol is informally described as follows: (i) participant* X *encrypts the message* m *by a public key of* Y *and sends the encrypted message to* Y*, (ii) participant* Y *decrypts the received message by his private key and answers to* X *by the same message* m *encrypted with* X*'s public key, (iii) finally* X *receives the confirmation message and decrypts it by his private key.*

In our formalism let $\mathsf{T} \stackrel{\text{def}}{=} \{0,1\}^*$, $\mathsf{K} \stackrel{\text{def}}{=} \{\mathsf{k}_\mathsf{X}, \mathsf{k}_\mathsf{Y}\}$, *and* $\mathcal{C}onst \stackrel{\text{def}}{=} \{\mathsf{X}, \mathsf{Y}\}$. *The protocol specification* Δ *is given by two equations*

$$\mathsf{X} \stackrel{\text{def}}{=} \{\mathsf{x}\}_{\mathsf{k}_\mathsf{X}}.\overline{\{\mathsf{x}\}_{\mathsf{k}_\mathsf{Y}}}.\mathsf{X} \qquad \mathsf{Y} \stackrel{\text{def}}{=} \{\mathsf{x}\}_{\mathsf{k}_\mathsf{Y}}.\overline{\{\mathsf{x}\}_{\mathsf{k}_\mathsf{X}}}$$

and the initial configuration of the protocol is $\overline{\{\mathsf{m}\}_{\mathsf{k}_\mathsf{Y}}}.\mathsf{X} \mid \mathsf{Y}$ *for a given* $\mathsf{m} \in \mathsf{T}$.

The intuition is that the process $\overline{\{\mathsf{m}\}_{\mathsf{k}_\mathsf{Y}}}.\mathsf{X}$ *can output the message* m *encrypted with the key* k_Y *and become the process* X*. Similarly,* Y *can input this message and become the process* $\overline{\{\mathsf{m}\}_{\mathsf{k}_\mathsf{X}}}.\mathsf{Y}$*. After the communication is established, the new configuration* $\mathsf{X} \mid \overline{\{\mathsf{m}\}_{\mathsf{k}_\mathsf{X}}}.\mathsf{Y}$ *is reached. The conformation phase of the protocol is analogous to the first communication.*

Note that we do not distinguish explicitly between private and public keys. This is done implicitly: a key supposed to be a private key for one or more process constants cannot be used in the input prefixes of other process constants.

$$\frac{(P \stackrel{\text{def}}{=} \sum_{i\in I} \nu_i.\overline{w_i}.P_i) \in \Delta \qquad u = \nu_i[m/x] \qquad m \in \mathcal{M}(T,K) \qquad i \in I}{P \mid \overline{u}.Q \longrightarrow \overline{w_i[m/x]}.P_i \mid Q}$$

$$\frac{P \longrightarrow P'}{P|Q \longrightarrow P'|Q}$$

Figure 5.1: SOS rules of ping-pong protocols

E.g. in our example k_X is a private key of X which means the process constant Y can use the key only in its output prefix and vice versa.

A formal semantics of ping-pong protocols is given in terms of transition systems. First, we define inductively a substitution $u[m'/x]$ of the message m' for the variable x in the prefix u ($m' \in \mathcal{M}(T,K)$ and $u \in \mathcal{M}(\{x\},K)$) by $x[m'/x] \stackrel{\text{def}}{=} m'$ and $\{m\}_k[m'/x] \stackrel{\text{def}}{=} \{m[m'/x]\}_k$.

A given protocol specification Δ determines a transition system $\mathcal{T}(\Delta) \stackrel{\text{def}}{=} (S, \longrightarrow, kn)$ where states are configurations of the protocol modulo the structural congruence ($S \stackrel{\text{def}}{=} \mathcal{C}onf/\equiv$); the transition relation $\longrightarrow$ is given by the SOS rules in Figure 5.1 (recall that '|' is commutative); and the knowledge function $kn : S \mapsto \mathcal{D}$ is assumed to be explicitly given. Note that the width of a configuration does not increase by performing a transition.

Example 5.4 *Let us consider the ping-pong protocol specification from Example 5.3. The interesting fragment of the transition system $\mathcal{T}(\Delta)$ is*

$$\overline{\{m\}_{k_Y}}.X \mid Y \longrightarrow X \mid \overline{\{m\}_{k_X}}.Y \longrightarrow \overline{\{m\}_{k_Y}}.X$$

In the rest of this section we will discuss the usefulness of bisimilarity checking with knowledge functions for validation of authenticity and secrecy of ping-pong protocols. The correctness checking of such protocols is done by comparing the protocol specification with its ideal behaviour [2] under an appropriate choice of the knowledge function.

For example, assume that we want to check whether a given protocol specification Δ ever outputs a plain-text message. Let us fix an arbitrary key $k \in K$. We define an ideal protocol Δ' which has the same behaviour as Δ but never communicates any plain-text message. This can be achieved e.g. by defining

$$\Delta' \stackrel{\text{def}}{=} \{P \stackrel{\text{def}}{=} \sum_{i\in I} \{\nu_i\}_k.\overline{\{w_i\}_k}.P_i \mid (P \stackrel{\text{def}}{=} \sum_{i\in I} \nu_i.\overline{w_i}.P_i) \in \Delta\}.$$

For any configuration $C \in \mathcal{C}onf$ let C' be a configuration where every occurrence of an output prefix of the form $\overline{w}.P$ is replaced with $\overline{\{w\}_k}.P$. Under the assumption that the plain-text knowledge function kn_{plain} is used, it holds that $(C, \mathcal{T}(\Delta)) \sim (C', \mathcal{T}(\Delta'))$ if and only if the protocol Δ starting from its initial configuration C is never capable of outputting any plain-text message.

Assume now that Δ contains input prefixes only of the from $\{x\}_k$ for some key $k \in K$ (only one key decryption at a time). The question whether a computation

from a given configuration C of the protocol specification Δ never deadlocks and it is always possible to decrypt messages encrypted with a fixed key k can be expressed as follows. Let $\Delta' \stackrel{\text{def}}{=} \{\mathsf{X} \stackrel{\text{def}}{=} \{\mathsf{x}\}_{\mathsf{k}}.\overline{\{\mathsf{x}\}_{\mathsf{k}}}.\mathsf{X}\}$ and let C' be the configuration $\overline{\{\mathsf{t}\}_{\mathsf{k}}}.\mathsf{X} \mid \mathsf{X}$ for some $\mathsf{t} \in \mathsf{T}$. Obviously, from C' there is exactly one transition leading back to C' and moreover C' is always capable of decrypting a message encrypted with the key k. We also define a knowledge function $kn : \mathcal{C}onf \mapsto \{0, 1\}$ by $kn(\mathsf{C}_1) \stackrel{\text{def}}{=} 1$ if $\mathsf{k} \in kn_{priv}(\mathsf{C}_1)$; and $kn(\mathsf{C}_1) \stackrel{\text{def}}{=} 0$ otherwise. Here kn_{priv} is the private-keys knowledge function introduced before. The considered validation question is now equivalent to the problem $(\mathsf{C}, \mathcal{T}(\Delta)) \sim (\mathsf{C}', \mathcal{T}(\Delta'))$.

Remark 5.1 *If instead of kn defined above we use $kn_\emptyset$, the bisimilarity question $(\mathsf{C}, \mathcal{T}(\Delta)) \sim (\mathsf{C}', \mathcal{T}(\Delta'))$ is equivalent to the problem whether there is no terminating computation of the protocol Δ starting in C.*

5.3 Ping-Pong Protocols of Width 3

In this section we show that ping-pong protocols of width at least 3 are surprisingly powerful enough to simulate Turing machines.

Let $\mathsf{M} = (\mathsf{Q}, \Sigma, \gamma, \mathsf{q}_0, \mathsf{q}_\mathsf{F})$ be a *Turing machine* such that Q is a finite set of *control states*, Σ is a finite *tape alphabet* containing special symbols $¢, \$ \in \Sigma$ ($¢$ is the left mark of the tape and \$ it the right mark; let $\Sigma_1 \stackrel{\text{def}}{=} \Sigma \setminus \{¢, \$\}$), $\mathsf{q}_0 \in \mathsf{Q}$ is the *initial state*, q_F is the *final state* and

$$\gamma : \Sigma \times \mathsf{Q} \times \Sigma \mapsto (\mathsf{Q} \times \Sigma \times \Sigma \ \cup \ \Sigma \times \Sigma \times \mathsf{Q} \ \cup \ \{halt\})$$

is a total function such that

- $\gamma(\mathsf{a}\mathsf{q}\mathsf{b}) \in \big(\mathsf{Q} \times \{\mathsf{a}\} \times \Sigma_1 \ \cup \ \{\mathsf{a}\} \times \Sigma_1 \times \mathsf{Q}\big)$ for all $\mathsf{a}, \mathsf{b} \in \Sigma_1$ and $\mathsf{q} \in \mathsf{Q}$
  (the head moves either to the left or to the right)
- $\gamma(¢\mathsf{q}\mathsf{a}) \in \{¢\} \times \Sigma_1 \times \mathsf{Q}$ for all $\mathsf{a} \in \Sigma$ and $\mathsf{q} \in \mathsf{Q}$
  (the head is not allowed to move on the left mark)
- $\gamma(\mathsf{a}\mathsf{q}\$) \in \big(\mathsf{Q} \times \{\mathsf{a}\} \times \{\$\} \ \cup \ \{\mathsf{a}\} \times \Sigma_1 \times \mathsf{Q}\big)$ for all $\mathsf{a} \in \Sigma$ and $\mathsf{q} \in \mathsf{Q}$
  (the right mark cannot be changed but the head can move to the right)
- $\gamma(\mathsf{a}\mathsf{q}_\mathsf{F}\mathsf{b}) = halt$ for all $\mathsf{a}, \mathsf{b} \in \Sigma$
  (when the final state q_F is reached, the computation stops).

A *configuration* of the machine M is an element from the set $\{¢\} \times \Sigma_1^* \times \mathsf{Q} \times \Sigma_1^* \times \{\$\}$. A *computational step* between configurations c_1 and c_2 (written $\mathsf{c}_1 \longrightarrow \mathsf{c}_2$) is defined in the usual way, i.e.,

- $\mathsf{c}_1 \longrightarrow \mathsf{c}_2$ if $\mathsf{c}_1 \equiv w_1\mathsf{a}\mathsf{q}\mathsf{b}w_2$ and $\mathsf{c}_2 \equiv w_1\gamma(\mathsf{a}\mathsf{q}\mathsf{b})w_2$ where $w_1, w_2 \in \Sigma^*$, $\mathsf{a}, \mathsf{b} \in \Sigma$ and $\mathsf{q} \in \mathsf{Q}$ such that $\mathsf{b} \neq \$$, or $\big(\mathsf{b} = \$ \ \wedge \ \gamma(\mathsf{a}\mathsf{q}\mathsf{b}) \in \mathsf{Q} \times \{\mathsf{a}\} \times \{\$\}\big)$
  (the head does not write on the right mark), or
- $\mathsf{c}_1 \longrightarrow \mathsf{c}_2$ if $\mathsf{c}_1 \equiv w_1\mathsf{a}\mathsf{q}\$$ and $\mathsf{c}_2 \equiv w_1\gamma(\mathsf{a}\mathsf{q}\$)\$$ where $w_1 \in \Sigma^*$, $\mathsf{a} \in \Sigma$ and $\mathsf{q} \in \mathsf{Q}$ such that $\gamma(\mathsf{a}\mathsf{q}\$) \in \{\mathsf{a}\} \times \Sigma_1 \times \mathsf{Q}$ (the head is at the end mark and moves to the right).

It is a well known fact that the problem whether the machine *halts* from the initial configuration $\mathfrak{q}\mathsf{q}_0\$$ (i.e. reaches a configuration containing the state q_F in a finite number of computational steps) is undecidable.

Let $\mathsf{M} = (\mathsf{Q}, \Sigma, \gamma, \mathsf{q}_0, \mathsf{q}_\mathsf{F})$ be a Turing machine and let $\mathsf{S} \stackrel{\text{def}}{=} \mathsf{Q} \cup \Sigma \cup \{\mathsf{z}\}$ and $\mathsf{S}' \stackrel{\text{def}}{=} \{\mathsf{s}' \mid \mathsf{s} \in \mathsf{S}\}$ such that z is a fresh symbol and $\mathsf{S} \cap \mathsf{S}' = \emptyset$. We define a ping-pong protocol Δ that will simulate the computation of the machine M. The set of plain-text messages is a singleton set $\mathsf{T} \stackrel{\text{def}}{=} \{\mathsf{t}\}$, the set of keys is $\mathsf{K} \stackrel{\text{def}}{=} \mathsf{S} \cup \mathsf{S}'$, and the set of process constants consists of $\mathcal{C}onst \stackrel{\text{def}}{=} \{\mathsf{B}_\mathsf{S}, \mathsf{B}_{\mathsf{S}'}, \mathsf{P}, \mathsf{R}\} \cup \{\mathsf{P}_{\mathfrak{a}'}, \mathsf{R}_\mathfrak{a} \mid \mathfrak{a} \in \mathsf{S}\} \cup \{\mathsf{V}_{\mathfrak{a}'\mathfrak{b}'\mathfrak{c}'}, \mathsf{V}_{\mathfrak{a}'\mathfrak{b}'\mathfrak{c}'\$'} \mid \mathfrak{a}, \mathfrak{b}, \mathfrak{c} \in \mathsf{S}\}$.

First, we define the equations for process constants B_S and $\mathsf{B}_{\mathsf{S}'}$ (buffers over S and S').

$$\mathsf{B}_\mathsf{S} \stackrel{\text{def}}{=} \sum_{\mathsf{s}\in\mathsf{S}} \{\mathsf{x}\}_\mathsf{s}.\overline{\{\mathsf{x}\}_\mathsf{s}}.\mathsf{B}_\mathsf{S} \qquad \mathsf{B}_{\mathsf{S}'} \stackrel{\text{def}}{=} \sum_{\mathsf{s}'\in\mathsf{S}'} \{\mathsf{x}\}_{\mathsf{s}'}.\overline{\{\mathsf{x}\}_{\mathsf{s}'}}.\mathsf{B}_{\mathsf{S}'}$$

The intuition is that the buffers B_S and B'_S can store their content as a sequence of encryption keys over S resp. S'. In our case the buffers will store configurations of the machine M.

The process constant P transfers the content of the buffer B_S to the buffer $\mathsf{B}_{\mathsf{S}'}$ and as soon as a control state is present, the computational step is performed. This is formally defined by

$$\mathsf{P} \stackrel{\text{def}}{=} \sum_{\mathfrak{a},\mathfrak{b},\mathfrak{c}\in\mathsf{S},\ \mathfrak{b}\notin\mathsf{Q}} \{\{\{\mathsf{x}\}_\mathfrak{c}\}_\mathfrak{b}\}_\mathfrak{a}.\overline{\{\{\mathsf{x}\}_\mathfrak{c}\}_\mathfrak{b}}.\mathsf{P}_{\mathfrak{a}'} +$$

$$\sum_{\mathfrak{a},\mathfrak{c}\in\Sigma,\ \mathfrak{q}\in\mathsf{Q},\ \gamma(\mathfrak{a}\mathfrak{q}\mathfrak{c})=\mathfrak{e}\mathfrak{f}\mathfrak{g},\ \neg\omega} \{\{\{\mathsf{x}\}_\mathfrak{c}\}_\mathfrak{q}\}_\mathfrak{a}.\overline{\mathsf{x}}.\mathsf{V}_{\mathfrak{e}'\mathfrak{f}'\mathfrak{g}'} +$$

$$\sum_{\mathfrak{a},\mathfrak{c}\in\Sigma,\ \mathfrak{q}\in\mathsf{Q},\ \gamma(\mathfrak{a}\mathfrak{q}\mathfrak{c})=\mathfrak{e}\mathfrak{f}\mathfrak{g},\ \omega} \{\{\{\mathsf{x}\}_\mathfrak{c}\}_\mathfrak{q}\}_\mathfrak{a}.\overline{\mathsf{x}}.\mathsf{V}_{\mathfrak{e}'\mathfrak{f}'\mathfrak{g}'\$'}$$

where ω is the condition saying that the head is at the end of the tape and it moves to the right, i.e., $\omega \equiv \mathfrak{c} = \$ \wedge \mathfrak{g} \in \mathsf{Q}$.

The process constant $\mathsf{P}_{\mathfrak{a}'}$ for all $\mathfrak{a}' \in \mathsf{S}'$ simply adds the symbol $\mathfrak{a}'$ to the buffer $\mathsf{B}_{\mathsf{S}'}$ and then it continues as P.

$$\mathsf{P}_{\mathfrak{a}'} \stackrel{\text{def}}{=} \sum_{\mathsf{s}'\in\mathsf{S}'} \{\mathsf{x}\}_{\mathsf{s}'}.\overline{\{\{\mathsf{x}\}_{\mathsf{s}'}\}_{\mathfrak{a}'}}.\mathsf{P}$$

The process constants $\mathsf{V}_{\mathfrak{a}'\mathfrak{b}'\mathfrak{c}'}$ and $\mathsf{V}_{\mathfrak{a}'\mathfrak{b}'\mathfrak{c}'\$'}$ (for $\mathfrak{a}', \mathfrak{b}', \mathfrak{c}' \in \mathsf{S}'$) add the corresponding sequence of keys to the buffer $\mathsf{B}_{\mathsf{S}'}$ and then continue as R.

$$\mathsf{V}_{\mathfrak{a}'\mathfrak{b}'\mathfrak{c}'} \stackrel{\text{def}}{=} \sum_{\mathsf{s}'\in\mathsf{S}'} \{\mathsf{x}\}_{\mathsf{s}'}.\overline{\{\{\{\{\mathsf{x}\}_{\mathsf{s}'}\}_{\mathfrak{a}'}\}_{\mathfrak{b}'}\}_{\mathfrak{c}'}}.\mathsf{R}$$

$$\mathsf{V}_{\mathfrak{a}'\mathfrak{b}'\mathfrak{c}'\$'} \stackrel{\text{def}}{=} \sum_{\mathsf{s}'\in\mathsf{S}'} \{\mathsf{x}\}_{\mathsf{s}'}.\overline{\{\{\{\{\{\mathsf{x}\}_{\mathsf{s}'}\}_{\mathfrak{a}'}\}_{\mathfrak{b}'}\}_{\mathfrak{c}'}\}_{\$'}}.\mathsf{R}$$

We finish the definition of the process constants by introducing R (standing for 'reverse'), which transfers the content of the buffer $B_{S'}$ back to B_S.

$$R \stackrel{\text{def}}{=} \sum_{a' \in S'} \{x\}_{a'}.\overline{x}.R_a$$

Similarly as $P_{a'}$, R_a for all $a \in S \setminus \{¢\}$ adds the symbol a to the buffer B_S and continues as R, except for the situation when the beginning of the tape was reached (in this case $R_¢$ continues as P).

$$R_a \stackrel{\text{def}}{=} \sum_{s \in S} \{x\}_s.\overline{\{\{x\}_s\}_a}.R \qquad R_¢ \stackrel{\text{def}}{=} \sum_{s \in S} \{x\}_s.\overline{\{\{x\}_s\}_¢}.P$$

Let $m \stackrel{\text{def}}{=} \{t\}_z$ and $m' \stackrel{\text{def}}{=} \{t\}_{z'}$ (recall that $t \in T$ is the only plain-text message). The following configuration of width 3 can simulate the computation of the machine M from the initial configuration $¢q_0\$$.

$$\overline{\{\{\{m\}_\$\}_{q_0}\}_¢}.B_S \mid \overline{m'}.B_{S'} \mid P$$

In order to see how the simulation works we use the notations $[w]_m$ and $[w]_{m'}$ to denote the messages m and m' encrypted with the sequence of keys w, i.e., $[\epsilon]_m \stackrel{\text{def}}{=} m$, $[\epsilon]_{m'} \stackrel{\text{def}}{=} m'$, $[aw]_m \stackrel{\text{def}}{=} \{[w]_m\}_a$ and $[aw]_{m'} \stackrel{\text{def}}{=} \{[w]_{m'}\}_a$ where ϵ is the empty sequence, $a \in K$ and $w \in K^*$.

