Roland Meyer

# Concurrency Theory

– Lecture Notes –

February 17, 2012

University of Kaiserslautern

# Contents

# List of Figures

# Part I
# Concurrent Programs and Petri Nets

Concurrent programs, communicating asynchronously over a shared memory or synchronously via remote method invocation, can be conveniently modelled in terms of Petri nets. We introduce the basics of place/transition Petri nets, discuss fundamental verification problems, and study related analysis algorithms.

# Chapter 1
# Introduction to Petri Nets

**Abstract** First definitions on Petri nets and the notion of boundedness.

## 1.1 Syntax and Semantics

We present the basic concepts of Petri nets along the lines of [**?**, **?**, **?**]. Different from classical textbooks, we put some emphasis verification problems like deadlock freedom, reachability, coverability, and boundedness.

**Definition 1.1 (Petri net).** A *Petri net* is a triple $N = (S, T, W)$ where $S$ is a finite set of so-called *places*, $T$ a finite set of *transitions*, and $W : (S \times T) \cup (T \times S) \to \mathbb{N}$ a *weight function*. Places and transitions are disjoint, $S \cap T = \emptyset$.

Graphically, we represent places by circles, transitions by boxes, and the weight function by directed edges. More precisely, for $W(s,t) = k$ with $k \in \mathbb{N}$ we draw an edge from $s$ to $t$ that is labelled by $k$, and similar for edges $W(t,s)$ from $t$ to $s$. We omit edges weighted zero, and draw unlabelled ones if the weight is one.

The *preset* ${}^\bullet s$ of a place $s \in S$ contains the transitions with an arc leading to this place, ${}^\bullet s := \{t \in T \mid W(t,s) > 0\}$. Those transitions act productive on $s$. The transitions that consume from $s$ are in the *postset of s*, $s^\bullet := \{t \in T \mid W(s,t) > 0\}$. For transitions $t \in T$, the notions are similar. They operate upon the places in their *preset* ${}^\bullet t := \{s \in S \mid W(s,t) > 0\}$ and in their *postset* $t^\bullet := \{s \in S \mid W(t,s) > 0\}$.

Petri nets are automata comparable to finite state automata or Turing machines. Like every automaton model, they come equipped with a notion of state — called a marking in Petri nets — that influences the actions taken along a computation. What differentiates Petri nets from the remaining automata is the concurrency they reflect. A single Petri net typically captures the interaction of several programs. Therefore, a marking has to determine the next actions in all programs. The solution is to collect the places the different programs are currently in.

**Definition 1.2 (Marking and marked Petri net).** Let $N = (S, T, W)$. A *marking* is a function $M \in \mathbb{N}^S$ that assigns a natural number to every place. A *marked Petri net* is a pair $(N, M_0)$ of a Petri net and an *initial marking* $M_0$. We also write $N = (S, T, W, M_0)$ for a marked Petri net.

Markings are visualised by *tokens*, dots inserted into the circles that represent the places of a Petri net. For $M(s) = k$ with $k \in \mathbb{N}$ we put $k$ dots into the corresponding circle and say that *place s contains k tokens*.

To assess the complexity of verification problems for Petri nets, we measure their size by counting the number of places, transitions, arcs, and tokens.

**Definition 1.3 (Size of a Petri net).** The *size* of $N = (S, T, W, M_0)$ is $\|N\| := |S| + |T| + \sum_{s\in S}\sum_{t\in T}(W(s,t) + W(t,s)) + \sum_{s\in S} M_0(s)$.

The execution of transitions, called firing and denoted by[1] $M_1[t\rangle M_2$, changes the token count. But as markings are defined to be semi-positive, there is a restriction. A transition can only be fired if the places in its preset contain enough tokens.

**Definition 1.4 (Enabledness and deadlock).** Consider $N = (S, T, W)$ and $M \in \mathbb{N}^S$. Transition $t \in T$ is *enabled in M* if $M \geq W(-,t)$. Marking $M$ is a *deadlock of N* if it does not enable any transition.

The detection of deadlocks in a concurrent program is a fundamental problem in verification. Deadlocks point to incorrect assumptions on the synchronisation behaviour and thus to major flaws in the system of interest.

If transition $t$ is enabled, its firing produces $W(t,s)$ tokens on every place $s$ in its postset and, at the same time, consumes $W(s,t)$ tokens from the places in its preset.

**Definition 1.5 (Firing relation).** Let $N = (S, T, W)$. By definition, the *firing relation* $[\rangle \subseteq \mathbb{N}^S \times T \times \mathbb{N}^S$ contains the triple $(M_1, t, M_2)$, denoted by $M_1[t\rangle M_2$, if $t$ is enabled in $M_1$ and $M_2 = M_1 - W(-,t) + W(t,-)$.

We extend the firing relation to finite sequences of transitions $\sigma \in T^*$ inductively as follows. The empty word $\varepsilon$ does not change a marking, $M[\varepsilon\rangle M$ for all $M \in \mathbb{N}^S$. For any two markings $M_1, M_2 \in \mathbb{N}^S$ we then have $M_1[\sigma.t\rangle M_2$ if there is $M \in \mathbb{N}^S$ with $M_1[\sigma\rangle M$ and $M[t\rangle M_2$. The syntax $M_1[\sigma\rangle$ indicates that transition sequence $\sigma \in T^*$ is *enabled in* $M_1$, which means there is a marking $M_2 \in \mathbb{N}^S$ so that $M_1[\sigma\rangle M_2$. An infinite transition sequence $\sigma \in T^\omega$ is *enabled in* $M_1$, if so are all its finite prefixes. This means for all $\sigma_a \in T^*$ and $\sigma_b \in T^\omega$ with $\sigma = \sigma_a.\sigma_b$ we have $M_1[\sigma_a\rangle$.

**Definition 1.6 (Termination).** A Petri net $N = (S, T, W, M_0)$ *terminates* if no infinite transition sequence is enabled in $M_0$.

Termination is a second elementary problem in verification. Infinite transition sequences point to livelocks in a concurrent program, where components fail to leave certain commands.