Now every configuration c of the machine M corresponds to the configuration $f(c) \stackrel{\text{def}}{=} (\overline{[c]_m}.B_S \mid \overline{[\epsilon]_{m'}}.B_{S'} \mid P)$ of the protocol Δ. Note that the initial configuration of Δ defined above is exactly $f(¢q_0\$)$.

The following considerations will describe the simulation of the Turing machine M by the protocol Δ. A single step of the machine M will be simulated by a finite number of transitions in the protocol.

Let $c_1 = w_1aqcw_2$ and $c_2 = w_1efgw_2$ be configurations of M such that $w_1, w_2 \in \Sigma^*$, $a, c \in \Sigma$, $q \in Q$, $\gamma(aqc) = efg$, and $\neg\omega$. This means that $c_1 \longrightarrow c_2$. We will show that $f(c_1) \longrightarrow^* f(c_2)$ such that the computation of the protocol from $f(c_1)$ is deterministic, i.e., for any $C \in \mathcal{C}onf$ such that $f(c_1) \longrightarrow^* C$ it is the case that $C \longrightarrow C'$ and $C \longrightarrow C''$ implies $C' \equiv C''$. For w_1 in the form $a_1 \cdots a_n$ let $w'_{1,R} \stackrel{\text{def}}{=} a'_n \cdots a'_1$ be the reversed word w_1 such that every letter is primed. The computation from $f(c_1)$ looks as follows.

$$f(c_1) = \overline{[w_1aqcw_2]_m}.B_S \mid \overline{[\epsilon]_{m'}}.B_{S'} \mid P \longrightarrow^* \overline{[aqcw_2]_m}.B_S \mid \overline{[w'_{1,R}]_{m'}}.B_{S'} \mid P \longrightarrow$$

$$B_S \mid \overline{[w'_{1,R}]_{m'}}.B_{S'} \mid \overline{[w_2]_m}.V_{e'f'g'} \longrightarrow \overline{[w_2]_m}.B_S \mid \overline{[w'_{1,R}]_{m'}}.B_{S'} \mid V_{e'f'g'} \longrightarrow$$

$$\overline{[w_2]_m}.B_S \mid B_{S'} \mid \overline{[g'f'e'w'_{1,R}]_{m'}}.R \longrightarrow \overline{[w_2]_m}.B_S \mid \overline{[g'f'e'w'_{1,R}]_{m'}}.B_{S'} \mid R \longrightarrow^*$$

$$\overline{[w_1efgw_2]_m}.B_S \mid \overline{[\epsilon]_{m'}}.B_{S'} \mid P = f(c_2)$$

It is easy to observe that such a computation is unique (deterministic).

Let $c_1 = w_1aq\$$ and $c_2 = w_1efg\$$ be configurations of M as before, however, this time the condition ω holds. This case is analogous to the previous one. The only difference is that $V_{e'f'g'}$ is replaced with $V_{e'f'g'\$'}$ and the end of the tape

\$ is added (the tape hence becomes longer by one cell). Assume now that $\mathsf{f}(\mathsf{c}_1)$ represents a halting configuration. The computation of the protocol from $\mathsf{f}(\mathsf{c}_1)$ starts as before, however, there is no summand in the definition of the process constant P for the situation that $\gamma(\mathtt{aqc}) = \mathit{halt}$ and hence the computation gets stuck in the configuration $\overline{[\mathtt{aqc}w_2]_\mathsf{m}}.\mathsf{B}_\mathsf{S} \mid \overline{[w'_{1,\mathsf{R}}]_{\mathsf{m}'}}.\mathsf{B}_{\mathsf{S}'} \mid \mathsf{P}$.

The following theorems are applications of the presented simulation.

Theorem 5.1 *Reachability is undecidable for ping-pong protocols of width* 3*.*

Theorem 5.2 *Bisimilarity is undecidable for ping-pong protocols of width* 3 *for any knowledge function which respects structural congruence and renaming of process constants.*

5.4 Ping-Pong Protocols of Width 2

If the family of protocols we consider contains at most two participants (parallel components), we get a class similar to that of the traditional ping-pong protocols [10, 11]. In fact, our class is more general in the sense that we allow for recursive definitions in the protocol specification. In the situation of at most two parallel components in any reachable configuration we may without loss of generality assume that such configurations are always of the form $\mathsf{P} \mid \overline{w}.\mathsf{Q}$ for some $\mathsf{P} \in \mathcal{C}\mathit{onst}$, $\mathsf{Q} \in \mathcal{C}\mathit{onst} \cup \{\mathbf{0}\}$ and $w \in \mathcal{M}(\mathsf{T}, \mathsf{K})$. If this is not the case, the computation of such a protocol is stuck — no communication can take place.

We shall now observe that the class of ping-pong protocols of width 2 is not Turing powerful since e.g. reachability becomes decidable. Hence we can still hope for automatic verification of some protocol properties.

Theorem 5.3 *The reachability problem for ping-pong protocols of width* 2 *is decidable in polynomial time.*

In contrast, the bisimilarity problem is again undecidable.

Theorem 5.4 *The problem of bisimilarity checking between a pair of ping-pong protocol configurations of width* 2 *is undecidable (provided that we allow for general but still computable, respecting and finite-domain knowledge functions).*

In most of the examples of knowledge functions we, however, do not use knowledge functions similar to the one described in the undecidability proof. In fact, it is quite satisfactory to consider knowledge functions which depend only on a constant number of the upper-most symbols in the output prefix. We shall prove that if this is the case then bisimilarity becomes decidable.

Let $\mathsf{P} \mid \overline{w}.\mathsf{Q}$ be a general form of a protocol configuration of width 2. A knowledge function kn is *local* if there is a constant $\mathsf{M} \in \mathbb{N}^0$ and a function

$$\mathsf{f} : (\mathcal{C}\mathit{onst} \cup \{\mathbf{0}\}) \times (\mathcal{C}\mathit{onst} \cup \{\mathbf{0}\}) \times \left((\mathsf{K}^{<\mathsf{M}} \times \mathsf{T}) \cup \mathsf{K}^{\mathsf{M}}\right) \mapsto \mathcal{D}$$

such that $kn(\mathsf{P} \mid \overline{w}.\mathsf{Q}) = \mathsf{f}(\mathsf{P}, \mathsf{Q}, w')$ where w' is $\langle w \rangle$ if $|\langle w \rangle| \leq \mathsf{M}$, and w' is the prefix of $\langle w \rangle$ of length M otherwise. (Here by $\langle \mathsf{m} \rangle \in (\mathsf{K} \cup \mathsf{T})^*$ we understand

the sequence of keys that occur in m followed by the corresponding plain-text message, i.e., $\langle \mathsf{t} \rangle = \mathsf{t}$ for $\mathsf{t} \in \mathsf{T}$ and $\langle \{\mathsf{m}\}_{\mathsf{k}} \rangle = \mathsf{k}\langle \mathsf{m} \rangle$.) In other words, a local knowledge function depends only on P, Q and at most M outer-most keys of w.

Remark 5.2 *Observe that local knowledge functions have finite co-domains. Hence for every local knowledge function there is an equivalent local knowledge function with a finite domain $\mathcal{D}$. In what follows we shall assume that $\mathcal{D}$ is finite.*

Theorem 5.5 *The problem of bisimilarity checking between a pair of ping-pong protocol configurations of width 2 is decidable for local knowledge functions.*

5.5 Conclusion

We have studied a simple, channel-free process calculus capable of describing recursive ping-pong protocols. Surprisingly, the calculus turns out to be Turing-powerful when we allow three or more principals. This implies that all interesting verification problems will remain undecidable in any richer calculus which can express at least the ping-pong behaviour. This fact remains valid even if we allow active attacks on the protocol since the syntax of ping-pong protocols is capable of describing this situation. This is, however, nontrivial to see and it is a part of our current work.

In the case of two principals, the reachability problem is in PTIME. Depending on our notion of observability, other properties (including strong bisimilarity) may also become decidable.

Amadio et al. in [5] have proved that reachability is polynomially decidable for processes with replication. However, they have only shown the result for a class of processes that, unlike the class studied in the present paper, does not involve an explicit representation of nondeterministic choice. It remains to be seen whether security properties are decidable for a version of our process calculus with replication replacing recursion, and whether our calculus without explicit nondeterminism remains Turing powerful. We claim that at least the latter is indeed the case and in our future work we shall further investigate the problem (including the connection with the results from [9] where it is shown that secrecy is decidable for protocols with replication but without nondeterminism).

Appendix

Theorem 5.1 *Reachability is undecidable for ping-pong protocols of width* 3.

Proof. Let M be a Turing machine and let Δ be the ping-pong protocol constructed above. We modify Δ by adding the following summand to the definition of the process constant P

$$\sum_{a,c\in\Sigma,\ q\in Q,\ \gamma(aqc)=halt} \{\{\{x\}_c\}_q\}_a.\overline{x}.D$$

where D is a new process constant with its definition

$$D \stackrel{\text{def}}{=} \sum_{a\in S\cup S'} \{x\}_a.\overline{x}.D.$$

This means that M halts if and only if a protocol configuration containing D is reachable. The process constant D can remove the content of both of the buffers and hence the question whether the configuration $\overline{m}.B_S \mid \overline{m'}.B_{S'} \mid D$ is reachable from the initial configuration $f(\text{¢}q_0\$)$ is undecidable. □

Theorem 5.2 *Bisimilarity is undecidable for ping-pong protocols of width* 3 *for any knowledge function which respects structural congruence and renaming of process constants.*

Proof. Let M be a Turing machine and let Δ be the ping-pong protocol constructed above. Let Δ' be a copy of Δ such that every process constant $X \in \mathcal{C}onst$ in Δ' is replaced with X'. Moreover, Δ also contains the following extra summand in the definition of the process constant P

$$\sum_{a,c\in\Sigma,\ q\in Q,\ \gamma(aqc)=halt} \{\{\{x\}_c\}_q\}_a.\overline{\{\{\{x\}_c\}_q\}_a}$$

and Δ' contains the following extra summand in the definition of the process constant P' plus a definition of a fresh process constant Z'

$$\sum_{a,c\in\Sigma,\ q\in Q,\ \gamma(aqc)=halt} \{\{\{x\}_c\}_q\}_a.\overline{\{\{\{x\}_c\}_q\}_a}.Z'$$

$$Z' \stackrel{\text{def}}{=} \sum_{a,c\in\Sigma,\ q\in Q,\ \gamma(aqc)=halt} \{\{\{x\}_c\}_q\}_a.\overline{\{\{\{x\}_c\}_q\}_a}.Z'.$$

If the machine M diverges then the initial configurations (as described above) of Δ and Δ' are bisimilar since both of them are capable of performing infinite computations (these computations are deterministic) and the knowledge function cannot distinguish between them (it respects renaming of process constants).

If the machine M halts then Δ' can still perform an infinite sequence of transitions but Δ gets stuck after using the extra summand in P defined above. Hence their initial configurations cannot be bisimilar for any knowledge function. □

Theorem 5.3 *The reachability problem for ping-pong protocols of width 2 is decidable in polynomial time.*

Proof. We reduce reachability of ping-pong protocols of width 2 to reachability of pushdown automata (PDA), disregarding the input alphabet. In fact, we will use a slightly more general notion of PDA where several stack symbols can be removed in one computational step.

Let Δ be a given protocol specification and let $P_1 \mid \overline{w_1}.Q_1$ and $P_2 \mid \overline{w_2}.Q_2$ be two configurations of the protocol. We shall construct a PDA system together with two configurations $p_1\alpha_1$ and $p_2\alpha_2$ such that $P_1 \mid \overline{w_1}.Q_1 \longrightarrow^* P_2 \mid \overline{w_2}.Q_2$ if and only if $p_1\alpha_1 \longrightarrow^* p_2\alpha_2$.

Let $m \in \mathcal{M}(T, K)$. By $\langle m \rangle \in (K \cup T)^*$ we understand the sequence of keys that occur in m followed by the corresponding plain-text message, i.e., $\langle t \rangle = t$ for $t \in T$ and $\langle \{m\}_k \rangle = k\langle m \rangle$. Similarly, if $m \in \mathcal{M}(\{x\}, K)$ then $\langle m \rangle \in K^*$ denotes the sequence of keys that occur in m, i.e., $\langle x \rangle = \epsilon$ and $\langle \{m\}_k \rangle = k\langle m \rangle$.

The set of control states of the PDA automaton is $\{(P, Q) \mid P, Q \in \mathcal{C}onst \cup \{\mathbf{0}\}\}$ and the stack alphabet contains the encryption keys plus plain-text messages, i.e., it is the set $K \cup T$. The set of (input) actions is a singleton set $\{a\}$. We have now a natural correspondence between configurations of the protocol and those of the PDA system. A protocol configuration $P \mid \overline{w}.Q$ corresponds to a PDA configuration $(P, Q)\langle w \rangle$ such that (P, Q) is the control state (its second component is always the one that has an output prefix) and $\langle w \rangle$ is the stack content. The PDA rewrite rules are defined as follows: $(P, Q)\langle v_i \rangle \xrightarrow{a} (Q, P_i)\langle w_i \rangle$ for every $P \in \mathcal{C}onst$ and $Q \in \mathcal{C}onst \cup \{\mathbf{0}\}$ such that $P \stackrel{\text{def}}{=} \sum_{i \in I} v_i.\overline{w_i}.P_i$.

It is now easy to see that $P_1 \mid \overline{w_1}.Q_1 \longrightarrow^* P_2 \mid \overline{w_2}.Q_2$ if and only if $(P_1, Q_1)\langle w_1 \rangle \longrightarrow^* (P_2, Q_2)\langle w_2 \rangle$.

The reachability problem of extended PDA is by standard techniques reducible to reachability of ordinary PDA where at most one stack symbol is removed by performing a single transition (it is enough to replace every rule of the form $px_1x_2 \ldots x_m \longrightarrow q\alpha$ by the rules $px_1 \longrightarrow p_1$, $p_1x_2 \longrightarrow p_2$, $\ldots$, $p_{m-1}x_m \longrightarrow q\alpha$ where $p_1, \ldots, p_{m-1}$ are new control states).

Since reachability of PDA is decidable [8], we can conclude that reachability of ping-pong protocols of width 2 is also decidable. Moreover, the reachability problem of ordinary PDA can be solved in polynomial time [7,13], which implies that reachability of ping-pong protocols of width 2 is decidable also in PTIME. □

Definition 5.1 *A* Minsky machine R *with two counters* c_1 *and* c_2 *is a finite sequence of* instructions

$$R = (I_1, \quad I_2, \quad \ldots, \quad I_{n-1}, \quad n : \texttt{halt})$$

where $n \geq 1$ *and every* I_p, $1 \leq p \leq n-1$ *is an instruction of one from the following two types:*

- *increment:* $\quad p : \ c_i := c_i + 1; \ \texttt{goto}\ q$

- *test and decrement:* $\mathtt{p}$: `if` $\mathtt{c_i} = 0$ `then goto` $\mathtt{q}$ `else` $\mathtt{c_i} := \mathtt{c_i} - 1$; `goto` $\mathtt{r}$

where $1 \leq i \leq 2$ *and* $1 \leq q, r \leq n$.

Definition 5.2 *A* configuration *of a Minsky machine* R *is a triple* (p, ν_1, ν_2) *where* p *is an instruction label (*$1 \leq p \leq n$*), and* $\nu_1, \nu_2 \in \mathbb{N}^0$ *are nonnegative integers representing the values of the counters* c_1 *and* c_2*, respectively. The transition relation* $\longrightarrow$ *between configurations is defined in the natural way.*

Note that the computation of R is deterministic, i.e., if $(p, \nu_1, \nu_2) \longrightarrow (p', \nu_1', \nu_2')$ and $(p, \nu_1, \nu_2) \longrightarrow (p'', \nu_1'', \nu_2'')$ the $p' = p''$, $\nu_1' = \nu_1''$ and $\nu_2' = \nu_2''$.

Definition 5.3 *A Minsky machine* R halts *with the initial counter values set to zero if* $(1, 0, 0) \longrightarrow^* (n, \nu_1, \nu_2)$ *for some* $\nu_1, \nu_2 \in \mathbb{N}^0$*. If* R *does not halt we say that it* diverges*.*

Proposition 5.1 *The halting problem for Minsky machines in undecidable.*

Let R be a given Minsky machine. We now construct a ping-pong protocol specification Δ and two configurations C_1 and C_2 of width 2 such that R diverges if and only if $(C_1, \mathcal{T}(\Delta)) \sim (C_2, \mathcal{T}(\Delta))$.

Let the set of plain-text messages be $T \stackrel{\text{def}}{=} \{t\}$, let the set of encryption keys be $K \stackrel{\text{def}}{=} \{1, \ldots, n\} \cup \{+_i, -_i \mid 1 \leq i \leq 2\} \cup \{p^i_{=0}, p^i_{\geq 0}, p^{\mathtt{cheat}} \mid 1 \leq p < n, 1 \leq i \leq 2\}$ and let $\mathcal{C}onst \stackrel{\text{def}}{=} \{B_K, P, Q\}$. The constant B_K stands for a buffer over a certain subset of K.

Let $\#_k(m)$ denote the number of occurrences of the key $k \in K$ in the message $m \in \mathcal{M}(T, K)$. The intuition of the reduction is that a configuration (p, ν_1, ν_2) of the Minsky machine R corresponds to a pair of protocol configurations $\overline{\{m\}_p}.B_K \mid P$ and $\overline{\{m\}_p}.B_K \mid Q$ such that $\#_{+_i}(m) - \#_{-_i}(m) = \nu_i$ for $1 \leq i \leq 2$. The definitions of the process constants P and Q are almost symmetric except for the situation when a halting configuration of the machine R is reachable. In this case the computation from the configuration containing Q is stuck while the configuration containing P performs an infinite sequence of transition. The knowledge function is designed in such a way that if a correct computation of the machine R is simulated by the attacker in the bisimulation game (a single step in the computation of R will be simulated by two transitions in Δ), the defender can only mimic the same transitions in the other process. However, if the attacker "cheats" in the bisimulation game (e.g. decreases a value of a counter into negative integers), the defender threatens by entering a syntactically equal (and hence bisimilar) pair of protocol configurations.

Formally, the protocol Δ is given as follows.

$$B_K \stackrel{\text{def}}{=} \sum_{k \in \{1,\ldots,n\}} \{x\}_k.\overline{\{x\}_k}.B_K +$$

$$\sum_{All(p)} \Big(\{x\}_{p^i_{=0}}.\overline{\{x\}_p}.B_K \; + \; \{x\}_{p^i_{\geq 0}}.\overline{\{x\}_p}.B_K +$$

$$\{x\}_{p^{cheat}}.\overline{\{x\}_p}.B_K\Big)$$

$$P \stackrel{\text{def}}{=} \sum_{Inc(p)} \{x\}_p.\overline{\{\{x\}_{+_i}\}_q}.P +$$
$$\sum_{Dec(p)} \Big(\{x\}_p.\overline{\{x\}_{q^i_{=0}}}.P \; + \; \{x\}_p.\overline{\{x\}_{q^{cheat}}}.P +$$
$$\{x\}_p.\overline{\{x\}_{q^{cheat}}}.Q\Big) +$$
$$\sum_{Dec(p)} \Big(\{x\}_p.\overline{\{\{x\}_{-_i}\}_{r^i_{\geq 0}}}.P \; + \; \{x\}_p.\overline{\{\{x\}_{-_i}\}_{r^{cheat}}}.P +$$
$$\{x\}_p.\overline{\{\{x\}_{-_i}\}_{r^{cheat}}}.Q\Big) \; + \; \{x\}_n.\overline{\{x\}_n}.P$$

$$Q \stackrel{\text{def}}{=} \sum_{Inc(p)} \{x\}_p.\overline{\{\{x\}_{+_i}\}_q}.Q +$$
$$\sum_{Dec(p)} \Big(\{x\}_p.\overline{\{x\}_{q^i_{=0}}}.Q \; + \; \{x\}_p.\overline{\{x\}_{q^{cheat}}}.Q +$$
$$\{x\}_p.\overline{\{x\}_{q^{cheat}}}.P\Big) +$$
$$\sum_{Dec(p)} \Big(\{x\}_p.\overline{\{\{x\}_{-_i}\}_{r^i_{\geq 0}}}.Q \; + \; \{x\}_p.\overline{\{\{x\}_{-_i}\}_{r^{cheat}}}.Q +$$
$$\{x\}_p.\overline{\{\{x\}_{-_i}\}_{r^{cheat}}}.P\Big)$$

where $Inc(p) \stackrel{\text{def}}{=} 1 \leq p < n \wedge I_p = (p: \; c_i := c_i + 1; \; \texttt{goto}\ q)$, and $Dec(p) \stackrel{\text{def}}{=} 1 \leq p < n \; \wedge \; I_p = (p: \texttt{if}\ c_i = 0\ \texttt{then goto}\ q\ \texttt{else}\ c_i := c_i - 1; \; \texttt{goto}\ r)$, and $All(p) \stackrel{\text{def}}{=} 1 \leq p \leq n \; \wedge \; 1 \leq i \leq 2$.

We shall now argue that R diverges if and only if

$$\overline{\{t\}_1}.B_K \mid P \;\; \sim \;\; \overline{\{t\}_1}.B_K \mid Q$$

in $\mathcal{T}(\Delta)$ where the knowledge function

$$kn : \mathcal{C}onf \mapsto \{OK_{=0}, OK_{\geq 0}, CHEAT, OTHER\}$$

is defined as follows (let $i \in \{1,2\}$, $X, Y \in \{B_K, P, Q\}$, and $p \in \{1, \ldots, n-1\}$).

$$kn(X \mid \overline{\{m\}_{p^i_{=0}}}.Y) \stackrel{\text{def}}{=} \begin{cases} OK_{=0} & \text{if } \#_{+_i}(m) = \#_{-_i}(m) \\ CHEAT & \text{otherwise} \end{cases}$$

$$kn(X \mid \overline{\{m\}_{p^i_{\geq 0}}}.Y) \stackrel{\text{def}}{=} \begin{cases} OK_{\geq 0} & \text{if } \#_{+_i}(m) \geq \#_{-_i}(m) \\ CHEAT & \text{otherwise} \end{cases}$$

$$kn(X \mid \overline{\{m\}_{p^{cheat}}}.Y) \stackrel{\text{def}}{=} CHEAT$$

For all other configurations C let $kn(C) \stackrel{\text{def}}{=} OTHER$.

Remark 5.3 *Note that kn is a computable and respecting knowledge function, and that its knowledge domain is finite.*

Lemma 5.1 *If* R *halts then the attacker has a winning strategy in the bisimulation game played from the pair of configurations* $\overline{\{\mathsf{t}\}_1}.\mathsf{B}_\mathsf{K} \mid \mathsf{P}$ *and* $\overline{\{\mathsf{t}\}_1}.\mathsf{B}_\mathsf{K} \mid \mathsf{Q}$.

Proof. Assume that $(1,0,0) \longrightarrow^* (n,\nu_1,\nu_2)$ for some $\nu_1,\nu_2 \in \mathbb{N}^0$. We will show that the attacker can force the defender to reach a pair of configurations

$$\overline{\{m\}_n}.\mathsf{B}_\mathsf{K} \mid \mathsf{P} \quad \text{and} \quad \overline{\{m\}_n}.\mathsf{B}_\mathsf{K} \mid \mathsf{Q}$$

for some $m \in \mathcal{M}(\mathsf{T},\{+_i,-_i \mid 1 \le i \le 2\})$ such that $\#_{+_i}(m) - \#_{-_i}(m) = \nu_i$ for $1 \le i \le 2$. From this pair of configurations the attacker wins because

$$\overline{\{m\}_n}.\mathsf{B}_\mathsf{K} \mid \mathsf{P} \longrightarrow \mathsf{B}_\mathsf{K} \mid \overline{\{m\}_n}.\mathsf{P}$$

whereas $\overline{\{m\}_n}.\mathsf{B}_\mathsf{K} \mid \mathsf{Q} \not\longrightarrow$.

In order to show that the attacker can force the defender to faithfully simulate the computation of the Minsky machine, we assume that the current configuration of R is (p,ν_1,ν_2) where $1 \le p < n$ and that the bisimulation game starts from the pair

$$\overline{\{m\}_p}.\mathsf{B}_\mathsf{K} \mid \mathsf{P} \quad \text{and} \quad \overline{\{m\}_p}.\mathsf{B}_\mathsf{K} \mid \mathsf{Q}$$

such that $\#_{+_i}(m) - \#_{-_i}(m) = \nu_i$ for $1 \le i \le 2$. We will show that if $(p,\nu_1,\nu_2) \longrightarrow (p',\nu_1',\nu_2')$ then after two rounds of the bisimulation game the attacker can force the defender to reach a pair of configurations $\overline{\{m'\}_{p'}}.\mathsf{B}_\mathsf{K} \mid \mathsf{P}$ and $\overline{\{m'\}_{p'}}.\mathsf{B}_\mathsf{K} \mid \mathsf{Q}$ such that $\#_{+_i}(m') - \#_{-_i}(m') = \nu_i'$ for $1 \le i \le 2$. There are three situations to be discussed.

(1) If the instruction I_p is of the form

$$p: \ c_i := c_i + 1; \ \texttt{goto}\ q$$

then the attacker makes two moves

$$\overline{\{m\}_p}.\mathsf{B}_\mathsf{K} \mid \mathsf{P} \longrightarrow \mathsf{B}_\mathsf{K} \mid \overline{\{\{m\}_{+_i}\}_q}.\mathsf{P} \longrightarrow \overline{\{\{m\}_{+_i}\}_q}.\mathsf{B}_\mathsf{K} \mid \mathsf{P}$$

and the defender can only answer by

$$\overline{\{m\}_p}.\mathsf{B}_\mathsf{K} \mid \mathsf{Q} \longrightarrow \mathsf{B}_\mathsf{K} \mid \overline{\{\{m\}_{+_i}\}_q}.\mathsf{Q} \longrightarrow \overline{\{\{m\}_{+_i}\}_q}.\mathsf{B}_\mathsf{K} \mid \mathsf{Q}.$$

(2) If the instruction I_p is of the form

$$p: \texttt{if}\ c_i = 0\ \texttt{then goto}\ q\ \texttt{else}\ c_i := c_i - 1; \ \texttt{goto}\ r$$

and $\nu_i = 0$ then the attacker plays

$$\overline{\{m\}_p}.\mathsf{B}_\mathsf{K} \mid \mathsf{P} \longrightarrow \mathsf{B}_\mathsf{K} \mid \overline{\{m\}_{q^i_{=0}}}.\mathsf{P} \longrightarrow \overline{\{m\}_q}.\mathsf{B}_\mathsf{K} \mid \mathsf{P}.$$

This time the defender has six choices how to respond to the first move of the attacker. However, notice that

$$kn(\mathsf{B}_\mathsf{K} \mid \overline{\{m\}_{q^i_{=0}}}.\mathsf{P}) = OK_{=0}$$

because $\nu_i = 0$ and hence the defender can only answer by

$$\overline{\{m\}_p}.B_K \mid Q \longrightarrow B_K \mid \overline{\{m\}_{q^i_{=0}}}.Q \longrightarrow \overline{\{m\}_q}.B_K \mid Q.$$

In all other possible moves the defender loses after the first move because the knowledge function returns different values.

(3) If the instruction I_p is of the form

$$p : \texttt{if } c_i = 0 \texttt{ then goto } q \texttt{ else } c_i := c_i - 1\texttt{; goto } r$$

and $\nu_i > 0$ then the attacker plays

$$\overline{\{m\}_p}.B_K \mid P \longrightarrow B_K \mid \overline{\{\{m\}_{-_i}\}_{r^i_{\geq 0}}}.P \longrightarrow \overline{\{\{m\}_{-_i}\}_r}.B_K \mid P.$$

Again, because

$$kn(B_K \mid \overline{\{\{m\}_{-_i}\}_{r^i_{\geq 0}}}.P) = OK_{\geq 0}$$

the defender can only answer by

$$\overline{\{m\}_p}.B_K \mid Q \longrightarrow B_K \mid \overline{\{\{m\}_{-_i}\}_{r^i_{\geq 0}}}.Q \longrightarrow \overline{\{\{m\}_{-_i}\}_r}.B_K \mid Q.$$

□

Lemma 5.2 *If* R *diverges then the defender has a winning strategy in the bisimulation game played from the pair* $\overline{\{t\}_1}.B_K \mid P$ *and* $\overline{\{t\}_1}.B_K \mid Q$.

Proof. We will show that the defender in the bisimulation game from $\overline{\{t\}_1}.B_K \mid P$ and $\overline{\{t\}_1}.B_K \mid Q$ can force the attacker to faithfully simulate the computation of the Minsky machine. Since the computation of R diverges, the bisimulation game is infinite and the defender wins.

Consider a configuration (p, ν_1, ν_2) where $1 \leq p < n$ of the machine R that was reached during the computation from $(1, 0, 0)$. Let the corresponding pair of protocol configurations be

$$\overline{\{m\}_p}.B_K \mid P \quad \text{and} \quad \overline{\{m\}_p}.B_K \mid Q$$

such that $\#_{+_i}(m) - \#_{-_i}(m) = \nu_i$ for $1 \leq i \leq 2$.

We will show that if $(p, \nu_1, \nu_2) \longrightarrow (p', \nu_1', \nu_2')$ then after two rounds of the bisimulation game the defender can force the attacker to reach a pair of configurations $\overline{\{m'\}_{p'}}.B_K \mid P$ and $\overline{\{m'\}_{p'}}.B_K \mid Q$ such that $\#_{+_i}(m') - \#_{-_i}(m') = \nu_i'$ for $1 \leq i \leq 2$, or the defender can win by reaching a pair of syntactically equal (and hence bisimilar) configurations. There are three situations to be discussed.

(1) If the instruction I_p is of the form

$$p : \ c_i := c_i + 1\texttt{; goto } q$$

then there is only one possible continuation of the bisimulation game such that after two rounds the players reach the pair

$$\overline{\{\{m\}_{+_i}\}_q}.B_K \mid P \ \text{ and } \ \overline{\{\{m\}_{+_i}\}_q}.B_K \mid Q.$$

(2) If the instruction I_p is of the form

$$p : \texttt{if } c_i = 0 \texttt{ then goto } q \texttt{ else } c_i := c_i - 1\texttt{; goto } r$$

and $\nu_i = 0$, the attacker is forced to make either the move $\overline{\{m\}_p}.B_K \mid P \longrightarrow B_K \mid \overline{\{m\}_{q_{=0}^i}}.P$ or $\overline{\{m\}_p}.B_K \mid Q \longrightarrow B_K \mid \overline{\{m\}_{q_{=0}^i}}.Q$ and the game faithfully simulates the computation of R as before. In all other attacker's moves the defender wins:

- if the attacker takes any of the four transitions (either from P or Q) introducing the key of the form p^{cheat} as the upper most encryption, the defender simply mimics the corresponding move in the other configuration except for the fact that he may switch P for Q or vice versa in order to achieve a pair of syntactically equal configurations (the knowledge function will return *CHEAT* in these situations);
- if the attacker takes a transition that should decrement a value of the counter c_i but the counter is already empty, i.e.,

$$\overline{\{m\}_p}.B_K \mid P \longrightarrow B_K \mid \overline{\{\{m\}_{-_i}\}_{r_{\geq 0}^i}}.P \quad \text{or}$$

$$\overline{\{m\}_p}.B_K \mid Q \longrightarrow B_K \mid \overline{\{\{m\}_{-_i}\}_{r_{\geq 0}^i}}.Q,$$

the defender answers by the transitions

$$\overline{\{m\}_p}.B_K \mid Q \longrightarrow B_K \mid \overline{\{\{m\}_{-_i}\}_{r^{cheat}}}.P \quad \text{resp.}$$

$$\overline{\{m\}_p}.B_K \mid P \longrightarrow B_K \mid \overline{\{\{m\}_{-_i}\}_{r^{cheat}}}.Q.$$

Since the attacker "cheated" the knowledge function allows this response (it returns the value *CHEAT* for these configurations). After the second round (there is only one possible continuation of the game which transfers the encrypted messages to the buffer), the protocol configurations become syntactically equal and hence the attacker loses.

(3) If the instruction I_p is of the form

$$p : \texttt{if } c_i = 0 \texttt{ then goto } q \texttt{ else } c_i := c_i - 1\texttt{; goto } r$$

and $\nu_i > 0$, the situation is similar to the previous case. □

Theorem 5.4 *The problem of bisimilarity checking between a pair of ping-pong protocol configurations of width 2 is undecidable (provided that we allow for general but still computable, respecting and finite-domain knowledge functions).*

Proof. From Lemma 5.1 and Lemma 5.2. □

Theorem 5.5 *The problem of bisimilarity checking between a pair of ping-pong protocol configurations of width 2 is decidable for local knowledge functions.*

Proof. We shall reduce our problem to strong bisimilarity checking of PDA. The result then follows from the fact that strong bisimilarity of PDA is decidable [17].

In the reduction we extend the construction provided in the proof of Theorem 5.3. We consider not only the PDA rules

$$(\mathsf{P},\mathsf{Q})\langle v_i\rangle \xrightarrow{a} (\mathsf{Q},\mathsf{P}_i)\langle w_i\rangle$$

for every $\mathsf{P} \in \mathcal{C}onst$ and $\mathsf{Q} \in \mathcal{C}onst \cup \{\mathbf{0}\}$ such that $\mathsf{P} \stackrel{\text{def}}{=} \sum_{i\in I} v_i.\overline{w_i}.\mathsf{P}_i$ but also the rules

$$(\mathsf{P},\mathsf{Q})\langle w'\rangle \xrightarrow{f(\mathsf{P},\mathsf{Q},w')} (\mathsf{P},\mathsf{Q})\langle w'\rangle$$

for all elements $(\mathsf{P},\mathsf{Q},w')$ in the co-domain of the function f. Hence the reduction steps in the ping-pong protocol correspond to the a-labelled transitions in the PDA system, and the condition that the knowledge function agrees on bisimilar states is tested in the PDA by executing loops labelled by the elements from $\mathcal{D}$. It is now easy to see that protocol configurations $\mathsf{P}_1 \mid \overline{w_1}.\mathsf{Q}_1$ and $\mathsf{P}_2 \mid \overline{w_2}.\mathsf{Q}_2$ are bisimilar if and only if the PDA configurations $(\mathsf{P}_1,\mathsf{Q}_1)\langle w_1\rangle$ and $(\mathsf{P}_2,\mathsf{Q}_2)\langle w_2\rangle$ are strongly bisimilar. □

Remark 5.4 *By standard techniques for pushdown automata one can extend the previous theorem to allow a slightly more general definition of local knowledge functions. In particular, the functions kn can depend also on a constant number of inner-most keys of the encrypted message in addition to the* M *outermost keys.*

Bibliography

[1] Martin Abadi and Andrew D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.

[2] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[3] R.M. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 499–514. Springer-Verlag, 2002.

[4] R.M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *LNCS*, pages 380–394. Springer-Verlag, 2000.

[5] Roberto M. Amadio, Denis Lugiez, and Vincent Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, October 2002.

[6] Michele Boreale. Symbolic trace analysis of cryptographic protocols. In *28th Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of *LNCS*, pages 667–681. Springer, July 2001.

[7] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*, pages 135–150. Springer-Verlag, 1997.

[8] J.R. Büchi. Regular canonical systems. *Arch. Math. Logik u. Grundlagenforschung*, 6:91–111, 1964.