A marking $M_2$ is said to be *reachable from marking* $M_1$, if there is a firing sequence leading from $M_1$ to $M_2$. We denote the *set of all markings reachable from* $M_1$ by $R(M_1) := \{M_2 \in \mathbb{N}^S \mid M_1[\sigma\rangle M_2 \text{ for some } \sigma \in T^*\}$. For the initial marking,

[1] Automata theory commonly uses the syntax $M_1 \xrightarrow{t} M_2$ for the execution of transitions.

we typically write $R(N)$ instead of $R(M_0)$ and call it the *state space of Petri net $N$*. Based on these notions, we define the *reachability graph* that documents the full firing behaviour of a Petri net. Its set of vertices is the state space of the Petri net, the edges correspond to the firing relation. More precisely, a $t$-labelled edge from $M_1$ to $M_2$ represents the firing $M_1[t\rangle M_2$. The initial marking forms the initial vertex.

**Definition 1.7 (Reachability graph).** Consider the Petri net $N = (S,T,W,M_0)$. The *reachability graph of $N$* is $RG(N) := (R(N), [\rangle \cap (R(N) \times T \times R(N)), M_0)$.

Mutual exclusion properties fail if *at least two programs* enter the critical section. Hence, in the correctness proof one ensures that no marking $M'$ is reachable that dominates marking $M$ with two programs in the critical section. This weaker notion of reachability is called coverability.

**Definition 1.8 (Coverability).** Consider the Petri net $N = (S,T,W)$. Marking $M_2$ *is coverable from* $M_1$, if there is $M \in R(M_1)$ with $M \geq M_2$.

We give a precise definition of the ordering among markings. Let $M,M' \in \mathbb{N}^S$. We write $M \geq M'$ for the fact that $M(s) \geq M'(s)$ for all places $s \in S$. Syntax $M \gneq M'$ indicates that $M \geq M'$ and additionally there is a place $s \in S$ with $M(s) > M'(s)$.

Not only is coverability often sufficient to ensure a system's correctness. As we shall see, it is also a property that remains decidable for infinite state models where reachability is lost.

## 1.2 Boundedness

Petri nets may have an infinite state space. Equivalently, there is no bound on the number of tokens that a place may contain. Amongst the bounded nets, safe nets where places carry at most one token play an important role.

**Definition 1.9 (Boundedness).** Consider Petri net $N = (S,T,W,M_0)$. Place $s \in S$ is called *$k$-bounded* with $k \in \mathbb{N}$ if in every reachable marking $M \in R(N)$ it carries at most $k$ tokens, $M(s) \leq k$. The place is *safe* if it is 1-bounded, and it is *bounded* if it is $k$-bounded for some $k \in \mathbb{N}$. The Petri net is called *$k$-bounded*, *safe*, or *bounded* if all its places satisfy the corresponding property.

We restrict our attention to unbounded and to safe Petri nets.

As was indicated above, unbounded Petri nets are those with an infinite state space.

**Lemma 1.1 (Finiteness).** *Petri net $N$ is bounded if and only if $R(N)$ is finite.*

One of the fascinating things about Petri nets is that important verification problems remain decidable in the infinite state case. In this section, we develop a decision procedure for boundedness that may be applied, for example, to examine the number of threads a server may generate. The argumentation serves as an appetiser for the proofs that are about to follow in this lecture. In particular, the idea of monotonicity appears in different flavours. Technically, *monotonicity* states that larger markings are able to imitate the behaviour of smaller ones.

**Lemma 1.2 (Monotonicity).** *Consider a Petri net $N = (S,T,W)$ and markings $M, M_1, M_2 \in \mathbb{N}^S$. If $M_1[\sigma\rangle M_2$ then $(M_1+M)[\sigma\rangle(M_2+M)$.*

Monotonicity shows that sequences of increasing markings point to an infinite state space.

**Lemma 1.3 (From increasing markings to unboundedness).** *Consider some Petri net $N$. If there are $M_1 \in R(N)$ and $M_2 \in R(M_1)$ with $M_2 \gneq M_1$, then $N$ is unbounded.*

*Idea.* By monotonicity, a transition sequence $\sigma$ from $M_1$ to $M_2$ with $M_2 \gneq M_1$ can be repeated in $M_2$. It leads to a marking $M_3$ with $M_3 \gneq M_2$ for which the argumentation again holds. We thus obtain

$$M_0[\tau\rangle M_1[\sigma\rangle M_2[\sigma\rangle M_3[\sigma\rangle \dots \qquad \text{with} \qquad M_1 \lneq M_2 \lneq M_3 \lneq \dots$$

The sequence adds an unbounded number of tokens to at least one place.

*Proof.* Assume $M_0[\tau\rangle M_1[\sigma\rangle M_2$ with $M_2 \gneq M_1$ and $\sigma, \tau \in T^*$. Since $M_2 \gneq M_1$, the difference $M := M_2 - M_1$ is greater zero, $M \gneq 0$. The observation that $M_2 = M_1 + M$ now justifies $M_1[\sigma\rangle(M_1+M)$. With monotonicity in Lemma 1.2, we add $M$ to $M_1$ and have $(M_1+M)[\sigma\rangle(M_1+2M)$. This means, sequence $\sigma$ can also be fired in $M_1+M$. Repeating the argumentation shows that $M_1[\sigma^i\rangle(M_1+iM)$ for every $i \in \mathbb{N}$.

To establish unboundedness, we proceed by contradiction. Let $k \in \mathbb{N}$ bound the token count in all markings on all places. Since $M \gneq 0$, there is a place $s \in S$ with $M(s) > 0$. We argued above that firing $\sigma$ for $(k+1)$-times is feasible and yields $M_1[\sigma^{k+1}\rangle(M_1+(k+1)M)$. In the resulting marking, place $s$ carries $(M_1+(k+1)M)(s) = M_1(s)+(k+1)M(s) \geq k+1$ tokens, which contradicts the boundedness assumption. ☐

Interestingly, also the reverse holds. If a Petri net is unbounded, one finds the token count increasing on some path. The proof relies on the fact that $\mathbb{N}^S$ is *well-quasi-ordered*. Every infinite sequence of markings $(M_i)_{i\in\mathbb{N}}$ contains comparable elements $i < j$ with $M_i \leq M_j$. We devote the full Section **??** to well-quasi-orderings and take this fact for granted.