[9] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proceedings of Rewriting Techniques and Applications (RTA'03)*, number 2706 in LNCS, pages 148–164. Springer-Verlag, 2003.

[10] D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols. *Information and Control*, 55(1–3):57–68, 1982.

[11] D. Dolev and A.C. Yao. On the security of public key protocols. *Transactions on Information Theory*, IT-29(2):198–208, 1983.

[12] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In N. Heintze and E. Clarke, editors, *Proceedings of Workshop on Formal Methods and Security Protocols (FMSP'99)*, July 1999.

[13] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 232–247. Springer-Verlag, 2000.

[14] M. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *14th IEEE Computer Security Foundations Workshop (CSFW '01)*, pages 160–173, Washington - Brussels - Tokyo, June 2001. IEEE.

[15] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *LNCS*, pages 354–372. Springer-Verlag, 2000.

[16] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *TCS: Theoretical Computer Science*, 299, 2003.

[17] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science(FOCS'98)*, pages 120–129. IEEE Computer Society, 1998.

[18] C. Stirling. Local model checking games. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *LNCS*, pages 1–11. Springer-Verlag, 1995.

[19] W. Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science (extended abstract). In *Proceedings of the 4th International Joint Conference CAAP/FASE, Theory and Practice of Software Development (TAPSOFT'93)*, volume 668 of *LNCS*, pages 559–568. Springer-Verlag, 1993.

Chapter 6

Recursion vs. Replication in Simple Cryptographic Protocols

The paper *Recursion vs. Replication in Simple Cryptographic Protocols* presented in this chapter has been published as the following conference paper.

> Recursion vs. Replication in Simple Cryptographic Protocols by H. Hüttel and J. Srba. In Proceedings of 31st Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'05), pages 175–184, volume 3381 of LNCS, Springer-Verlag, 2005.

An extended version appeared as a technical report.

> Recursion vs. Replication in Simple Cryptographic Protocols by H. Hüttel and J. Srba. Technical report RS-04-23, BRICS Research Series, 26 pages, 2004.

The technical report extends the conference paper by describing all the construction details and full proofs, including the presentation of the active intruder and the reduction of reachability from the replicative ping-pong calculi to weak process rewrite systems.

Except for minor typographical changes the content of this chapter is equal to the technical report.

Recursion vs. Replication in Simple Cryptographic Protocols

Hans Hüttel and Jiří Srba

Abstract. We use some recent techniques from process algebra to draw several conclusions about the well studied class of ping-pong protocols introduced by Dolev and Yao. In particular we show that all nontrivial properties, including reachability and equivalence checking wrt. the whole van Glabbeek's spectrum, become undecidable for a very simple recursive extension of the protocol. The result holds even if no nondeterministic choice operator is allowed. We also show that the extended calculus is capable of an implicit description of the active intruder, including full analysis and synthesis of messages in the sense of Amadio, Lugiez and Vanackère. We conclude by showing that reachability analysis for a replicative variant of the protocol becomes decidable.

6.1 Introduction

Process calculi have been suggested as a natural vehicle for reasoning about cryptographic protocols. In [1], Abadi and Gordon introduced the spi-calculus (a variant of the π-calculus) and described how properties such as secrecy and authenticity can be expressed via notions of observational equivalence (like may-testing). Alternatively, security questions have been studied using reachability analysis [3, 5, 11].

We provide a basic study of expressiveness and feasibility of cryptographic protocols. We are interested in two verification approaches: *reachability analysis* and *equivalence (preorder) checking*. In reachability analysis the question is whether a certain (bad or good) configuration of the protocol is reachable from a given initial one. In equivalence checking the question is whether a protocol implementation is equivalent (e.g. bisimilar) to a given specification (optimal behaviour). These verification strategies can be used even in the presence of an *active intruder* (in the Dolev-Yao style), i.e., an agent with capabilities to listen to any communication, to perform analysis and synthesis of communicated messages according to the actual knowledge of compromised keys, and to actively participate in the protocol behaviour by transmitting new messages. This can be naturally implemented not only into the reachability analysis (see e.g. [4]) but also into the equivalence checking approach. As described in [12], these questions for equivalence (preorder) checking approach can be formulated as follows: "a protocol P guarantees a security property X if, whatever hostile environment E with a certain initial knowledge ϕ_{I}, then P is equivalent (in preorder) to (with) the specification $\alpha(\mathsf{P})$." Formally this is given by saying that

$$\text{protocol P satisfies property X} \quad \text{iff} \quad \forall \mathsf{E} \in \mathcal{E} : \ \mathsf{P}|\mathsf{E} \approx \alpha(\mathsf{P}). \tag{6.1}$$

By an appropriate choice of the specification function α and a suitable equivalence (preorder) $\approx$, several security properties can be verified. Here is a small selection:

- *Secrecy* (confidential information should be available only to the partners of the communication). Here $\approx$ stands for trace preorder.
- *(Message) authenticity* (identification of other agents (messages) participating in communication). Here $\approx$ stands for trace equivalence or preorder.
- *Fairness* (in a contract, no party can gain advantage by ending the protocol prematurely). Here $\approx$ stands for failure equivalence.

Various notions of bisimilarity are studied in this context as bisimilarity is usually the "most decidable behavioral equivalence" as confirmed e.g. by several positive decidability results in process algebra [6]. Hence the questions whether a certain class of cryptographic protocols has decidable reachability and equivalence (bisimilarity) checking are of particular importance for automated verification.

A number of security properties are decidable for finite protocols [3, 19]. In the case of an unbounded number of protocol configurations, the picture is more complex. Durgin et al. showed in [10] that security properties are undecidable in a restricted class of so-called bounded protocols (that still allows for infinitely many reachable configurations). In [2] Amadio and Charatonik consider a language of tail-recursive protocols with bounded encryption depth and name generation; they show that, whenever certain restrictions on decryption are violated, one can encode two-counter machines in the process language. On the other hand, Amadio, Lugiez and Vanackère show in [4] that the reachability problem is in PTIME for a class of protocols with iteration.

In this paper we focus solely on ping-pong based behaviours of recursive and replicative protocols (perhaps the simplest behaviour of all studied calculi) in order to draw general conclusions about expressiveness and tractability of formal verification of cryptographic protocols. The class of *ping-pong protocols* was introduced in 1983 by Dolev and Yao [9]. The formalism deals with memory-less protocols which may be subjected to arbitrarily long attacks. Here, the secrecy of a finite ping-pong protocol can be decided in polynomial time. Later, Dolev, Even and Karp found a cubic-time algorithm [8]. The class of protocols studied in [4] contains iterative ping-pong protocols and, as a consequence, secrecy properties remain polynomially decidable even in this case.

In the present paper we continue our study of recursive and replicative extensions of ping-pong protocols. In [14] we showed that the recursive extension of the calculus is Turing powerful, however, the nondeterministic choice operator appeared to be essential in the construction. The question whether the calculus is Turing powerful even without any explicit way to define nondeterministic processes was left open. Here we present a radically new reduction from multi-stack automata and strengthen the undecidability results to hold even for protocols without nondeterministic choice. We prove, in particular,

that both reachability and equivalence checking for all equivalences and preorders between trace equivalence/preorder and isomorphism of labelled transition systems (which includes all equivalences and preorders from van Glabbeek's spectrum [20]) become undecidable. These results are of general importance because they prove the impossibility of automated verification for essentially all recursive cryptographic protocols capable of at least the ping-pong behaviour.

In the initial study from [14], the question of active attacks on the protocol was not dealt with. We shall demonstrate that a complete notion of the active intruder (including analysis and synthesis of messages in the sense of Amadio, Lugiez and Vanackère [4]) can be explicitly encoded into our formalism in order to analyze general properties like in the scheme (6.1).

Finally, we study a replicative variant of the calculus. Surprisingly, such a calculus becomes decidable, at least with regard to reachability analysis. We use a very recent result from process algebra (decidability of reachability for weak process rewrite systems by Křetínský, Řehák and Strejček [15]) in order to derive the result. We believe that this is one of the reasons which formally confirm the general trend that replication is a good choice for cryptographic formalisms and that is why recursion is only rarely studied.

6.2 Basic definitions

6.2.1 Labelled transition systems with label abstraction

In order to provide a uniform framework for our study of ping-pong protocols, we define their semantics by means of labelled transition systems. A *labelled transition system* (LTS) is a triple $\mathcal{T} = (S, \mathcal{A}ct, \longrightarrow)$ where S is a set of *states* (or *processes*), $\mathcal{A}ct$ is a set of *labels* (or *actions*), and $\longrightarrow \subseteq S \times \mathcal{A}ct \times S$ is a *transition relation*, written $\alpha \xrightarrow{a} \beta$, for $(\alpha, a, \beta) \in \longrightarrow$. As usual we extend the transition relation to the elements of $\mathcal{A}ct^*$. We also write $\alpha \longrightarrow^* \beta$, whenever $\alpha \xrightarrow{w} \beta$ for some $w \in \mathcal{A}ct^*$.

The idea is that the states represent *global configurations* of a given protocol and the transitions describe the *information flow*. Labels on the transitions moreover represent the messages (both plain-text and cipher-text) which are being communicated during the state changes.

Remark 6.1 *In [14] the semantics of ping-pong protocols is given in terms of transition systems with knowledge, i.e., unlabelled transition systems where each state it assigned its knowledge, represented as a subset of a certain set of all possible knowledge values. By standard techniques such a knowledge-based semantics can be translated to labelled transition systems and the studied verification properties (reachability, equivalence checking, etc.) are preserved. For example a state* A *with two knowledge values* p_1 *and* p_2 *can be transformed to a labelled transition system where the values* p_1 *and* p_1 *are represented as self-loops in state* A *which are visible under special actions* p_1 *and* p_2*. A fresh action* a *is used to represent the change of the state (the unlabelled transitions in the original knowledge-based semantics).*

The explicit possibility to observe the full content of messages is sometimes not very realistic; it means that an external observer of such a system can e.g. distinguish between two different messages encrypted by the same encryption key, without the actual knowledge of the key.

In order to restrict capabilities of the observer we introduce a so called *label abstraction function* $\phi : \mathcal{A}ct \mapsto \mathcal{A}ct$. Given a LTS $\mathcal{T} = (S, \mathcal{A}ct, \longrightarrow_{\mathcal{T}})$ and a label abstraction function ϕ we define a new LTS $\mathcal{T}_\phi \stackrel{\text{def}}{=} (S, \mathcal{A}ct, \longrightarrow_{\mathcal{T}_\phi})$ where $\alpha \stackrel{\phi(a)}{\longrightarrow}_{\mathcal{T}_\phi} \beta$ iff $\alpha \stackrel{a}{\longrightarrow}_{\mathcal{T}} \beta$ for all $\alpha, \beta \in S$ and $a \in \mathcal{A}ct$. We call $\mathcal{T}_\phi$ a *labelled transition system with label abstraction.*

Let us now focus on the messages (actions). Assume a given set of encryption keys $\mathcal{K}$. The set of all messages over $\mathcal{K}$ is given by the following abstract syntax

$$m ::= k \mid k \cdot m$$

where k ranges over $\mathcal{K}$. Hence every element of the set $\mathcal{K}$ is a *(plain-text) message* and if m is a message then $k \cdot m$ is a *(cipher-text) message* (meaning that the message m is encrypted by the key k). Given a message $k_1 \cdot k_2 \cdots k_n$ over $\mathcal{K}$ we usually[1] write it only as a word $k_1 k_2 \cdots k_n$ from $\mathcal{K}^*$. Note that k_n is the plain-text part of the message and the outermost encryption key is always on the left (k_1 in our case). In what follows we shall identify the set of messages and $\mathcal{K}^*$, and we denote the extra element of $\mathcal{K}^*$ consisting of the empty sequence of keys by ϵ.

Example 6.1 *Let us consider a labelled transition system* $\mathcal{T} \stackrel{\text{def}}{=} (S, \mathcal{A}ct, \longrightarrow)$ *where* $S \stackrel{\text{def}}{=} \{A, B, C\}$, $\mathcal{A}ct \stackrel{\text{def}}{=} \mathcal{K}^*$ *for a given set of keys* $\mathcal{K} = \{k_1, k_2, \lambda\}$ *and* $\longrightarrow$ *is given by the following picture.*

$$A \xrightarrow{k_1 k_2} B \xrightarrow{k_2} C$$

The protocol computation starts in the state A *and is very simple. First a plain-text* k_2 *encrypted by the encryption key* k_1 *is communicated to the process* B*, which decrypts the message and sends out the plain-text* k_2*. Let us now assume a label abstraction function* ϕ *defined by* $\phi(k) = k$ *if* $k \in \mathcal{K}$ *and* $\phi(m) = \lambda$ *otherwise. The labelled transition system* T_ϕ *with label abstraction function* ϕ *now looks as follows.*

$$A \xrightarrow{\lambda} B \xrightarrow{k_2} C$$

This translates to the fact that the external observer is not allowed to see the content of encrypted messages (the action λ *is used instead) and only plain-text messages can be recognized.*

The level of abstraction we may select depends on the particular studied property we are interested in and it directly corresponds to the specification

[1] In our previous work on ping-pong protocols [14] we denoted a message m encrypted by a key k as $\{m\}_k$. We changed the notation in order to improve the clarity of the proofs. In particular, when messages like $k_1 k_2 \cdots k_n$ are used, the previous syntax described the keys in a reversed order, which was technically inconvenient.

function α from (6.1). Nevertheless, it seems reasonable to require at least the possibility to distinguish between plain-text and cipher-text messages. We say that a label abstraction function ϕ is *reasonable* iff $\phi(k) \neq \phi(k'w)$ for all $k, k' \in \mathcal{K}$ and $w \in \mathcal{K}^+$.

6.2.2 A calculus of recursive ping-pong protocols

We shall now define a calculus which captures exactly the class of ping-pong protocols by Dolev and Yao [9] extended (in a straightforward manner) with recursive definitions.

Let $\mathcal{K}$ be a set of encryption keys. A *specification* of a recursive ping-pong is a finite set of process definitions Δ such that for every *process constant* P (from a given set $\mathcal{C}onst$) the set Δ contains exactly one process definition of the form

$$P \stackrel{\text{def}}{=} \sum_{i_1 \in I_1} v_{i_1} \rhd . \overline{w_{i_1} \rhd}.P_{i_1} + \sum_{i_2 \in I_2} v_{i_2}.P_{i_2} + \sum_{i_3 \in I_3} \overline{w_{i_3}}.P_{i_3}$$

where I_1, I_2 and I_3 are finite sets of indices such that $I_1 \cup I_2 \cup I_3 \neq \emptyset$, and v_{i_1}, v_{i_2}, w_{i_1} and w_{i_3} are messages (belong to $\mathcal{K}^*$) for all $i_1 \in I_1$, $i_2 \in I_2$ and $i_3 \in I_3$, and $P_i \in \mathcal{C}onst \cup \{\mathbf{0}\}$ for all $i \in I_1 \cup I_2 \cup I_3$ such that $\mathbf{0}$ is a special constant called the *empty process*. We moreover require that v_{i_2} and w_{i_3} for all $i_2 \in I_2$ and $i_3 \in I_3$ are different from the empty message ϵ. (Observe that any specification Δ contains only finitely many keys.)

Summands continuing in the empty process constant $\mathbf{0}$ will be written without the $\mathbf{0}$ symbol and process definitions will often be written in their unfolded form using the *nondeterministic choice operator* '+'. An example of a process definition is e.g. $P \stackrel{\text{def}}{=} k_1 \rhd . \overline{k_2 \rhd}.P_1 + k_1 \rhd . \overline{k_3 \rhd} + k_1 k_2.P_1 + \overline{k_1 k_1} + \overline{k_1 k_2}.P_2$.

The intuition is that each summand of the form $v_{i_1} \rhd . \overline{w_{i_1} \rhd}.P_{i_1}$ can receive a message encrypted by a sequence v_{i_1} of outermost keys, decrypt the message using these keys, send it out encrypted by the sequence of keys w_{i_1}, and finally behave as the process constant P_{i_1}. The symbol $\rhd$ stands for the rest of the message after decrypting it with the key sequence v_{i_1}. This describes a standard ping-pong behaviour of the process.

In addition to this we may have summands of the forms $v_{i_2}.P_{i_2}$ and $\overline{w_{i_3}}.P_{i_3}$, meaning simply that a message is received and forgotten or unconditionally transmitted, respectively. This is a small addition to the calculus we presented in [14] in order to allow for discarding of old messages and generation of new messages. These two features were not available in the earlier version of the calculus but they appear to be technically convenient when modeling an explicit intruder and for strengthening the positive decidability results in Section 6.5. Nevertheless, the undecidability results presented in Section 6.3 are valid even without this extension since only the standard ping-pong behaviour is used in the constructions. A feature very similar to the forgetful input operation can be also found in [4].

A *configuration* of a ping-pong protocol specification Δ is a parallel composition of process constants, possibly preceded by output messages. Formally

the set $\mathcal{C}onf$ of configurations is given by the following abstract syntax

$$\mathsf{C} ::= \quad \mathbf{0} \quad | \quad \mathsf{P} \quad | \quad \overline{w}.\mathsf{P} \quad | \quad \mathsf{C}|\mathsf{C}$$

where $\mathbf{0}$ is the empty configuration, $\mathsf{P} \in \mathcal{C}onst \cup \{\mathbf{0}\}$ ranges over process constants including the empty process, $w \in \mathcal{K}^*$ ranges over the set of messages, and '|' is the operator of parallel composition.

We introduce a structural congruence relation $\equiv$ which identifies configurations that represent the same state of the protocol. The relation $\equiv$ is defined as the least congruence over configurations ($\equiv \subseteq \mathcal{C}onf \times \mathcal{C}onf$) such that $(\mathcal{C}onf, |, \mathbf{0})$ is a commutative monoid and $\overline{\epsilon}.\mathsf{P} \equiv \mathsf{P}$ for all $\mathsf{P} \in \mathcal{C}onst$. In what follows we shall identify configurations up to structural congruence.

Remark 6.2 *We let* $\overline{\epsilon}.\mathsf{P} \equiv \mathsf{P}$ *because the empty message should never be communicated. This means that when a prefix like* $\mathsf{k}\triangleright\,.\,\overline{\triangleright}.\mathsf{P}$ *receives a plain-text message* k *and tries to output* $\overline{\epsilon}.\mathsf{P}$*, it simply continues as the process* P*.*

We shall now define the semantics of ping-pong protocols in terms of labelled transition systems. We define a set $\mathcal{C}onf_S \subseteq \mathcal{C}onf$ consisting of all configurations that do not contain the operator of parallel composition and call these *simple configurations*. We also define two sets $In(\mathsf{C}, m), Out(\mathsf{C}, m) \subseteq \mathcal{C}onf_S$ for all $\mathsf{C} \in \mathcal{C}onf_S$ and $m \in \mathcal{K}^+$. The intuition is that $In(\mathsf{C}, m)$ ($Out(\mathsf{C}, m)$) contains all configurations which can be reached from the simple configuration C after receiving (resp. outputting) the message m from (to) the environment. Formally, $In(\mathsf{C}, m)$ and $Out(\mathsf{C}, m)$ are the smallest sets which satisfy:

- $\mathsf{Q} \in In(\mathsf{P}, m)$ whenever $\mathsf{P} \in \mathcal{C}onst$ and $m.\mathsf{Q}$ is a summand of P
- $\overline{w\alpha}.\mathsf{Q} \in In(\mathsf{P}, m)$ whenever $\mathsf{P} \in \mathcal{C}onst$ and $\nu\triangleright\,.\,\overline{w\triangleright}.\mathsf{Q}$ is a summand of P such that $m = \nu\alpha$
- $\mathsf{P} \in Out(\overline{m}.\mathsf{P}, m)$ whenever $\mathsf{P} \in \mathcal{C}onst \cup \{\mathbf{0}\}$
- $\mathsf{Q} \in Out(\mathsf{P}, m)$ whenever $\mathsf{P} \in \mathcal{C}onst$ and $\overline{m}.\mathsf{Q}$ is a summand of P.

A given protocol specification Δ determines a labelled transition system $\mathsf{T}(\Delta) \stackrel{\text{def}}{=} (\mathsf{S}, \mathcal{A}ct, \longrightarrow)$ where the states are configurations of the protocol modulo the structural congruence ($\mathsf{S} \stackrel{\text{def}}{=} \mathcal{C}onf/\equiv$), the set of labels (actions) is the set of messages that can be communicated between the agents of the protocol ($\mathcal{A}ct \stackrel{\text{def}}{=} \mathcal{K}^+$), and the transition relation $\longrightarrow$ is given by the following SOS rule (recall that '|' is commutative).

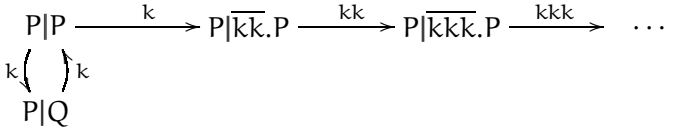
$$\frac{m \in \mathcal{K}^+ \qquad \mathsf{C}_1, \mathsf{C}_2 \in \mathcal{C}onf_S \qquad \mathsf{C}_1' \in Out(\mathsf{C}_1, m) \qquad \mathsf{C}_2' \in In(\mathsf{C}_2, m)}{\mathsf{C}_1|\mathsf{C}_2|\mathsf{C} \stackrel{m}{\longrightarrow} \mathsf{C}_1'|\mathsf{C}_2'|\mathsf{C}}$$

This means that (in the context C) two simple configurations (agents) C_1 and C_2 can communicate a message m in such a way that C_1 outputs m and becomes C_1' while C_2 receives the message m and becomes C_2'.

Example 6.2 *Let us consider a protocol specification* Δ*.*

$$\mathsf{P} \stackrel{\text{def}}{=} \overline{\mathsf{k}}.\mathsf{P} \;+\; \mathsf{k}\triangleright\,.\,\overline{\mathsf{kk}\triangleright}.\mathsf{P} \;+\; \mathsf{k}.\mathsf{Q} \qquad\qquad \mathsf{Q} \stackrel{\text{def}}{=} \mathsf{k}.\mathsf{P}$$

A fragment of the labelled transition system reachable from the initial configuration $\mathsf{P}|\mathsf{P}$ *looks as follows.*



For further discussion and examples of recursive ping-pong protocols we refer the reader to [14].

6.2.3 Reachability and behavioural equivalences

One of the problems that is usually studied is that of *reachability analysis*: given two configurations $\mathsf{C}_1, \mathsf{C}_2 \in \mathcal{C}onf$ we ask whether C_2 is reachable from C_1, i.e., if $\mathsf{C}_1 \longrightarrow^* \mathsf{C}_2$. In this case the set of labels is irrelevant.

As the semantics of our calculus is given in terms of labelled transition systems (together with an appropriate label abstraction function), we can also study the *equivalence checking* problems. Given some behavioural equivalence or preorder $\leftrightarrow$ from van Glabbeek's spectrum [20] (e.g. strong bisimilarity or trace, failure and simulation equivalences/preorders just to mention a few) and two configurations $\mathsf{C}_1, \mathsf{C}_2 \in \mathcal{C}onf$ of a protocol specification Δ, the question is to decide whether C_1 and C_2 are $\leftrightarrow$-equivalent (or $\leftrightarrow$-preorder related) in $\mathsf{T}(\Delta)$, i.e., whether $\mathsf{C}_1 \leftrightarrow \mathsf{C}_2$.

6.3 Recursive ping-pong protocols without explicit choice

In this section we strengthen the undecidability result from [14] and show that the reachability and equivalence checking problems are undecidable for ping-pong protocols without an explicit operator of nondeterminism and using classical ping-pong behaviour only, i.e., for protocols without any occurrence of the choice operator '+' and where every defining equation is of the form $\mathsf{P} \stackrel{\text{def}}{=} \nu\triangleright\,.\,\overline{w\triangleright}.\mathsf{P}'$ such that $\mathsf{P}' \in \mathcal{C}onst$.

Remark 6.3 *Note that every process constant is allowed to have exactly one defining equation, however, no constraints are imposed on the communication behaviour of the parallel components.*

We moreover show that the negative results apply to all behavioural equivalences and preorders between trace equivalence/preorder and isomorphism of LTS (which preserves labelling) with regard to all reasonable label abstraction functions as defined in Section 6.2.

These results are achieved by showing that recursive ping-pong protocols can step-by-step simulate a Turing powerful computational device, in our case a computational model called multi-stack machines.

A *multi-stack machine* R with ℓ stacks ($\ell \geq 1$) is a triple $\mathsf{R} = (Q, \Gamma, \longrightarrow)$ where Q is a finite set of *control-states*, Γ is a finite *stack alphabet* such that $Q \cap \Gamma = \emptyset$, and $\longrightarrow \subseteq Q \times \Gamma \times Q \times \Gamma^*$ is a finite set of *transition rules*, written $pX \longrightarrow q\alpha$ for $(p, X, q, \alpha) \in \longrightarrow$.

A *configuration* of a multi-stack machine R is an element from $Q \times (\Gamma^*)^\ell$. We assume a given initial configuration $(q_0, w_1, \ldots, w_\ell)$ where $q_0 \in Q$ and $w_i \in \Gamma^*$ for all i, $1 \leq i \leq \ell$. If some of the stacks w_i are empty, we denote them by ϵ.

A *computational step* is defined such that whenever there is a transition rule $pX \longrightarrow q\alpha$ then a configuration which is in the control-state p and has X on top of the i'th stack (the tops of the stacks are on the left) can perform the following transition:

$$(p, w_1, \ldots, Xw_i, \ldots, w_\ell) \longrightarrow (q, w_1, \ldots, \alpha w_i, \ldots, w_\ell)$$

for all $w_1, \ldots, w_\ell \in \Gamma^*$ and for all i, $1 \leq i \leq \ell$.

It is a folklore result that multi-stack machines are Turing powerful. Hence (in particular) the following problem is easily seen to be undecidable: given an initial configuration $(q_0, w_1, \ldots, w_\ell)$ of a multi-stack machine R, can we reach the configuration $(h, \epsilon, \ldots, \epsilon)$ for a distinguished halting control-state $h \in Q$ such that all stacks are empty? Without loss of generality we can even assume that a configuration in the control-state h is reachable iff all stacks are empty.

Let $\mathsf{R} = (Q, \Gamma, \longrightarrow)$ be a multi-stack machine. We define the following set of keys of a ping-pong specification Δ: $\mathcal{K} \stackrel{\text{def}}{=} Q \cup \Gamma \cup \{k_p \mid p \in Q\} \cup \{t, k_*\}$. Here t is a special key such that every communicated message is an encryption of the plain-text key t. The reason for this is that it ensures that the protocol never communicates any plain-text message. The key k_* is a special purpose locking key and it is explained later on in the construction.

We shall construct a ping-pong protocol specification Δ as follows.

- For every transition rule $pX \longrightarrow q\alpha$ we have a process constant $\mathsf{P}_{pX \longrightarrow q\alpha}$ with the following defining equation.

$$\mathsf{P}_{pX \longrightarrow q\alpha} \stackrel{\text{def}}{=} pX\triangleright \,.\, \overline{k_q \alpha \triangleright}.\mathsf{P}_{pX \longrightarrow q\alpha}$$

- For every state $p \in Q$ we have two process constants T_p and T'_p.

$$\mathsf{T}_p \stackrel{\text{def}}{=} k_p\triangleright \,.\, \overline{k_*\triangleright}.\mathsf{T}'_p$$

$$\mathsf{T}'_p \stackrel{\text{def}}{=} k_*\triangleright \,.\, \overline{p\triangleright}.\mathsf{T}_p \quad \text{if } p \in Q \setminus \{h\}, \text{ and} \qquad \mathsf{T}'_h \stackrel{\text{def}}{=} h\triangleright \,.\, \overline{h\triangleright}.\mathsf{T}'_h$$

  Recall that $h \in Q$ is the halting control-state.

- Finally, we define a process constant B (standing for a buffer over a fixed key k_*).

$$\mathsf{B} \stackrel{\text{def}}{=} k_*\triangleright \,.\, \overline{k_*\triangleright}.\mathsf{B}$$

  In this defining equation the key k_* locks the content of the buffer such that it is accessible only by some T'_p.

Note that Δ does not contain any choice operator '+' as required.

Let $(q_0, w_1, \ldots, w_\ell)$ be an initial configuration of the multi-stack machine R. The corresponding initial configuration of the protocol Δ is defined as follows (the meta-symbol Π stands for a parallel composition of the appropriate components).

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \mid \Big(\prod_{p\in Q\setminus\{q_0\}} T_p\Big) \mid T'_{q_0} \mid \Big(\prod_{j\in\{1,\ldots,\ell\}} \overline{k_* w_j t}.B\Big) \qquad (6.2)$$

The following invariants will be preserved during any computational sequence starting from this initial configuration:

- at most one T'_p for some $p \in Q$ is present as a parallel component (the intuition is that this represents the fact that the machine R is in the control-state p), and
- plain-text messages are never communicated.

Let $(p, w_1, \ldots, w_i, \ldots w_\ell) \longrightarrow (q, w_1, \ldots, \alpha w'_i, \ldots w_\ell)$ be a computational step of R using the rule $pX \longrightarrow q\alpha$ such that $w_i = Xw'_i$. This one step is simulated by a sequence of four transitions in the ping-pong protocol Δ (see Figure 6.3). In the first step one buffer is selected and unlocked (the current control-state p replaces the locking key k_* in the outermost encryption). In particular the buffer $\overline{k_* w_i t}.B$ can be unlocked. No other kinds of transitions are possible in the first step. Opening of the selected buffer means that some of the process constants $P_{rA\longrightarrow s\beta}$ become able to accept this message. In particular the process constant $P_{pX\longrightarrow q\alpha}$ can receive the message and output $\overline{k_q \alpha w_i t}$ for further communication; the key k_q determines the control-state change. (At this stage also a communication between $\overline{k_* w_i t}.B$ and B is enabled but it does not change the current state and hence it cannot contribute to a computational progress.) In the next step only T_q can receive the message $\overline{k_q \alpha w_i t}$, it remembers the new control-state q by becoming T'_q and offers the k_*-locked message for a communication with B. This last communication (when B receives back the modified buffer) ends a simulation of one computational step of R.

The following property is easy to see: if after some number of steps starting in a protocol configuration corresponding to $(p, w_1, \ldots, w_\ell)$ we reach a first protocol configuration where T'_q appears for some $q \in Q$ then this corresponds to one correct computational step in R. On the other hand, the computation of Δ can get stuck after the first communication step (in case that the unlocked buffer does not enable an application of any rule $rA \longrightarrow s\beta$) or an infinite sequence of communication steps of the form $\overline{m}.B|B \xrightarrow{m} \overline{m}.B|B$ is also possible.

This is formally captured in the following lemma.

Lemma 6.1 *In the given multi-stack machine* R *the configuration* $(q, w'_1, \ldots, w'_\ell)$ *is reachable from* $(p, w_1, \ldots, w_\ell)$ *if and only if*

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \mid \Big(\prod_{p\in Q\setminus\{q\}} T_p\Big) \mid T'_q \mid \Big(\prod_{j\in\{1,\ldots,\ell\}} \overline{k_* w'_j t}.B\Big)$$

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \mid \Big(\prod_{r\in Q\setminus\{p\}} T_r\Big) \mid T'_p \mid \Big(\prod_{j\in\{1,\ldots,\ell\}} \overline{k_* w_j t}.B\Big)$$

$$\downarrow k_* w_i t$$

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \mid \Big(\prod_{r\in Q\setminus\{p\}} T_r\Big) \mid \overline{p w_i t}.T_p \mid \Big(\prod_{j\in\{1,\ldots,\ell\},j\neq i} \overline{k_* w_j t}.B\Big) \mid B$$

$$\downarrow p w_i t$$

$$\Big(\prod_{(r,A,s,\beta)\in(\longrightarrow\setminus\{(p,X,q,\alpha)\})} P_{rA\longrightarrow s\beta}\Big) \mid \overline{k_q \alpha w'_i t}.P_{pX\longrightarrow q\alpha} \mid \Big(\prod_{r\in Q} T_r\Big) \mid \Big(\prod_{j\in\{1,\ldots,\ell\},j\neq i} \overline{k_* w_j t}.B\Big) \mid B$$

$$\downarrow k_q \alpha w'_i t$$

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \mid \Big(\prod_{r\in Q\setminus\{q\}} T_r\Big) \mid \overline{k_* \alpha w'_i t}.T'_q \mid \Big(\prod_{j\in\{1,\ldots,\ell\},j\neq i} \overline{k_* w_j t}.B\Big) \mid B$$

$$\downarrow k_* \alpha w'_i t$$

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \mid \Big(\prod_{r\in Q\setminus\{q\}} T_r\Big) \mid T'_q \mid \Big(\prod_{j\in\{1,\ldots,\ell\},j\neq i} \overline{k_* w_j t}.B\Big) \mid \overline{k_* \alpha w'_i t}.B$$

Figure 6.1: Simulation of $(p, w_1, \ldots, w_i, \ldots w_\ell) \longrightarrow (q, w_1, \ldots, w'_i\alpha, \ldots w_\ell)$ s.t. $w_i = Xw'_i$

is reachable (in Δ) from

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \mid \Big(\prod_{p\in Q\setminus\{p\}} T_p\Big) \mid T'_p \mid \Big(\prod_{j\in\{1,\ldots,\ell\}} \overline{k_* w_j t}.B\Big).$$

The following theorems are now easily derived.

Theorem 6.1 *The reachability problem for recursive ping-pong protocols without an explicit choice operator is undecidable.*

Proof. Immediately from Lemma 6.1 and from the undecidability of reachability for multi-stack machines. □

Theorem 6.2 *The equivalence checking problem for recursive ping-pong protocols without an explicit choice operator is undecidable for any behavioral equivalence/preorder between trace equivalence/preorder and isomorphism (including all equivalences and preorders from van Glabbeek's spectrum [20]) and for any reasonable label abstraction function.*

Proof. Let R be a multi-stack machine and Δ the protocol specification constructed above with the initial configuration C as given by (6.2). We consider the question whether $\mathsf{C}|\overline{\mathsf{h}}$ is equivalent (or in preorder) with C.

In case that the halting control-state h is not reachable from the initial configuration of R, we know from Lemma 6.1 that T'_{h} will never appear as a parallel component in any reachable state from C. This implies that the plain-text message $\overline{\mathsf{h}}$ will never be communicated and hence $\mathsf{C}|\overline{\mathsf{h}}$ and C exhibit isomorphic behaviours under any label abstraction function.

On the other hand, if h is reachable from the initial configuration of R then because of Lemma 6.1 a configuration in Δ with the parallel component T'_{h} is reachable. Such a configuration is stuck in the process on the right, however, in the process on the left the plain-text message h can be communicated between T'_{h} and the extra parallel component $\overline{\mathsf{h}}$. This means that $\mathsf{C}|\overline{\mathsf{h}}$ and C are not even related by the trace preorder (and hence they are also not trace equivalent) because after a finite sequence of communicated messages there is a successor of the configuration $\mathsf{C}|\overline{\mathsf{h}}$ which can communicate the plain-text h while (as argued before) C can only exchange cipher-text messages. As the label abstraction function ϕ is reasonable, necessarily for all messages m (cipher-texts) communicated in C it is the case that $\phi(\mathsf{m}) \neq \phi(\mathsf{h})$.

To sum up, if the machine R cannot reach the halting configuration then $\mathsf{C}|\overline{\mathsf{h}}$ and C are isomorphic and if R halts than $\mathsf{C}|\overline{\mathsf{h}}$ and C are not in trace preorder. This implies that all equivalences and preorders between trace and isomorphism are undecidable for any reasonable label abstraction function. □

6.4 The active intruder

In the literature on applying process calculi to the study of cryptographic protocols, there have been several proposals for explicit modelling the active intruder (environment). Foccardi, Gorrieri and Martinelli in [12] express the environment within the process calculus, namely as a process running in parallel with the protocol. In [4] Amadio, Lugiez and Vanackère describe a tiny process calculus similar to ours, except that they use replication instead of recursion. Moreover, the environment is described in the semantics of the calculus. Transitions are of the form

$$(\mathsf{C}, \mathsf{T}) \to (\mathsf{C}', \mathsf{T}')$$

where C and C' are protocol configurations and T and T' denote the sets of messages known to the environment (all communication occurs only by passing messages through these sets).

The environment is assumed to be hostile; it may compute new messages by means of the operations of analysis and synthesis and pass these on to the

process. Let $\mathcal{K}$ be a set of encryption keys as before. The *analysis* of a set of messages $T \subseteq \mathcal{K}^*$ is the least set $\mathcal{A}(T)$ satisfying

$$\mathcal{A}(T) = T \cup \{w \mid kw \in \mathcal{A}(T),\ k \in \mathcal{K} \cap \mathcal{A}(T)\}. \tag{6.3}$$

The *synthesis* of a set of messages $T \subseteq \mathcal{K}^*$ is the least set $\mathcal{S}(T)$ satisfying

$$\mathcal{S}(T) = \mathcal{A}(T) \cup \{kw \mid w \in \mathcal{S}(T),\ k \in \mathcal{K} \cap \mathcal{S}(T)\}. \tag{6.4}$$

The next lemmas follow immediately from Tarski's fixed-point theorem.