**Lemma 1.4 (Comparable elements).** *Consider Petri net $N$. Every infinite sequence $(M_i)_{i\in\mathbb{N}}$ of markings in $R(N)$ contains indices $i < j$ with $M_i \leq M_j$.*

**Lemma 1.5 (From unboundedness to increasing markings).** *If $N$ is unbounded, then there are $M_1 \in R(N)$ and $M_2 \in R(M_1)$ with $M_2 \gneq M_1$.*

*Idea.* We summarise all transition sequences that do not repeat markings in a tree. To be precise, we say that a transition sequence $M_0[t_1\rangle M_1[t_2\rangle \dots [t_n\rangle M_n$ *does not repeat markings*, if $M_i \neq M_j$ for all $i \neq j$. Note that every reachable marking can be obtained without repetitions. Hence, as we have an infinite state space, the tree is infinite. The outdegree is bounded by the number of transitions. An application of König's lemma[2] now shows the existence of an infinite path $(M_i)_{i\in\mathbb{N}}$ in this tree. Lemma 1.4 gives two comparable elements $i < j$ with $M_i \leq M_j$ on the path. By construction, the markings are distinct, $M_i \lneq M_j$, as required.

*Proof.* Consider the unbounded Petri net $N = (S,T,W,M_0)$. We construct a tree $(Q,\rightarrow,q_0,lab)$ with vertices $Q$, edges $\rightarrow\, \subseteq Q\times T\times Q$, root $q_0$, and vertex labelling $lab$ that assigns to every $q \in Q$ a marking $M \in \mathbb{N}^S$. The procedure, stated in pseudo code in Figure 1.1, works as follows. The root $q_0$ is labelled by the initial marking. For every vertex $q_1 \in Q$ labelled by $M_1$ and every transition $t \in T$, compute $M_2$ with $M_1[t\rangle M_2$. If $M_2$ does not label a vertex on the path from $q_0$ to $q_1$, add a new vertex $q_2$ to the tree and label it by $M_2$. Add the edge $(q_1,t,q_2)$. If transition $t$ is disabled in $M_1$ or $M_2$ has already been seen, consider the next transition.

$lab(q_0) = M_0$
**for all** $q_1 \in Q$ **do**
  **for all** $t \in T$ **do**
    **let** $lab(q_1) = M_1$
    **if** $M_1[t\rangle M_2$ and $M_2$ does not label a vertex on the path from $q_0$ to $q_1$ **then**
      add new vertex $q_2$ to $Q$ with $lab(q_2) = M_2$
      add edge $(q_1,t,q_2)$ to $\rightarrow$
      (♠)  //decision procedure in Theorem 1.1
    **end if**
  **end for all**
**end for all**
(♣)  //decision procedure in Theorem 1.1

**Fig. 1.1** Tree computation in the decision procedure for boundedness.

Since $N$ is unbounded, its state space is infinite according to Lemma 1.1. Every marking $M \in R(N)$ is reachable without repetitions and thus labels some $q \in Q$. Therefore, the tree computed above is infinite. Its outdegree is bounded by the number of transitions, hence finite. By König's lemma, there is an infinite path $q_0,t_1,q_1,t_2,q_2,t_3\ldots$ in the tree with $(q_i,t_{i+1},q_{i+1}) \in\, \rightarrow$ for all $i \in \mathbb{N}$. Its vertex labelling $lab(q_i) = M_i$ forms an infinite sequence of markings $(M_i)_{i\in\mathbb{N}}$ for which Lemma 1.4 finds two comparable elements $M_i \leq M_j$ with $i < j$. They are different by construction, $M_i \lneq M_j$. The observation that edges represent transition firings yields $M_0[t_1\rangle M_1[t_2\rangle M_2[t_3\rangle\ldots$ and allows us to conclude $M_i \in R(M_0)$ and $M_j \in R(M_i)$. □

The proof of Lemma 1.5 suggests the following decision procedure for boundedness. Compute the tree of all transition sequences and report unboundedness at (♠)

[2] König's lemma states that every infinite tree of finite outdegree contains an infinite path.

when increasing markings are found. If no such markings exist, the computation eventually stops and returns boundedness at (♣). The procedure is correct for both, depth-first and breadth-first implementations of the tree computation.

**Theorem 1.1 (Decidability of boundedness).** *It is decidable, whether a Petri net $N$ is bounded.*

To turn the algorithm given in Figure 1.1 into a decision procedure for boundedness, add the following commands at the points specified:

(♠) **if** $M_2 \gneq M$ for some $M$ labelling a vertex on the path from $q_0$ to $q_1$ **then**
  **return** unbounded
 **end if**

(♣) **return** bounded

*Proof.* To establish correctness, assume the decision procedure is applied to an unbounded Petri net. By Lemma 1.5, there are $M_1 \in R(N)$ and $M_2 \in R(M_1)$ with $M_2 \gneq M_1$. Hence, a breadth-first implementation of the tree computation will find vertices $q_1$ reachable from $q_0$ and labelled by $M_1$ as well as $q_2$ reachable from $q_1$ and labelled by $M_2$. The **if** condition in (♠) applies and returns *unbounded*.

The proof of Lemma 1.5 actually shows that *every* infinite path contains two comparable elements $M_2 \gneq M_1$. Hence, a depth-first implementation either finds (♠) satisfied or backtracks from a finite path. As the tree contains an infinite path, the search eventually returns the correct answer.

If the algorithm is applied to a bounded Petri net, by Lemma 1.3 (contraposition) there are no $M_1 \in R(N)$ and $M_2 \in R(M_1)$ with $M_2 \gneq M_1$. Condition (♠) never applies. However, as the Petri net is bounded its state space is finite by Lemma 1.1. The tree computation eventually stops and returns *bounded* at (♣). □

For bounded Petri nets, the decision procedure determines the full state space. To demonstrate that this leads to unacceptable runtimes, we present a construction by Ernst Mayr (*1950) and Albert Meyer (*1941). It provides bounded Petri nets of size $O(n)$ that generate the astronomic number of $A(n)$ tokens. The Ackermann function $A(n)$ is well known to be not primitive recursive.

**Definition 1.10 (Ackermann function).** Consider the functions $A_i \in \mathbb{N}^{\mathbb{N}}$ defined by

$$A_0(x) := x+1 \qquad A_{n+1}(0) := A_n(1) \qquad A_{n+1}(x+1) := A_n(A_{n+1}(x))$$

The *Ackermann function* $A \in \mathbb{N}^{\mathbb{N}}$ is defined as $A(n) := A_n(n)$.

**Theorem 1.2 (Mayr and Meyer [?]).** *For every $n \in \mathbb{N}$, there is a bounded Petri net $N_n$ of size $O(n)$ that generates $A(n)$ tokens on some place.*

As a consequence, there is no primitive recursive relationship between the size of a bounded Petri net and the size of its state space.

**Corollary 1.1.** *For bounded Petri nets $N$, the size of $R(N)$ and thus $RG(N)$ is not bounded by a primitive recursive function in the size of $N$.*

*Proof (of Theorem 1.2).* We follow the presentation in [**?**]. The sequence of Petri nets $N_n$ is constructed inductively. To this end, we equip them with four interface places. A *starting* and a *halting place* control the computation so that the Petri nets terminate when the halting place gets marked. The *input place* expects $x \in \mathbb{N}$ tokens and the transition sequences produce up to $A_n(x)$ tokens on the *output place*.

Let $N_n(x)$ mean that the input of $N_n$ contains $x \in \mathbb{N}$ tokens and the start place has a single token. The remaining places are unmarked. To make our statement precise, we require that $N_n(x)$ is bounded by $A_n(x)$. The Petri net terminates. Moreover, $M \in R(N_n(x))$ is a deadlock if and only if $M(\text{stop}) = 1$ and $M(\text{start}) = 0$. There is such a deadlock with $M(\text{stop}) = 1$, $M(\text{out}) = A_n(x)$, and $M(s) = 0$ otherwise.

Petri net $N_0$ is depicted in Figure 1.2. It moves all tokens on the input place to the output place, and finally adds one token before halting. The conditions on boundedness and termination are readily checked.

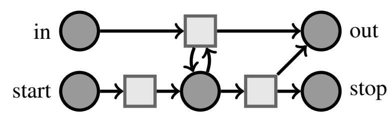



**Fig. 1.2** Petri net $N_0$ computing $A_0(x) := x+1$.

Petri net $N_{n+1}$ extends $N_n$ by connecting to the interface places. The construction, given in Figure 1.3, exploits the equation

$$A_{n+1}(x) = \underbrace{A_n(\ldots A_n(A_n}_{(x+1)-times}(1))\ldots).$$

To compute $A_{n+1}(x)$ with $N_{n+1}(x)$, observe that by the induction hypothesis $N_n(1)$ always terminates with a token on stop. Among the transition sequences there is one that, upon halting, gives $A_n(1)$ tokens on the output place and an otherwise (up to stop) empty net $N_n$. We transfer the tokens back to the input place of $N_n$, thereby decrementing $x$. This restarts $N_n$ with input $A_n(1)$, $N_n(A_n(1))$. We again apply the hypothesis to find a terminating run that produces $A_n(A_n(1))$ tokens on the output place of $N_n$. This means, we have $A_{n+1}(1)$ tokens on the output place of $N_n$ and still $x-1$ tokens in the input place of $N_{n+1}$. Repeating the computation and transfer of $N_n$ for another $(x-1)$-times provides $A_{n+1}(x)$ tokens on the output place of $N_n$. The Petri net cannot be restarted as no tokens are left on the input of $N_{n+1}$. Instead the $A_{n+1}(x)$ tokens on the output of $N_n$ are transferred to the output of $N_{n+1}$. Petri net $N_{n+1}(x)$ then halts with the required token count. In Figure 1.3, the start of $N_n$ on 1 as well as the loop construction from the output place of $N_n$ back to its input place are highlighted in red and blue, respectively.

Petri net $N_{n+1}(x)$ does not guarantee that all tokens on the output place of $N_n$ are transferred back to the input. This does not lead to a bound larger than $A_{n+1}(x)$. Assume we transfer $y$ tokens and $x$ tokens remain in the output place of $N_n$. After

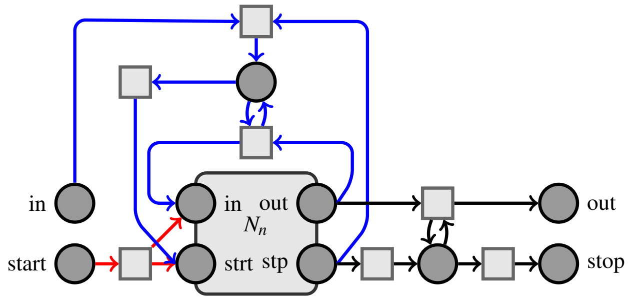



**Fig. 1.3** Petri net $N_{n+1}$ computing $A_{n+1}(x+1) := A_n(A_{n+1}(x))$.

the computation of $N_n(y)$, we find at best $A_n(y)+x$ tokens on the output of $N_n$. By $A_n(y)+x \leq A_n(y+x)$, keeping tokens in the output place only decreases the overall token count. Hence, $A_{n+1}(x)$ in fact bounds the token count of $N_{n+1}(x)$.