Lemma 6.2 *The analysis of a set of messages $T \subseteq \mathcal{K}^*$ is the union of the family of sets $\mathcal{A}_i(S)$ defined by*

$$\begin{aligned} \mathcal{A}_0(T) &= T \\ \mathcal{A}_{i+1}(T) &= \mathcal{A}_i(T) \cup \{w \mid kw \in \mathcal{A}_i(T),\ k \in \mathcal{K} \cap \mathcal{A}_i(T)\} \end{aligned}$$

Lemma 6.3 *The synthesis of a set of messages $T \subseteq \mathcal{K}^*$ is the union of the family of sets $\mathcal{S}_i(T)$ defined by*

$$\begin{aligned} \mathcal{S}_0(T) &= \mathcal{A}(T) \\ \mathcal{S}_{i+1}(T) &= \mathcal{S}_i(T) \cup \{kw \mid w \in \mathcal{S}_i(T),\ k \in \mathcal{K} \cap \mathcal{S}_i(T)\} \end{aligned}$$

The set of *compromised keys* K_c for a given set $T \subseteq \mathcal{K}^*$ of messages is defined by $K_c \stackrel{\text{def}}{=} \mathcal{K} \cap \mathcal{S}(T)$, which is easily seen to be equal to $\mathcal{K} \cap \mathcal{A}(T)$. (The compromised keys are immediately available for the intruder because they are either in his initial knowledge or can be discovered by the analysis.)

Remark 6.4 *Let $T \subseteq \mathcal{K}^*$ be a given set of messages. The following observation is easy to verify: in order to compute the complete set K_c of compromised keys of size n, it is enough to find messages $m_1, \ldots, m_n \in T$ such that when we analyze them in a sequence, we discover exactly all compromised keys. Formally, we define*

$$K_c^0 \stackrel{\text{def}}{=} \emptyset \quad \text{and} \quad K_c^i \stackrel{\text{def}}{=} K_c^{i-1} \cup \{k \in \mathcal{K} \mid m_i = wk,\ w \in (K_c^{i-1})^*\}$$

for all i, $1 \le i \le n$, and then $K_c = K_c^n$.

Proposition 6.1 *It holds that $w \in \mathcal{S}(T)$ if and only if w can be written as $w = uw'$ for some $u \in K_c^*$ and there exists $u' \in K_c^*$ such that $u'w' \in T$.*

Proof. Notice that because of Lemma 6.2 $w \in \mathcal{A}(T)$ iff there is $u \in K_c^*$ such that $uw \in T$. The proposition then follows by an application of Lemma 6.3. □

We can now design an environment sensitive semantics for our calculus close in style to that of [4]. We define the reduction relation $\rightarrow$ by the following set of axioms (here $x \in P$ means that x is a summand in the defining equation of the process constant P).

$$
\begin{array}{rcll}
(\mathsf{P}|\mathsf{C},\mathsf{T}) & \rightarrow & (\overline{w\alpha}.\mathsf{P}'|\mathsf{C},\mathsf{T}) & \text{if } (\nu\rhd\,.\,\overline{w\rhd}.\mathsf{P}') \in \mathsf{P} \text{ and } \nu\alpha \in \mathcal{S}(\mathsf{T}) \quad \text{(A1)} \\
(\mathsf{P}|\mathsf{C},\mathsf{T}) & \rightarrow & (\mathsf{P}'|\mathsf{C},\mathsf{T}) & \text{if } (\nu.\mathsf{P}') \in \mathsf{P} \text{ and } \nu \in \mathcal{S}(\mathsf{T}) \quad \text{(A2)} \\
(\overline{w}.\mathsf{P}|\mathsf{C},\mathsf{T}) & \rightarrow & (\mathsf{P}|\mathsf{C},\mathsf{T}\cup\{w\}) & \quad \text{(A3)} \\
(\mathsf{P}|\mathsf{C},\mathsf{T}) & \rightarrow & (\mathsf{P}'|\mathsf{C},\mathsf{T}\cup\{w\}) & \text{if } (\overline{w}.\mathsf{P}') \in \mathsf{P} \quad \text{(A4)}
\end{array}
$$

We show that this semantics can be internalized in our calculus within our existing semantics.

6.4.1 Intruder implementation

Let Pr be an initial configuration of a given protocol Δ with its (finite) set of *protocol keys* $\mathsf{K_p}$. Without loss of generality we may assume that Δ is *input-guarded*, meaning that it contains no input prefixes of the form $\rhd$ (i.e., every input prefix is guarded by at least one encryption key; this is easily guaranteed by replacing every summand of the form $\rhd.\,\overline{w\rhd}.\mathsf{P_i}$ with $\sum_{\mathsf{k}\in\mathsf{K_p}} \mathsf{k}\rhd\,.\,\overline{w\mathsf{k}\rhd}.\mathsf{P_i}$, which can be easily seen to generate the same labelled transition system).

Given a protocol configuration Pr, we shall define the intruder as an extra parallel component in the configuration (and hence using the same syntax). The intruder will remember a set $\mathsf{K_c} \subseteq \mathsf{K_p}$ of the compromised keys (as defined above) plus he will use an extra set of keys $\{\ell_\mathsf{P}, \ell_\mathsf{B}, \ell_\mathsf{T}\}$ of so called *locking keys* and another key # called a *separation key*.

We implement the intruder as an abstract set of operations that can modify two main data structures, a *pool* and a *buffer*. Note that this is a conceptual abstraction: in fact the set of all data is represented as a single message containing the total sequence of keys of messages that have been encountered. For this the structure pool will store all available messages separated by the key #, and the buffer structure can be either empty or it can store a single message in order to enable its analysis and synthesis. Moreover (for technical reasons) all messages placed into the pool will be in the reversed order of encryption keys.

All communication between the intruder and the protocol Pr will pass through the buffer. The available operations are in Table 6.1. Note that all

Operation	Direction	Process constant
Listen	Protocol $\rightarrow$ Buffer	$\mathsf{L}^{\mathsf{K_c}}$
Output	Buffer $\rightarrow$ Protocol	$\mathsf{O}^{\mathsf{K_c}}$
Deposit	Buffer $\rightarrow$ Pool	$\mathsf{D}^{\mathsf{K_c}}$
Retrieve	Pool $\rightarrow$ Buffer	$\mathsf{R}^{\mathsf{K_c}}$
Analyze	Buffer $\rightarrow$ Buffer	$\mathsf{A}^{\mathsf{K_c}}$
Synthesize	Buffer $\rightarrow$ Buffer	$\mathsf{S}^{\mathsf{K_c}}$
Switch	no data operation	$\mathsf{X}^{\mathsf{K_c}}$

Table 6.1: Abstract operations of the intruder

the corresponding process constants are parametrized by the set $\mathsf{K_c}$ of currently discovered compromised keys. The intuition is that *listen* can transfer any message offered by the protocol Pr and store it to the buffer. Similarly *output* can

offer the content of the buffer for communication with the original protocol Pr. *Deposit* can transfer the message stored in buffer into the pool (the extra key # is used to separate the messages stored in the pool and the transfered message is stored into P in the reversed order). *Retrieve* can nondeterministically select a single message from the pool and copy it into the buffer. *Analyze* can provide an analysis of the current message in the buffer according to the set of compromised keys $\mathsf{K_c}$ (during this a new compromised key can be discovered and the set $\mathsf{K_c}$ is updated accordingly). *Synthesize* can add an arbitrarily long sequence of compromised keys onto the message currently stored in the buffer. Finally, the *switch* process constant $\mathsf{X}^{\mathsf{K_c}}$ allows for a nondeterministic selection of any one of the previously mentioned operations. In any configuration of the intruder, exactly one of the process constants from Table 6.1 is present as a parallel component.

All internal communication within the intruder uses one of the locking keys mentioned above as the outermost key in the communicated messages. Together with our assumption that the original protocol is input-guarded by the keys $\mathsf{K_p}$, this guarantees that none of those messages are available for a communication with the protocol as they have to be unlocked first. Also observe that because the process constants from Table 6.1 are parametrized by $\mathsf{K_c} \subseteq \mathsf{K_p}$, we have exponentially many of them with respect to the size of the set $\mathsf{K_p}$. This is for technical convenience only, and it is discussed in more detail in Conclusion.

Let us now define all the components of the intruder.

6.4.2 Buffer implementation

All operations of the intruder use their own private *buffer*. A buffer B_ℓ is a process which can temporarily store any message encrypted with the locking key ℓ.

$$\mathsf{B}_\ell \stackrel{\text{def}}{=} \ell\triangleright\,.\,\overline{\ell\triangleright}.\mathsf{B}_\ell \tag{6.5}$$

The intruder uses three such buffers. We assign them the names P (pool), B (buffer) and $\mathsf{B_T}$ (temporary buffer) as follows.

$$\mathsf{P} = \mathsf{B}_{\ell_\mathsf{P}} \qquad \mathsf{B} = \mathsf{B}_{\ell_\mathsf{B}} \qquad \mathsf{B_T} = \mathsf{B}_{\ell_\mathsf{T}}$$

6.4.3 Initial configuration

The *initial intruder* with a given set of compromised keys $\mathsf{K_c}$ is described by the following configuration.

$$\mathrm{I}_{init} = \overline{\ell_\mathsf{P}\#}.\mathsf{P}|\overline{\ell_\mathsf{B}}.\mathsf{B}|\overline{\ell_\mathsf{T}}.\mathsf{B_T}|\mathsf{X}^{\mathsf{K_c}} \tag{6.6}$$

The prefix $\overline{\ell_\mathsf{P}\#}$ ensures that the empty message ϵ is available on the pool initially, and this will make it possible to output any compromised key k by using the synthesis operation on ϵ to obtain $\mathsf{k}\epsilon = \mathsf{k}$. The other two buffers are simply initialized to empty by transmitting only the corresponding locking key.

An *intruder configuration* is any configuration I such that $\mathsf{I}_{init}\rightarrow^*\mathsf{I}$. The behaviour of the intruder will be described in such a way that any intruder configuration will always be of the form

$$\mathsf{C_P}|\mathsf{C_B}|\mathsf{C_{B_T}}|\mathsf{C_X}$$

where $\mathsf{C_P}$ is either P or $\overline{\ell_\mathsf{P} w}.\mathsf{P}$ for some $w \in (\mathsf{K_p}\cup\{\#\})^*$, $\mathsf{C_B}$ is either B or $\overline{\ell_\mathsf{B} w}.\mathsf{B}$ for some $w \in \mathsf{K_p}^*$, $\mathsf{C_{B_T}}$ is either $\mathsf{B_T}$ or $\overline{\ell_\mathsf{T} w}.\mathsf{B_T}$ for some $w \in (\mathsf{K_p}\cup\{\#\})^*$, and $\mathsf{C_X}$ is either Y or $\overline{w}.\mathsf{Y}$ where $\mathsf{Y} \in \{\mathsf{X}^{\mathsf{K_c}}, \mathsf{L}^{\mathsf{K_c}}, \mathsf{O}^{\mathsf{K_c}}, \mathsf{D}^{\mathsf{K_c}}, \mathsf{R}^{\mathsf{K_c}}, \mathsf{A}^{\mathsf{K_c}}, \mathsf{S}^{\mathsf{K_c}}\}$ and $w \in \mathcal{K}^*$.

As the original protocol is assumed to be input-guarded, in a configuration like $Pr|\mathsf{I}$ (where Pr is a protocol configuration and I is an intruder configuration), we will make sure that only a process constant Y in the $\mathsf{C_X}$ part of I can manipulate the available buffers in parts $\mathsf{C_P}$, $\mathsf{C_B}$ and $\mathsf{C_{B_T}}$.

6.4.4 Switching between operations

The intruder is always capable of choosing nondeterministically between any of the six operations in Table 6.1. In this way, the intruder can manipulate messages by any means necessary. This is expressed by the central component of the intruder, the switch process constant $\mathsf{X}^{\mathsf{K_c}}$, which is parametrized by the set of currently compromised keys $\mathsf{K_c}$.

$$\begin{aligned}
\mathsf{X}^{\mathsf{K_c}} \stackrel{\text{def}}{=}\ & \ell_\mathsf{B}.\mathsf{L}^{\mathsf{K_c}} && (6.7)\\
& + \ell_\mathsf{P}\triangleright\,.\,\overline{\ell_\mathsf{P}\triangleright}.\mathsf{O}^{\mathsf{K_c}} && (6.8)\\
& + \ell_\mathsf{P}\triangleright\,.\,\overline{\ell_\mathsf{P}\triangleright}.\mathsf{D}^{\mathsf{K_c}} && (6.9)\\
& + \ell_\mathsf{B}.\mathsf{R}^{\mathsf{K_c}} && (6.10)\\
& + \ell_\mathsf{P}\triangleright\,.\,\overline{\ell_\mathsf{P}\triangleright}.\mathsf{A}^{\mathsf{K_c}} && (6.11)\\
& + \ell_\mathsf{P}\triangleright\,.\,\overline{\ell_\mathsf{P}\triangleright}.\mathsf{S}^{\mathsf{K_c}} && (6.12)
\end{aligned}$$

As $\mathsf{X}^{\mathsf{K_c}}$ is always in parallel with a pool of the form $\overline{\ell_\mathsf{P} w}.\mathsf{P}$, the following communication is possible $\overline{\ell_\mathsf{P} w}.\mathsf{P}|\mathsf{X}^{\mathsf{K_c}} \xrightarrow{\ell_\mathsf{P} w} \mathsf{P}|\overline{\ell_\mathsf{P} w}.\mathsf{Y} \xrightarrow{\ell_\mathsf{P} w} \overline{\ell_\mathsf{P} w}.\mathsf{P}|\mathsf{Y}$ for any $\mathsf{Y} \in \{\mathsf{O}^{\mathsf{K_c}}, \mathsf{D}^{\mathsf{K_c}}, \mathsf{A}^{\mathsf{K_c}}, \mathsf{S}^{\mathsf{K_c}}\}$. This corresponds to giving a control to the appropriate process constant Y (the use of pool to make the switch is here only for technical reasons and the pool content is untouched). The process constants $\mathsf{L}^{\mathsf{K_c}}$ and $\mathsf{R}^{\mathsf{K_c}}$ use a different switching method which guarantees that the state change is possible only if the buffer B is empty (i.e., it outputs only the key ℓ_B).

6.4.5 Listen to the protocol

As the intruder runs in parallel with the protocol configuration Pr, it may collect any message transmitted by Pr. It then encrypts the message by the locking key ℓ_B and stores the message into the buffer B. The control is returned to $\mathsf{X}^{\mathsf{K_c}}$ afterwards.

$$\mathsf{L}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \sum_{\mathsf{k}\in\mathsf{K_p}} \mathsf{k}\triangleright\,.\,\overline{\ell_\mathsf{B}\mathsf{k}\triangleright}.\mathsf{X}^{\mathsf{K_c}} \qquad (6.13)$$

Formally, if $\mathsf{m} \in \mathsf{K_p}^*$ and Pr can transmit m and become Pr', we can perform the following communication.

$$Pr|\mathsf{B}|\mathsf{L}^{\mathsf{K_c}} \xrightarrow{\mathsf{m}} Pr'|\mathsf{B}|\overline{\ell_{\mathsf{B}}\mathsf{m}}.\mathsf{X}^{\mathsf{K_c}} \xrightarrow{\ell_{\mathsf{B}}\,\mathsf{m}} Pr'|\overline{\ell_{\mathsf{B}}\mathsf{m}}.\mathsf{B}|\mathsf{X}^{\mathsf{K_c}}$$

6.4.6 Output a message to the protocol

The intruder may output any message stored in the buffer B by removing its locking key ℓ_{B}, thereby making the message available to the protocol.

$$\mathsf{O}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \ell_{\mathsf{B}}\rhd\,.\,\overline{\rhd}.\mathsf{O'}^{\mathsf{K_c}} \tag{6.14}$$

$$\mathsf{O'}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \overline{\mathsf{l_B}}.\mathsf{X}^{\mathsf{K_c}} \tag{6.15}$$

Hence if Pr can receive a message $\mathsf{m} \in \mathsf{K_p}^*$ and become Pr' then we get the following communication.

$$Pr|\overline{\ell_{\mathsf{B}}\mathsf{m}}.\mathsf{B}|\mathsf{O}^{\mathsf{K_c}} \xrightarrow{\ell_{\mathsf{B}}\,\mathsf{m}} Pr|\mathsf{B}|\overline{\mathsf{m}}.\mathsf{O'}^{\mathsf{K_c}} \xrightarrow{\mathsf{m}}$$

$$Pr'|\mathsf{B}|\mathsf{O'}^{\mathsf{K_c}} \xrightarrow{\ell_{\mathsf{B}}} Pr'|\overline{\ell_{\mathsf{B}}}.\mathsf{B}|\mathsf{X}^{\mathsf{K_c}}$$

6.4.7 Analysis and synthesis

Assume that the buffer B contains some message of the form $\overline{\ell_{\mathsf{B}}w}.\mathsf{B}$ where $w \in \mathsf{K_p}^*$. Computing an element of the analysis amounts to step-by-step decryption with any of the compromised keys from $\mathsf{K_c}$.

$$\mathsf{A}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \sum_{\mathsf{k}\in\mathsf{K_c}} \ell_{\mathsf{B}}\mathsf{k}\rhd\,.\,\overline{\ell_{\mathsf{B}}\rhd}.\mathsf{A}^{\mathsf{K_c}} + \sum_{\mathsf{k}\in\mathsf{K_p}} \ell_{\mathsf{B}}\mathsf{k}.\mathsf{A}1_{\mathsf{k}}^{\mathsf{K_c}} + \ell_{\mathsf{P}}\rhd\,.\,\overline{\ell_{\mathsf{P}}\rhd}.\mathsf{X}^{\mathsf{K_c}} \tag{6.16}$$

$$\mathsf{A}1_{\mathsf{k}}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \overline{\ell_{\mathsf{B}}}.\mathsf{X}^{\mathsf{K_c}\cup\{\mathsf{k}\}} \quad \text{for all } \mathsf{k} \in \mathsf{K_p} \tag{6.17}$$

The second summand in equation (6.16) corresponds to the discovery of a plain-text key, which is promptly added to the set of compromised keys by equation (6.17), the buffer B is initialized to $\overline{\ell_{\mathsf{B}}}.\mathsf{B}$ and the control is given to $\mathsf{X}^{\mathsf{K_c}\cup\{\mathsf{k}\}}$. The third summand in (6.16) allows the intruder to end the analysis phase and return the control to the switch $\mathsf{X}^{\mathsf{K_c}}$.