The size of $N_0$ is 17. For $N_n$, we add 33 items to the size of $N_{n-1}$ and obtain

$$\|N_n\| = 33 + \|N_{n-1}\| = 33n + 17.$$

Initially, the input place of $N_n$ carries $n$ tokens and the start place has a single token. The marked Petri net thus has a size of $34n + 18 \in O(n)$. $\square$

# Chapter 2
# Invariants

**Abstract** Linear programming techniques for Petri net verification.

## 2.1 Marking Equation

Our goal is to exploit linear algebraic techniques for reasoning about (un)reachability and (non)coverability of markings. To this end, we rephrase the firing relation as a linear algebraic operation. As a first step, equip the set of places in $N = (S, T, W, M_0)$ with a total ordering that we indicate by indices $S = \{s_1, \ldots, s_n\}$. It turns marking $M \in \mathbb{N}^S$ into a vector $M \in \mathbb{N}^{|S|}$ of dimension $|S|$. Similarly, for a fixed transition $t \in T$ the weight function $W : (S \times T) \cup (T \times S) \to \mathbb{N}$ is expressed by the two vectors

$$W(-,t) := \begin{pmatrix} W(s_1,t) \\ \vdots \\ W(s_n,t) \end{pmatrix} \qquad W(t,-) := \begin{pmatrix} W(t,s_1) \\ \vdots \\ W(t,s_n) \end{pmatrix} \qquad \text{in } \mathbb{N}^{|S|}.$$

With an additional ordering on transitions, $T = \{t_1, \ldots, t_m\}$, the weight function $W$ yields two matrices. The *forward matrix* $\mathbb{F} \in \mathbb{N}^{|S| \times |T|}$ contains the weights on arcs from places to transitions, i.e., forward relative to the places. The *backwards matrix* $\mathbb{B} \in \mathbb{N}^{|S| \times |T|}$ gives the arcs from transitions to places:

$$\mathbb{F} := \big( W(-,t_1) \, \ldots \, W(-,t_m) \big) \qquad \mathbb{B} := \big( W(t_1,-) \, \ldots \, W(t_m,-) \big)\,.$$

Together, forward and backward matrix contain full information about the weight function so that we may alternatively give $N = (S, T, W)$ as $N = (S, T, \mathbb{F}, \mathbb{B})$.

**Definition 2.1 (Connectivity matrix).** Let $N = (S, T, \mathbb{F}, \mathbb{B})$. Its *connectivity matrix* is $\mathbb{C} := \mathbb{B} - \mathbb{F} \in \mathbb{Z}^{|S| \times |T|}$.

The connectivity matrix does not indicate loops in the Petri net. If there are arcs from place $s$ to transition $t$ and vice versa, $W(s,t) = 1 = W(t,s)$, we get $\mathbb{C}(s,t) = 0$. Missing arcs with $W(s,t) = 0 = W(t,s)$ have the same entry.

The $i$-th column of $\mathbb{C}$ gives the difference $W(t_i,-) - W(-,t_i)$, which is precisely the vector added to a marking upon firing of $t_i$. With this insight, we can determine the goal marking $M_2$ reached from $M_1$ via a full transition sequence $\sigma$ directly from $M_1$ and $\sigma$, without intermediary firings. The key is that just the number of transitions but not their ordering in a firing sequence is important.

**Definition 2.2 (Parikh image).** Consider transitions $T$. The *Parikh image of* $\sigma \in T^*$ is the function $p(\sigma) \in \mathbb{N}^T$ with $(p(\sigma))(t) =$ number of occurrences of $t$ in $\sigma$.

To illustrate the independence of the transition ordering, consider some sequence $\sigma = t_1.t_2.t_1$ over two transitions. If $M_1[\sigma\rangle M_2$ by definition of firing $M_2$ satisfies

$$\begin{aligned} M_2 &= M_1 + \mathbb{C}(-,t_1) + \mathbb{C}(-,t_2) + \mathbb{C}(-,t_1) \\ &= M_1 + \mathbb{C}(-,t_1)(p(\sigma)(t_1)) + \mathbb{C}(-,t_2)(p(\sigma)(t_2)). \end{aligned}$$

Taking the Parikh image as a vector, this sum is $M_0 + \mathbb{C} \cdot p(\sigma)$.

**Lemma 2.1 (Marking equation).** *Consider* $N = (S,T,W)$ *with connectivity matrix* $\mathbb{C} \in \mathbb{N}^{|S|\times|T|}$*,* $M_1, M_2 \in \mathbb{N}^{|S|}$*, and* $\sigma \in T^*$*. If* $M_1[\sigma\rangle M_2$ *then* $M_2 = M_1 + \mathbb{C} \cdot p(\sigma)$*.*

As an immediate consequence, if marking $M_2$ is reachable from $M_1$ the equation $M_2 - M_1 = \mathbb{C} \cdot x$ has a solution in $\mathbb{N}^{|T|}$. This can be applied in contraposition for verification. If the equation has no solution then $M_2$ is not reachable from $M_1$.

Vice versa, any vector $K \in \mathbb{N}^{|T|}$ is the Parikh image of a transition sequence, $K = p(\sigma)$ for some $\sigma \in T^*$. This $\sigma$ has an enabling marking, say $M_1 \in \mathbb{N}^{|S|}$. Hence, the following weak reverse of the above holds. If $M = \mathbb{C} \cdot x$ has a natural solution, then there is a marking $M_1$ that reaches $M_1 + M$. We summarise both arguments.

**Lemma 2.2.** *Consider* $N = (S,T,\mathbb{F},\mathbb{B})$ *with connectivity matrix* $\mathbb{C} \in \mathbb{N}^{|S|\times|T|}$ *and let* $M \in \mathbb{N}^{|S|}$*. There is a marking* $M_1 \in \mathbb{N}^{|S|}$ *with* $M_1 + M \in R(M_1)$ *if and only if* $M = \mathbb{C} \cdot x$ *has a solution in* $\mathbb{N}^{|T|}$*.*

Interestingly, in combination with the statements on boundedness from Section 1.2, Lemma 2.2 provides a characterisation of *structural boundedness*, i.e., boundedness under any initial marking.

**Proposition 2.1 (Characterisation of structural boundedness).** *Consider Petri net* $N = (S,T,\mathbb{F},\mathbb{B})$ *with connectivity matrix* $\mathbb{C} \in \mathbb{N}^{|S|\times|T|}$*. There is an initial marking* $M_0$ *so that* $(N, M_0)$ *is unbounded if and only if* $\mathbb{C} \cdot x \gneq 0$ *has a solution in* $\mathbb{N}^{|T|}$*.*

*Proof.* For the direction from right to left, assume $\mathbb{C} \cdot x \gneq 0$ has a solution in $\mathbb{N}^{|T|}$. This means, there is a marking $M \gneq 0$ with $\mathbb{C} \cdot x = M$. By Lemma 2.2, we find a marking $M_1$ with $M_1 + M \in R(M_1)$. We are thus in a position to apply Lemma 1.3 to the Petri net $N$ with initial marking $M_1$, which proves unboundedness. $\square$

Parametric verification considers families of systems where the instances differ only in certain parameters. A typical example are client server architectures that vary in the number of threads. In this light, Proposition 2.1 can be interpreted as follows. Solving $\mathbb{C} \cdot x \gneq 0$ checks for whether the size, for example of buffers, is bounded in all instances of the architectural family.

## 2.2 Structural and Transition Invariants

In our running example, we noticed that always either the semaphore or the critical section carries a token. This can be formulated as

$$M(pcs) + M(sem) = 1, \tag{2.1}$$

and the equation holds in every reachable marking. Structural invariants provide a means to derive such equations. Technically, a structural invariant is a vector $I \in \mathbb{Z}^{|S|}$ with $\mathbb{C}^T \cdot I = 0$. Here, $\mathbb{C}^T$ denotes the transpose of the connectivity matrix, and therefore $I$ is in fact an $|S|$-dimensional vector. The intuition is that $I$ gives a weight to the tokens on each place.

**Definition 2.3 (Structural invariant).** Consider $N = (S, T, W)$ with connectivity matrix $\mathbb{C} \in \mathbb{Z}^{|S| \times |T|}$. A *structural invariant* $I \in \mathbb{Z}^{|S|}$ is a solution to $\mathbb{C}^T \cdot x = 0$.

The main property of structural invariants is that the $I$-weighted sum of tokens stays constant under transition firings. This follows, by a beautiful algebraic trick, from the marking equation and the requirement that $\mathbb{C}^T \cdot I = 0$.

**Theorem 2.1 (Invariance of structural invariants).** *Let $I \in \mathbb{Z}^{|S|}$ be a structural invariant of $N = (S, T, W, M_0)$. Then for all $M \in R(N)$ we have $I^T \cdot M = I^T \cdot M_0$.*

Before we turn to the proof, we show how to derive Equation 2.1 in our example with the help of Theorem 2.1. Note that $I(pw) = 0$ and $I(pcs) = 1 = I(sem)$ is a structural invariant of $N_{M+S}$. The initial marking is $M_0(pw) = 2$, $M_0(sem) = 1$, and $M_0(pcs) = 0$. An application of Theorem 2.1 yields

$$I^T \cdot M = 0M(pw) + 1M(pcs) + 1M(sem) = 1 = I^T M_0,$$

as required. The equation holds for all markings reachable in $N_{M+S}$.

*Idea (Theorem 2.1).* The proof relies on the marking equation $M = M_0 + \mathbb{C} \cdot p(\sigma)$. We multiply the invariant from the left and exploit the laws of transposition

$$I^T \cdot (\mathbb{C} \cdot p(\sigma)) = (\mathbb{C}^T \cdot I)^T \cdot p(\sigma)$$

to swap the positions of $I$ and $\mathbb{C}$. The definition of structural invariants concludes the proof.

*Proof.* Let $N = (S,T,W,M_0)$ be a Petri net with connectivity matrix $\mathbb{C} \in \mathbb{Z}^{|S|\times|T|}$ and structural invariant $I \in \mathbb{Z}^{|S|}$. Assume marking $M \in R(N)$ is reachable by $M_0[\sigma\rangle M$ for some $\sigma \in T^*$. The marking equation yields $M = M_0 + \mathbb{C} \cdot p(\sigma)$. We multiply the transposed of $I$ to both sides and obtain

$$\begin{aligned}
& I^T \cdot M \\
\{ \text{ Marking equation } \} &= I^T \cdot (M_0 + \mathbb{C} \cdot p(\sigma)) \\
\{ \text{ Distributivity } \} &= I^T \cdot M_0 + I^T \cdot (\mathbb{C} \cdot p(\sigma)) \\
\{ \text{ Associativity, transposition self inverse } \} &= I^T \cdot M_0 + (I^T \cdot \mathbb{C}^{T^T}) \cdot p(\sigma) \\
\{ \text{ Transposition law } B^T \cdot A^T = (A \cdot B)^T \} &= I^T \cdot M_0 + (\mathbb{C}^T \cdot I)^T \cdot p(\sigma) \\
\{ \text{ Definition structural invariant } \} &= I^T \cdot M_0 + 0^T \cdot p(\sigma).
\end{aligned}$$

The latter is $I^T \cdot M_0$ which concludes the proof. □

Applied in contraposition, Theorem 2.1 yields a reachability check. If a marking does not satisfy the equality stated above, it cannot be reachable.

**Corollary 2.1 (Unreachability via structural invariants).** *Let $N = (S,T,W,M_0)$ with structural invariant $I \in \mathbb{Z}^{|S|}$ and $M \in \mathbb{N}^{|S|}$. If $I^T \cdot M \neq I^T \cdot M_0$, then $M \notin R(N)$.*

Structural invariants can be computed in polynomial time using Gauss elimination with $O(\|N\|^3)$. Therefore, this test can be performed efficiently prior to heavier reachability analyses. To obtain more expressive structural invariants, they can be added up and scaled by a constant.

**Lemma 2.3.** *If $I_1$ and $I_2$ are structural invariants of N, so are $I_1 + I_2$ and $kI_1$, $k \in \mathbb{Z}$.*

Invariants also yield bounds for the places that are weighted strictly positive. The lemma follows from Theorem 2.1 and only holds for non-negative invariants.