Computing an element of the synthesis simply enables to add (step-by-step) as many compromised keys as the intruder wants to the message stored in the buffer B.

$$\mathsf{S}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \sum_{\mathsf{k}\in\mathsf{K_c}} \ell_{\mathsf{B}}\rhd\,.\,\overline{\ell_{\mathsf{B}}\mathsf{k}\rhd}.\mathsf{S}^{\mathsf{K_c}} + \ell_{\mathsf{P}}\rhd\,.\,\overline{\ell_{\mathsf{P}}\rhd}.\mathsf{X}^{\mathsf{K_c}} \tag{6.18}$$

6.4.8 Deposit

This operation deposits a message from the buffer B into the pool P by moving it key by key.

$$\mathsf{D}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \sum_{\mathsf{k}\in\mathsf{K_p}} \ell_{\mathsf{B}}\mathsf{k}\triangleright\,.\,\overline{\ell_{\mathsf{B}}\triangleright}.\mathsf{D}_{\mathsf{k}}^{\mathsf{K_c}} + \ell_{\mathsf{B}}.\mathsf{D'}_{\#}^{\mathsf{K_c}} \tag{6.19}$$

$$\mathsf{D}_{\mathsf{k}}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \ell_{\mathsf{P}}\triangleright\,.\,\overline{\ell_{\mathsf{P}}\mathsf{k}\triangleright}.\mathsf{D}^{\mathsf{K_c}} \quad \text{for all } \mathsf{k}\in\mathsf{K_p} \tag{6.20}$$

$$\mathsf{D'}_{\#}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \overline{\ell_{\mathsf{B}}}.\mathsf{D}_{\#}^{\mathsf{K_c}} \tag{6.21}$$

$$\mathsf{D}_{\#}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \ell_{\mathsf{P}}\triangleright\,.\,\overline{\ell_{\mathsf{P}}\#\triangleright}.\mathsf{X}^{\mathsf{K_c}} \tag{6.22}$$

The equations (6.19) and (6.20) perform the key transfer from B to P. One step of such a transfer looks as follows (here $\mathsf{k}\in\mathsf{K_p}$ and w_1 and w_2 are arbitrary messages).

$$\overline{\ell_{\mathsf{P}}w_1}.\mathsf{P}|\overline{\ell_{\mathsf{B}}\mathsf{k}w_2}.\mathsf{B}|\mathsf{D}^{\mathsf{K_c}} \xrightarrow{\ell_{\mathsf{B}}\mathsf{k}w_2} \overline{\ell_{\mathsf{P}}w_1}.\mathsf{P}|\mathsf{B}|\overline{\ell_{\mathsf{B}}w_2}.\mathsf{D}_{\mathsf{k}}^{\mathsf{K_c}} \xrightarrow{\ell_{\mathsf{B}}w_2}$$

$$\overline{\ell_{\mathsf{P}}w_1}.\mathsf{P}|\overline{\ell_{\mathsf{B}}w_2}.\mathsf{B}|\mathsf{D}_{\mathsf{k}}^{\mathsf{K_c}} \xrightarrow{\ell_{\mathsf{P}}w_1} \mathsf{P}|\overline{\ell_{\mathsf{B}}w_2}.\mathsf{B}|\overline{\ell_{\mathsf{P}}\mathsf{k}w_1}.\mathsf{D}^{\mathsf{K_c}} \xrightarrow{\ell_{\mathsf{P}}\mathsf{k}w_1}$$

$$\overline{\ell_{\mathsf{P}}\mathsf{k}w_1}.\mathsf{P}|\overline{\ell_{\mathsf{B}}w_2}.\mathsf{B}|\mathsf{D}^{\mathsf{K_c}}$$

Recall that as indicated before, the messages are stored in reverse order in P. When the bottom of the buffer B is reached, the second summand in equation (6.19) can be applied. First the key ℓ_{B} is returned to the buffer B by equation (6.21) and finally the control is given to the process constant $\mathsf{X}^{\mathsf{K_c}}$ while adding the separation key # into the pool by equation (6.22).

6.4.9 Retrieve

To retrieve a message from the pool (the most complicated operation of the construction), the retrieval process scans through the message separation markers # until it nondeterministically decides to copy a message to the buffer (here it is also the only place where the temporary buffer is used).

$$\mathsf{R}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \overline{\ell_{\mathsf{B}}}.{\mathsf{R}_1}^{\mathsf{K_c}} \tag{6.23}$$

$${\mathsf{R}_1}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \sum_{\mathsf{k}\in\mathsf{K_p}\cup\{\#\}} \ell_{\mathsf{P}}\mathsf{k}\triangleright\,.\,\overline{\ell_{\mathsf{P}}\triangleright}.\mathsf{R}_{\mathsf{k}}^{\mathsf{K_c}} + \ell_{\mathsf{P}}\#\triangleright\,.\,\overline{\ell_{\mathsf{P}}\triangleright}.\mathsf{S}_{\#}^{\mathsf{K_c}} + \ell_{\mathsf{P}}.{\mathsf{T}_2'}^{\mathsf{K_c}} \tag{6.24}$$

$$\mathsf{R}_{\mathsf{k}}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \ell_{\mathsf{T}}\triangleright\,.\,\overline{\ell_{\mathsf{T}}\mathsf{k}\triangleright}.{\mathsf{R}_1}^{\mathsf{K_c}} \quad \text{for all } \mathsf{k}\in\mathsf{K_p}\cup\{\#\} \tag{6.25}$$

$$\mathsf{S}_{\#}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \ell_{\mathsf{T}}\triangleright\,.\,\overline{\ell_{\mathsf{T}}\#\triangleright}.\mathsf{T}^{\mathsf{K_c}} \tag{6.26}$$

$$\mathsf{T}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \sum_{\mathsf{k}\in\mathsf{K_p}} \ell_{\mathsf{P}}\mathsf{k}\triangleright\,.\,\overline{\ell_{\mathsf{P}}\triangleright}.\mathsf{T}_{\mathsf{k}}^{\mathsf{K_c}} + \ell_{\mathsf{P}}\#\triangleright\,.\,\overline{\ell_{\mathsf{P}}\#\triangleright}.{\mathsf{T}_2}^{\mathsf{K_c}} + \ell_{\mathsf{P}}.{\mathsf{T}_2'}^{\mathsf{K_c}} \tag{6.27}$$

$$\mathsf{T}_{\mathsf{k}}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \ell_{\mathsf{T}}\triangleright\,.\,\overline{\ell_{\mathsf{T}}\mathsf{k}\triangleright}.{\mathsf{T}_1}_{\mathsf{k}}^{\mathsf{K_c}} \quad \text{for all } \mathsf{k}\in\mathsf{K_p} \tag{6.28}$$

$${\mathsf{T}_1}_{\mathsf{k}}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \ell_{\mathsf{B}}\triangleright\,.\,\overline{\ell_{\mathsf{B}}\mathsf{k}\triangleright}.\mathsf{T}^{\mathsf{K_c}} \quad \text{for all } \mathsf{k}\in\mathsf{K_p} \tag{6.29}$$

$${\mathsf{T}_2'}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \overline{\ell_{\mathsf{P}}}.{\mathsf{T}_2}^{\mathsf{K_c}} \tag{6.30}$$

$${\mathsf{T}_2}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \sum_{\mathsf{k}\in\mathsf{K_p}\cup\{\#\}} \ell_{\mathsf{T}}\mathsf{k}\triangleright\,.\,\overline{\ell_{\mathsf{T}}\triangleright}.{\mathsf{T}_3}_{\mathsf{k}}^{\mathsf{K_c}} + \ell_{\mathsf{T}}.{\mathsf{T}_4}^{\mathsf{K_c}} \tag{6.31}$$

$${\mathsf{T}_3}_{\mathsf{k}}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \ell_{\mathsf{P}}\triangleright\,.\,\overline{\ell_{\mathsf{P}}\mathsf{k}\triangleright}.{\mathsf{T}_2}^{\mathsf{K_c}} \quad \text{for all } \mathsf{k}\in\mathsf{K_p}\cup\{\#\} \tag{6.32}$$

$${\mathsf{T}_4}^{\mathsf{K_c}} \stackrel{\text{def}}{=} \overline{\ell_{\mathsf{T}}}.\mathsf{X}^{\mathsf{K_c}} \tag{6.33}$$

First the buffer B is initialized with the key ℓ_{B} by equation (6.23). Then by equations (6.24) and (6.25) all keys from $\mathsf{K_p}$ can be moved one by one to the temporary buffer $\mathsf{B_T}$. In case that the #-key is discovered during the process, the intruder can either continue as before by placing the key into the temporary buffer (using the first summand in (6.24)), or he can decide that # is the start of the selected message to be copied into the buffer B and use the second summand of (6.24).

In the latter case equation (6.26) places the # key into the temporary buffer $\mathsf{B_T}$ and evokes the process constant $\mathsf{T}^{\mathsf{K_c}}$ which copies the next piece of the message from the pool both into the temporary buffer $\mathsf{B_T}$ and also into the buffer B. This is done using the equations (6.28) and (6.29) and the control returns to $\mathsf{T}^{\mathsf{K_c}}$. Whenever the first separation key # appears as the outermost key, the second summand in (6.27) applies and the intruder continues from ${\mathsf{T}_2}^{\mathsf{K_c}}$ in equation (6.31). The same happens when the end of the pool is reached (third summand of equation (6.27)). In this case the key ℓ_{P} is first return into the pool by the process constant ${\mathsf{T}_2'}^{\mathsf{K_c}}$ and then the process continues by equation (6.31) as before.

The remaining equations (6.31) – (6.33) simply return the content of the temporary buffer $\mathsf{B_T}$ back to the pool and give the control to $\mathsf{X}^{\mathsf{K_c}}$ (making sure that the temporary buffer is reset to $\overline{\ell_{\mathsf{T}}}.\mathsf{B_T}$).

6.4.10 Correctness of the encoding

We now show that the intruder faithfully describes the environment sensitive semantics defined in the beginning of Section 6.4 by axioms (A1) – (A4).

To do this, we need to define the knowledge of the intruder. We take this to be the totality of the set of messages that he can output to the environment.

Let $\mathsf{E} \stackrel{\text{def}}{=} \sum_{\mathsf{k}\in\mathsf{K_p}} \mathsf{k}\rhd\,.\,\overline{\mathsf{k}\rhd}.\mathsf{E}$ be an observer (environment) which allows us to observe all messages encrypted by keys from $\mathsf{K_p}$. Let $\Longrightarrow$ be an (unlabelled) reflexive and transitive closure of transition relations labelled by actions that are not available to the environment (internal communication of the intruder), i.e.,

$$\Longrightarrow \stackrel{\text{def}}{=} \Big(\bigcup_{\mathsf{m}\in(\mathcal{K}\setminus\mathsf{K_p})\cdot\mathcal{K}^*} \stackrel{\mathsf{m}}{\longrightarrow}\Big)^*.$$

The set of *known messages* of an intruder configuration I is defined by

$$km(\mathrm{I}) \stackrel{\text{def}}{=} \{\mathsf{w}\in {\mathsf{K_p}}^* \mid \mathsf{E}|\mathrm{I} \Longrightarrow \circ \stackrel{\mathsf{w}}{\longrightarrow}\}.$$

We say that I is a *proper* intruder configuration whenever I does not contain any parallel component of the form $\mathsf{L}^{\mathsf{K_c}}$, $\mathsf{O}^{\mathsf{K_c}}$ or $\overline{\mathsf{m}}.{\mathsf{O}'}^{\mathsf{K_c}}$. In other words it is not directly committed to communicate with the environment.

We now show an operational correspondence result which states that the intruder, as described above, accurately describes the behaviour of the environment sensitive semantics expressed by axioms (A1) – (A4). The following lemma shows that the intruder can synthesize all possible messages.

Lemma 6.4 *For any proper intruder configuration* I *it holds that* $\mathrm{km}(\mathrm{I}) = \mathcal{S}(\mathrm{km}(\mathrm{I}))$.

Proof. The inclusion $km(\mathrm{I}) \subseteq \mathcal{S}(km(\mathrm{I}))$ follows from the definition of synthesis. Let $\mathsf{w} \in \mathcal{S}(km(\mathrm{I}))$. We will show that $\mathsf{w} \in km(\mathrm{I})$. Because of Remark 6.4 we know that the set of compromised keys $\mathsf{K_c}$ (where $\mathsf{n} = |\mathsf{K_c}|$) can be computed by analyzing of certain messages $\mathsf{m_1}, \ldots, \mathsf{m_n} \in km(\mathrm{I})$. As these messages belong to $km(\mathrm{I})$ they can consequently be placed (in the given order) into the buffer B and analyzed. This will update the current set of compromised keys remembered by the intruder to the complete set $\mathsf{K_c}$. Because of Proposition 6.1 we know that w can be constructed by first placing a selected message into the buffer and by removing all the compromised keys (by $\mathsf{A}^{\mathsf{K_c}}$) and then by adding the compromised keys in order to form w (by $\mathsf{S}^{\mathsf{K_c}}$). The message w can then be communicated to the environment E by $\mathsf{O}^{\mathsf{K_c}}$ and hence $\mathsf{w} \in km(\mathrm{I})$. □

The first theorem states that the intruder can mimic the behaviour of the environment-sensitive semantics.

Theorem 6.3 (Completeness) *Let* C *be a protocol configuration, let* $\mathsf{T} \subseteq \mathcal{K}^*$, *and let* I *be a proper intruder configuration such that* $\mathcal{S}(\mathsf{T}) = \mathrm{km}(\mathrm{I})$. *If*

$$(\mathsf{C}, \mathsf{T}) \to (\mathsf{C}', \mathsf{T}')$$

then there exists $\mathsf{w} \in \mathsf{K_p}^*$ *and a transition sequence*

$$(\mathsf{C}|\mathrm{I}) \Longrightarrow \circ \stackrel{\mathsf{w}}{\longrightarrow} (\mathsf{C}'|\mathrm{I}')$$

such that $\mathcal{S}(\mathsf{T}') = \mathrm{km}(\mathrm{I}')$ *and* I′ *is a proper intruder configuration.*

Proof. The intruder's strategy is clear. He listens to every message communicated by the protocol and stores all such messages into the pool in order to be able to synthesize the appropriate messages and communicate them back to the protocol at some later stage.

First observe that there are four possible axioms (A1) – (A4) to perform $(\mathsf{C},\mathsf{T}) \to (\mathsf{C}',\mathsf{T}')$. In case of the axioms (A1) and (A2) we know that the protocol configuration C receives a message w from $\mathcal{S}(\mathsf{T})$ and becomes C', and that $\mathsf{T}' = \mathsf{T}$. Because $w \in \mathcal{S}(\mathsf{T}) = km(\mathrm{I})$ we know that the intruder I can perform $\mathrm{I} \Longrightarrow \mathrm{I}''$ such that $\mathsf{E}|\mathrm{I}'' \xrightarrow{w} \mathsf{E}'|\mathrm{I}'$ and hence also $(\mathsf{C}|\mathrm{I}) \Longrightarrow \circ \xrightarrow{w} (\mathsf{C}'|\mathrm{I}')$. Moreover, $km(\mathrm{I}') = km(\mathrm{I})$ as required because every message that appears in the buffer B can be also copied to the pool P for a use later on. It is also easy to see that I' is proper.

In case of the axioms (A3) and (A4) the protocol configuration C emits a message w and becomes C', and $\mathsf{T}' = \mathsf{T} \cup \{w\}$. In this case the intruder deposits the content of the buffer into the pool (in case that it was not empty) and then receives the message w from C. This means that $(\mathsf{C}|\mathrm{I}) \Longrightarrow \circ \xrightarrow{w} (\mathsf{C}'|\mathrm{I}')$ and moreover $km(\mathrm{I}')$ now contains the message w (obviously I' is proper). This implies that $\mathcal{S}(\mathsf{T}') = \mathcal{S}(\mathsf{T} \cup \{w\}) = km(\mathrm{I}')$ because of Lemma 6.4. □

Conversely, the intruder cannot gain any extra knowledge than the one he can construct by analysis and synthesis of the available messages.

Theorem 6.4 (Soundness) *Let* C *be a protocol configuration, let* $\mathsf{T} \subseteq \mathcal{K}^*$, *and let* I *be an intruder configuration such that* $\mathrm{km}(\mathrm{I}) \subseteq \mathcal{S}(\mathsf{T})$. *If for some* $w \in \mathsf{K}_{\mathsf{p}}^*$

$$(\mathsf{C}|\mathrm{I}) \Longrightarrow \circ \xrightarrow{w} (\mathsf{C}'|\mathrm{I}')$$

then there exists a transition

$$(\mathsf{C},\mathsf{T}) \to (\mathsf{C}',\mathsf{T}') \qquad \textit{or} \qquad (\mathsf{C},\mathsf{T}) \to \circ \to (\mathsf{C}',\mathsf{T}')$$

such that $\mathrm{km}(\mathrm{I}') \subseteq \mathcal{S}(\mathsf{T}')$.

Proof. There are two cases to distinguish. In the first case the intruder either listens to the message w sent by C, or he outputs w for a communication with the protocol C. This means that there is a corresponding transition also in $(\mathsf{C},\mathsf{T}) \to (\mathsf{C}',\mathsf{T}')$ and it is easy to verify that $km(\mathrm{I}') \subseteq \mathcal{S}(\mathsf{T}')$. In the second case the intruder does not interfere with the communication in C (hence $km(\mathrm{I}') = km(\mathrm{I})$) and such a communication step is simulated by two steps $(\mathsf{C},\mathsf{T}) \to \circ \to (\mathsf{C}',\mathsf{T}')$ (in the environment sensitive semantics all messages must pass through T). By the fact that $\mathcal{S}(\mathsf{T}) \subseteq \mathcal{S}(\mathsf{T}')$, we can immediately see that $km(\mathrm{I}') \subseteq \mathcal{S}(\mathsf{T}')$. □

6.5 Replicative ping-pong protocols

In this section we shall define a replicative variant of our calculus for ping-pong protocols. We will then show that this formalism is not Turing powerful because

the reachability problem becomes decidable. This is to be put in contrast with several results where replicative calculi are known to be capable of simulating recursive ones (see e.g. [17] for the case of pi-calculus which implies also the same result for spi-calculus, and [13, 18] for other examples). On the other hand, a similar discrimination result as ours between recursion and replication was recently proved for CCS [7].