**Lemma 2.4 (Place boundedness from structural invariants).** *Let $N = (S,T,W)$ with structural invariant $I \in \mathbb{N}^{|S|}$. Let $s \in S$ with $I(s) > 0$. Then s is bounded under any initial marking $M_0 \in \mathbb{N}^{|S|}$.*

As a consequence, a Petri net $N = (S,T,W)$ that has a *covering structural invariant* $I \in \mathbb{N}^{|S|}$ with $I(s) > 0$ for all $s \in S$ is bounded under any initial marking.

**Corollary 2.2 (Structural boundedness from structural invariants).** *If N has a covering structural invariant, it is structurally bounded.*

**Definition 2.4 (Transition invariant).** Consider $N = (S,T,W)$ with connectivity matrix $\mathbb{C} \in \mathbb{Z}^{|S|\times|T|}$. A *transition invariant* $J \in \mathbb{N}^{|T|}$ is a solution to $\mathbb{C} \cdot x = 0$.

Transition sequences with Parikh vector $J$ do not change the marking. Vice versa, if a transition sequence does not change the marking, then its Parikh vector is a transition invariant.

**Theorem 2.2 (Invariance of transition invariants).** *Consider transition sequence $M[\sigma\rangle M'$ in a Petri net $N = (S,T,W)$. Then $M' = M$ if and only if $p(\sigma)$ is a transition invariant of N.*

Transition invariants again can be added up and scaled by a constant. Reachability graphs of Petri nets without transition invariants are acyclic.

## 2.3 Traps and Siphons

Petri nets are bipartite graphs. Up to now, we have used this fact only indirectly when we defined the connectivity matrix. In this section, we explictily use the graph structure to derive inequalities on the token count that are invariant under firings. The corresponding notions of siphons and traps are classical in Petri net theory. They describe regions of places in a Petri net that tokens can never leave or enter.

Traps and siphons can be defined for Petri nets with weighted arcs, but this causes a formal overhead. We therefore assume that transitions are weighted either zero or one. More formally, in this section we consider *ordinary* Petri nets $N = (S,T,W)$ where $W : (S\times T)\cup(T\times S) \to \{0,1\}$.

**Definition 2.5 (Trap).** A *trap* is a subset $Q \subseteq S$ of places that satisfies $Q^\bullet \subseteq {}^\bullet Q$. A trap is said to be *marked under* $M \in \mathbb{N}^{|S|}$ if $M(q) \geq 1$ for some $q \in Q$.

Intuitively, whenever a transition intends to remove tokens from a place in the trap, then the transition will also produce tokens in some place in the trap. The places need not coincide. As a consequence, initially marked traps remain marked in all reachable markings.

**Lemma 2.5 (Trap property).** *Let $Q$ be a trap of Petri net $N = (S,T,W,M_0)$ that is marked under $M_0$. Then $\sum_{q\in Q} M(q) \ \geq\ 1$ holds for all $M \in R(N)$.*

The inequality may be satisfied by sets of places that do not form a trap. Therefore, the reverse of the implication does not hold. The statement may be interpreted as a necessary condition for reachability.

**Corollary 2.3.** *Let $Q \subseteq S$ be an initially marked trap of $N$ and $M \in \mathbb{N}^{|S|}$ a marking. If $\sum_{q\in Q} M(q) \ =\ 0$ then $M$ is not reachable, $M \notin R(N)$.*

It can be shown that the union of traps again forms a trap. As a consequence, we can restrict ourselves to families of traps that generate the remaining traps by union. On the one hand, this reduces the effort when computing traps. On the other hand, smaller traps yield more precise inequalities in Lemma 2.5.

**Definition 2.6 (Generating family of traps).** Let $\{Q_1,\ldots,Q_n\}$ be a family of traps in a Petri net $N$. The family is said to be *generating* if every trap $Q$ of $N$ can be obtained as union of traps in the family, $Q = \bigcup_{j\in J} Q_j$ for some subset $J \subseteq \{1,\ldots,n\}$. A trap is called *minimal* if it does not contain further traps.

A generating family certainly contains all minimal traps in a Petri net. But in turn, the minimal traps do not necessarily form a generating family. Moreover, there are Petri nets with a single family of generating traps that is exponential in the size of the net. Therefore, to use traps in verification, we should represent them symbolically in some formalism instead of computing them explicitly.

As symbolic formalism, we again target linear programming. We set up a system of linear inequalities

$$\begin{aligned} Y^T \cdot \mathbb{C}_Q &\geq 0 \\ Y &\geq 0 \end{aligned} \tag{2.2}$$

whose rational solutions characterize traps. For the technical development, we first introduce a variable $Y(s)$ for every place $s \in S$. By definition, traps $Q \subseteq S$ satisfy $Q^\bullet \subseteq {}^\bullet Q$. For every place $s_1 \in Q$ and every transition $t \in s_1^\bullet$ some place $s_2 \in t^\bullet$ belongs to $Q$. With place variables, we formulate the requirement equivalently as

$$Y(s_1) \;\leq\; \sum_{s_2 \in t^\bullet} Y(s_2).$$

Fix $s_1 \in S$ and $t \in s_1^\bullet$. To rephrase the above inequality with matrix multiplication, let $E_{s_1}$ denote the unit vector for dimension $s_1$. Moreover, we set up the *postset vector* $V_{t^\bullet} \in \{0,1\}^{|S|}$ with $V_{t^\bullet}(s) = 1$ iff $s \in t^\bullet$. The inequality is then equivalent to

$$E_{s_1}^T \cdot Y \;\leq\; V_{t^\bullet}^T \cdot Y \qquad \Leftrightarrow \qquad Y^T \cdot (V_{t^\bullet} - E_{s_1}) \;\geq\; 0.$$

To check the inclusion $Q^\bullet \subseteq {}^\bullet Q$ on all places in a trap and all surrounding transitions, we summarize the above vectors $V_{t^\bullet} - E_{s_1}$ in a matrix.

**Definition 2.7 (Trap matrix).** The *trap matrix* $\mathbb{C}_Q \in \mathbb{Z}^{|S| \times |S||T|}$ is defined by setting $\mathbb{C}_Q(-,(s,t)) := V_{t^\bullet} - E_s$ if $t \in s^\bullet$ and $\mathbb{C}_Q(-,(s,t)) := 0$ otherwise.