Let us now define *replicative ping-pong protocols*. Let $\mathcal{K}$ be the set of encryption keys as before. The set $\mathcal{C}onf$ of protocol configurations is given by the following abstract syntax

$$\mathsf{C} \ ::= \ \mathbf{0} \ \mid \ \nu\triangleright\,.\,\overline{w\triangleright} \ \mid \ \nu \ \mid \ \overline{w} \ \mid \ !(\nu\triangleright\,.\,\overline{w\triangleright}) \ \mid \ !(\nu) \ \mid \ !(\overline{w}) \ \mid \ \mathsf{C}|\mathsf{C}$$

where $\mathbf{0}$ is the symbol for the empty configuration, ν and w range over $\mathcal{K}^*$, and $!$ is the bang operator (replication). As before, we shall introduce structural congruence $\equiv$, which is the smallest congruence over $\mathcal{C}onf$ such that

- $(\mathcal{C}onf, |, \mathbf{0})$ is a commutative monoid
- $\epsilon|\mathsf{C} \equiv \mathsf{C} \equiv \ \overline{\epsilon}|\mathsf{C}$
- $!(\epsilon) \equiv \mathbf{0} \equiv \ !(\overline{\epsilon})$
- $!(\mathsf{C}) \equiv \mathsf{C}|!(\mathsf{C})$.

A labelled transition system determined by a configuration (where states are configurations modulo $\equiv$ and labels are non-empty messages as before) is defined by the following SOS rules (recall the replicative axiom $!(\mathsf{C}) \equiv \mathsf{C}|!(\mathsf{C})$ and the fact that '$|$' is commutative).

$$\frac{\mathfrak{m} \in \mathcal{K}^+}{\overline{\mathfrak{m}}|\mathfrak{m}|\mathsf{C} \ \stackrel{\mathfrak{m}}{\longrightarrow} \ \mathsf{C}} \qquad \frac{\mathfrak{m} \in \mathcal{K}^+ \qquad \mathfrak{m} = \nu\alpha}{\overline{\mathfrak{m}}|\nu\triangleright\,.\,\overline{w\triangleright}|\mathsf{C} \ \stackrel{\mathfrak{m}}{\longrightarrow} \ \overline{w\alpha}|\mathsf{C}}$$

Example 6.3 *An initial configuration* $\mathsf{C} \stackrel{\text{def}}{=} !(\overline{\mathsf{k}}) \mid !(\mathsf{k}\triangleright\,.\,\overline{\mathsf{k}\mathsf{k}\triangleright})$ *generates a labelled transition system in Figure 6.2 with infinitely many reachable configurations (only a finite fragment is depicted). Observe that (unlike recursive protocols) the number of parallel components can grow arbitrarily (e.g. the left-most branch in the picture). The reason is that we allow prefixes of the form* $!(\overline{w})$ *which can unconditionally output new messages and that we have replicative agents accepting these messages.*

In case that the number of output prefixes for all reachable configurations is bounded by some number $\mathfrak{n}$*, the parallel components of the form* $!(\nu)$*,* $!(\overline{w})$ *and* $!(\nu\triangleright\,.\,\overline{w\triangleright})$ *can be replaced by* $\mathfrak{n}$ *parallel occurrences of fresh process constants* $\mathsf{P}_{!(\nu)}$*,* $\mathsf{P}_{!(\overline{w})}$ *and* $\mathsf{P}_{!(\nu\triangleright\,.\,\overline{w\triangleright})}$ *respectively such that* $\mathsf{P}_{!(\nu)} \stackrel{\text{def}}{=} \nu.\mathsf{P}_{!(\nu)}$*,* $\mathsf{P}_{!(\overline{w})} \stackrel{\text{def}}{=} \overline{w}.\mathsf{P}_{!(\overline{w})}$ *and* $\mathsf{P}_{!(\nu\triangleright\,.\,\overline{w\triangleright})} \stackrel{\text{def}}{=} \nu\triangleright\,.\,\overline{w\triangleright}.\mathsf{P}_{!(\nu\triangleright\,.\,\overline{w\triangleright})}$*, and hence it can be simulated by recursive protocols.*

We shall now show that the reachability problem for general replicative ping-pong protocols is decidable. We reduce our problem to reachability of

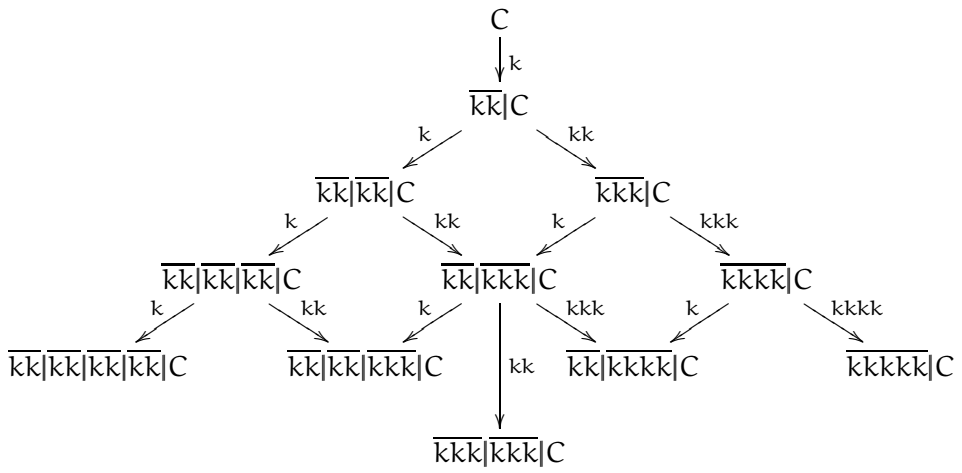


Figure 6.2: Initial fragment of a labelled transition system for $C \stackrel{\text{def}}{=} !(\overline{k}) \mid !(k\triangleright . \overline{kk\triangleright})$

weak process rewrite systems (wPRS) which was very recently proven to be decidable [15].

Following the work of Mayr [16], let $\mathcal{C}onst$ be a set of *process constants*. The set $\mathcal{E}$ of *process expressions* over $\mathcal{C}onst$ is defined by

$$E ::= \epsilon \mid X \mid E.E \mid E|E$$

where ϵ is the symbol for empty expression, X ranges over $\mathcal{C}onst$, '.' is the operator of sequential composition and '|' is the operator of parallel composition. We identify processes up to structural congruence, which is the least congruence such that '.' is associative, '|' is associative and commutative and ϵ is a unit for '.' and '|'.

Let Q be a finite set of *control-states* and $\mathcal{A}ct$ a set of *actions*. A *state-extended process rewrite system* (sePRS) is a finite set Δ of rewrite rules of the form $pE \xrightarrow{a} qF$ where $p, q \in Q$, $a \in \mathcal{A}ct$, and $E, F \in \mathcal{E}$ such that $E \neq \epsilon$.

A given sePRS Δ generates a labelled transition system $T(\Delta)$ where states are process expressions over $\mathcal{C}onst$ modulo the structural congruence, the set of actions is $\mathcal{A}ct$ and the transition relation is given by the following SOS rules (recall that '|' is commutative).

$$\frac{(pE \xrightarrow{a} qF) \in \Delta}{pE \xrightarrow{a} qF} \qquad \frac{pE \xrightarrow{a} qE'}{p(E.F) \xrightarrow{a} q(E'.F)} \qquad \frac{pE \xrightarrow{a} qE'}{p(E|F) \xrightarrow{a} q(E'|F)}$$

A sePRS Δ is called a *weak process rewrite system* (wPRS) iff there is a partial ordering $\leq$ on Q such that all rewrite rules $pE \xrightarrow{a} qF$ from Δ satisfy that $q \leq p$.

Theorem 6.5 ([15]) *The reachability problem for wPRS is decidable.*

Let us now consider an arbitrary configuration C in the calculus of replicative ping-pong protocols. We shall construct a wPRS system Δ which preserves the reachability property.

The configuration C can be naturally written as $\mathsf{C} \equiv \mathsf{A}|\mathsf{B}|\mathsf{O}$ where A contains all parallel components of the form $!(\nu\triangleright\,.\,\overline{w\triangleright})$, $!(\nu)$ and $!(\overline{w})$; B contains all parallel components of the form $\nu\triangleright\,.\,\overline{w\triangleright}$ and ν; and O contains all output prefixes of the form $\overline{w}$. Here we assume that the rules of the structural congruence $\equiv$ are applied as long as possible in order to minimize the size of the configuration C (such assumptions are implicit also later on). Observe now that any configuration $\mathsf{C}' \equiv \mathsf{A}|\mathsf{B}'|\mathsf{O}'$ reachable from C contains exactly the same part A and every parallel component from B' is also in B.

The intuition of the reduction is that A does not have to be remembered as all parallel components from A are always available, B will be stored in control-states of the wPRS (note that there are only finitely many different components in B' for all reachable configurations of the form $\mathsf{A}|\mathsf{B}'|\mathsf{O}'$) and the parallel components from O will be stored as a parallel composition of stacks in the wPRS system.

Let $\mathsf{C} \equiv \mathsf{A}|\mathsf{B}|\mathsf{O}$ be the initial configuration. Formally, the wPRS rules Δ where $\mathcal{C}onst \stackrel{\text{def}}{=} \mathcal{K} \cup \{\mathsf{Z}, \mathsf{X}\}$ (Z is a special symbol for the bottom of a stack, X is a process constant for creating more parallel components) and where $\mathsf{Q} \stackrel{\text{def}}{=} \{\mathfrak{p}^{\mathsf{B}'} \mid \exists \mathsf{B}'' \text{ s.t. } \mathsf{B} \equiv \mathsf{B}'|\mathsf{B}''\}$ are defined as follows.

1.	$\mathfrak{p}^{\mathsf{B}'}\mathsf{X} \longrightarrow \mathfrak{p}^{\mathsf{B}'}(\mathsf{X}\|w.\mathsf{Z})$	$\mathsf{B}' \subseteq \mathsf{B}$ and $!(\overline{w}) \in \mathsf{A}$
2.	$\mathfrak{p}^{\mathsf{B}'}w.\mathsf{Z} \longrightarrow \mathfrak{p}^{\mathsf{B}'}$	$\mathsf{B}' \subseteq \mathsf{B}$ and $!(\overline{w}) \in \mathsf{A}$
3.	$\mathfrak{p}^{\mathsf{B}'}\mathsf{Z} \longrightarrow \mathfrak{p}^{\mathsf{B}'}$	$\mathsf{B}' \subseteq \mathsf{B}$
4.	$\mathfrak{p}^{\mathsf{B}'}\nu.\mathsf{Z} \longrightarrow \mathfrak{p}^{\mathsf{B}'}$	$\mathsf{B}' \subseteq \mathsf{B}$ and $!(\nu) \in \mathsf{A}$
5.	$\mathfrak{p}^{\mathsf{B}'}\nu \longrightarrow \mathfrak{p}^{\mathsf{B}'}w$	$\mathsf{B}' \subseteq \mathsf{B}$ and $!(\nu\triangleright\,.\,\overline{w\triangleright}) \in \mathsf{A}$
6.	$\mathfrak{p}^{\mathsf{B}'}\nu.\mathsf{Z} \longrightarrow \mathfrak{p}^{\mathsf{B}''}$	$\mathsf{B}' \subseteq \mathsf{B}$ and $\mathsf{B}' \equiv \mathsf{B}''\|\nu$
7.	$\mathfrak{p}^{\mathsf{B}'}\nu \longrightarrow \mathfrak{p}^{\mathsf{B}''}w$	$\mathsf{B}' \subseteq \mathsf{B}$ and $\mathsf{B}' \equiv \mathsf{B}''\|\nu\triangleright\,.\,\overline{w\triangleright}$

In the definitions above, whenever we have $w \in \mathcal{K}^*$, we use the word w also in the wPRS rules in the meaning that it represents a sequential composition of process constants contained in w, i.e., if $w = \mathfrak{a}_1\mathfrak{a}_2\cdots\mathfrak{a}_n$ then w in the wPRS rules stands for the sequential composition $\mathfrak{a}_1.\mathfrak{a}_2.\ldots.\mathfrak{a}_n$. Moreover $\mathsf{B}' \subseteq \mathsf{B}$ means that there is some B'' such that $\mathsf{B} \equiv \mathsf{B}'|\mathsf{B}''$ and $\mathfrak{x} \in \mathsf{A}$ means that the expression $\mathfrak{x}$ is a parallel component in A. All actions are omitted as they are irrelevant for the reachability question.

Rules 1. – 3. correspond to the structural congruence $\equiv$. As $!(\overline{w}) \equiv \overline{w}|!(\overline{w})$ rule 1. enables us to create a new parallel component $\overline{w}$ whenever $!(\overline{w}) \in \mathsf{A}$ and by rule 2. such a component can always be deleted. Rule 3. corresponds

to the fact that $\epsilon|C \equiv C$. (Recall that we assume that C does not contain any component of the form $!(\epsilon)$ or $!(\overline{\epsilon})$.)

Rules 4. – 7. are computational rules: in rules 4. and 5. we allow the reception of messages by the components in A and the control-state does not change in this case; in rules 6. and 7. we do the same for components in B' (the current remaining part of B) and whenever such a component is used, we remove it from B' by changing the control-state to $p^{B''}$.

Let us assume the initial configuration $C \equiv A|B|O$ as above such that $O \equiv \overline{w_1}|\overline{w_2}|\cdots|\overline{w_n}$. The initial configuration of the wPRS system Δ is then

$$p^B(X|w_1.Z|w_2.Z|\cdots|w_n.Z).$$

It is easy to see that every rewriting step in Δ corresponds either to a single computational step in the replicative ping-pong protocol or to an application of some congruence rule. On the other hand, any communication in the protocol can be directly simulated in Δ.

Hence we can make the following observation (while assuming that $O' \equiv \overline{w'_1}|\overline{w'_2}|\cdots|\overline{w'_{n'}}$).

Lemma 6.5 *It holds that*

$$A|B|O \longrightarrow^* A|B'|O'$$

if and only if

$$p^B(X|w_1.Z|w_2.Z|\cdots|w_n.Z) \longrightarrow^* p^{B'}(X|w'_1.Z|w'_2.Z|\cdots|w'_{n'}.Z).$$

Theorem 6.6 *The reachability problem for replicative ping-pong protocols is decidable.*

Proof. By Lemma 6.5 we reduced the problem to the reachability question for wPRS (observe that $p^{B'} \geq p^{B''}$ iff $B'' \subseteq B'$ is the natural ordering on control-states of the wPRS demonstrating that the control-state unit has a monotone behaviour). The decidability result then follows from Theorem 6.5. □

6.6 Conclusion

We have seen that ping-pong protocols extended with recursive definitions have full Turing power. This is the case even in the absence of nondeterministic choice operator '+'. A result like this implies that any reasonable property for all richer calculi cannot be automatically verified.

We also presented an explicit description of the active intruder in the syntax of recursive ping-pong protocols. The presented encoding gives an exponential blow-up in the size of the protocol. However, it is in fact not necessary and for a given protocol an active intruder of polynomial size can be constructed (it is enough to store the set of compromised keys in an extra buffer). We have decided to present the construction which is exponentially larger for clarity

reasons. Nevertheless the polynomial modification of the analysis and synthesis part using the extra buffer can be easily implemented – it only becomes technically more tedious.

Finally, we showed that reachability analysis for a replicative variant of the protocol becomes feasible. Our proof uses very recent results from process algebra [15] and can be compared to the work of Amadio, Lugiez and Vanackère [4] which establishes the decidability of reachability for a similar replicative protocol capable of ping-pong behaviour. Their approach uses a notion of a pool of messages explicitly modelled in the semantics and reduces the question to a decidable problem of reachability for prefix rewriting. In our approach we allow spontaneous generation of new messages which is not possible in their calculus. Moreover, we can distinguish between replicated and once-only behaviours (unlike in [4] where all processes have to be replicated).

Last but not least we hope that our approach can be possibly extended to include other operations as the decidability result for replicative protocols uses only a limited power of wPRS (only a parallel composition of stacks). Hence there is a place for further extensions of the protocol syntax while preserving a decidable calculus (e.g. messages of the form $\overline{\mathsf{k}_1(\mathsf{k}_2\ op\ \mathsf{k}_3)\mathsf{k}_4}$ for some extra composition operation op on keys can be easily stored in wPRS as $\mathsf{k}_1.(\mathsf{k}_2|\mathsf{k}_3).\mathsf{k}_4$). Such a study is left for future research.

Other open problems include decidability of bisimilarity for replicative ping-pong protocols and the question to determine general conditions that guarantee equi-expressiveness of recursion and replication (see e.g. [13, 17, 18]).

Bibliography

[1] M. Abadi and A.D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.

[2] R.M. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 499–514. Springer-Verlag, 2002.

[3] R.M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *LNCS*, pages 380–394. Springer-Verlag, 2000.

[4] R.M. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, October 2002.

[5] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of the 28th Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *LNCS*, pages 667–681. Springer, July 2001.

[6] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.

[7] N. Busi, M. Gabbrielli, and G. Zavattaro. Replication vs. recursive definitions in channel based calculi. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP'03)*, volume 2719 of *LNCS*, pages 133–144. Springer-Verlag, 2003.

[8] D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols. *Information and Control*, 55(1–3):57–68, 1982.

[9] D. Dolev and A.C. Yao. On the security of public key protocols. *Transactions on Information Theory*, IT-29(2):198–208, 1983.

[10] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In N. Heintze and E. Clarke, editors, *Proceedings of Workshop on Formal Methods and Security Protocols (FMSP'99)*, July 1999.

[11] M. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *14th IEEE Computer Security Foundations Workshop (CSFW '01)*, pages 160–173, Washington - Brussels - Tokyo, June 2001. IEEE.

[12] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *LNCS*, pages 354–372. Springer-Verlag, 2000.

[13] P. Giambiagi, G. Schneider, and F.D. Valencia. On the expressiveness of infinite behavior and name scoping in process calculi. In *Proceedings of the 7nd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'04)*, volume 2987 of *LNCS*, pages 226–240. Springer-Verlag, 2004.

[14] H. Hüttel and J. Srba. Recursive ping-pong protocols. In *Proceedings of the 4th International Workshop on Issues in the Theory of Security (WITS'04)*, pages 129–140, 2004.

[15] M. Křetínský, V. Řehák, and J. Strejček. Extended process rewrite systems: Expressiveness and reachability. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *LNCS*, pages 355–370. Springer-Verlag, 2004.

[16] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.

[17] R. Milner. The polyadic pi-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

[18] M. Nielsen, C. Palamidessi, and F.D. Valencia. On the expressive power of temporal concurrent constraint programming languages. In *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 156–167. ACM Press, 2002.

[19] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *TCS: Theoretical Computer Science*, 299, 2003.

[20] R.J. van Glabbeek. The linear time - branching time spectrum I: The semantics of concrete, sequential processes. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier Science, 2001.