In combination with the equivalences derived above, we obtain a linear algebraic characterization of traps. For a concise statement, consider $Q \subseteq S$. The associated vector is $K_Q \in \mathbb{Q}^{|S|}$ with $K_Q(s) := 1$ if $s \in Q$ and $K_Q(s) := 0$ otherwise. Vice versa, every vector $K \in \mathbb{Q}^{|S|}$ describes the set $Q_K := \{s \in S \mid K(s) > 0\}$.

**Proposition 2.2.** *Consider Petri net $N$. If $Q \subseteq S$ is a trap, then $K_Q \in \mathbb{Q}^{|S|}$ satisfies Inequality 2.2. In turn, if $K \in \mathbb{Q}^{|S|}$ satisfies Inequality 2.2, then $Q_K \subseteq S$ is a trap.*

The concept dual to traps are so-called siphons. They describe regions in a Petri net that cannot receive tokens. Technically, every transition that acts productive on the places in a siphon also consumes tokens from the siphon.

**Definition 2.8 (Siphon).** A *siphon* of Petri net $N$ is a subset $D \subseteq S$ of places that satisfies ${}^\bullet D \subseteq D^\bullet$. A siphon is *empty under $M \in \mathbb{N}^{|S|}$* if $M(s) = 0$ for all $s \in D$.

A siphon that is initially empty blocks all transitions that produce tokens on it.

**Lemma 2.6 (Siphon property).** *Consider Petri net $N = (S,T,W,M_0)$ with siphon $D \subseteq S$ that is empty under the initial marking $M_0 \in \mathbb{N}^{|S|}$. Then $\sum_{s \in D} M(s) = 0$ holds for all $M \in R(N)$.*

Like for traps, a union of siphons again yields a siphon. Moreover, empty siphons characterize deadlock situations in a Petri net. The statement only holds for Petri nets where every transition depends on a place, i.e., in the following we assume that for each $t \in T$ there is $s \in S$ with $s \in {}^\bullet t$.

**Lemma 2.7 (Deadlocks and empty siphons).** *If $M \in \mathbb{N}^{|S|}$ is a deadlock of $N$ then there is a siphon $D \subseteq S$ that is empty under $M$.*

Set $D := \{s \in S \mid M(s) = 0\}$ to contain the places that are empty in $M$. Consider $t \in {}^\bullet D$. Since $t$ is dead, there is $s \in {}^\bullet t$ with $M(s) = 0$. This means $t \in D^\bullet$.

## 2.4 Verification by Linear Programming

We develop a powerful constraint-based verification algorithm for Petri nets that is based on the linear algebraic insights obtained so far. Instead of constructing the Petri net's state space, the algorithm sets up a system of inequalities whose infeasibility proves correctness. As a consequence, the approach circumvents the state space explosion problem and is rather fast. Moreover, it is not restricted to finite state systems. On the downside, the algorithm is only sound but not complete. If it finds the constraint system infeasible, it concludes correctness of the Petri net. In turn, although the Petri net is correct the algorithm may find the constraint system feasible. In this case it returns unknown. To begin with, we make the notion of correctness precise.

**Definition 2.9 (Property).** A *property* is a function $\mathscr{P} : \mathbb{N}^{|S|} \longrightarrow \mathbb{B}$ that assigns a Boolean value to each marking. We write $\mathscr{P}(M)$ rather than $\mathscr{P}(M) = t$ and similarly $\neg\mathscr{P}(M)$ for $\mathscr{P}(M) = f$. A property *holds* for a Petri net $N$ if $\mathscr{P}(M)$ holds for all $M \in R(N)$. A property is *co-linear* if its violation can be expressed by a linear inequality: $\neg\mathscr{P}(M)$ if and only if $A \cdot M \geq B$ for some $A \in \mathbb{Q}^{k\times|S|}$ and $B \in \mathbb{Q}^k$ for some $k \in \mathbb{N}$.

**Definition 2.10 (Linear, integer, mixed programming).** A *linear programming problem* is a set of linear inequalities $A \cdot X \leq B$ with $A \in \mathbb{Q}^{m\times n}$ and $B \in \mathbb{Q}^m$ on a set of variables $X \in \mathbb{Q}^n$. The inequalities are also called *constraints*. There may be an additional *objective function* $C^T \cdot X$ with $C \in \mathbb{Q}^n$ to be maximized. We denote a linear programming problem by

Variables: $X$ (potentially with type)
Maximize $C^T \cdot X$ subject to
$A \cdot X \leq B$.

A *solution* to the problem is a vector $K \in \mathbb{Q}^n$ that satisfies $A \cdot X \leq B$. The solution is *optimal* if it maximizes $C^T \cdot X$ in the space of all solutions.

If the solution is required to be *integer*, $K \in \mathbb{Z}^n$, then the problem is called *integer programming problem*. If some variables are to receive integer values while others can be evaluated rational, we have a *mixed programming problem*. A linear, integer, or mixed programming problem is called *feasible* if it has a solution. Otherwise it is called *infeasible*.

Linear programming is in P while mixed and integer programming are NP-complete. We explain how integer programming helps checking whether a Petri net $N$ satisfies a property $\mathscr{P}$. Assume this is not the case. Then there is a marking $M \in R(N)$ that violates the property. Recall that the marking equation overapproximates the state space. This means $M$ satisfies $M = M_0 + \mathbb{C} \cdot X$ for some $X \in \mathbb{N}^{|T|}$. By co-linearity, violation $\neg\mathscr{P}(M)$ is expressed by $A \cdot M \geq B$. To sum up, a reachable marking that violates the property solves the following integer programming problem.

**Definition 2.11 (Basic verification system).** Consider Petri net $N = (S, T, W, M_0)$ and a co-linear property $\mathscr{P}$ defined by $A \cdot X \geq B$ for some $A \in \mathbb{Q}^{k \times |S|}$ and $B \in \mathbb{Q}^k$ with $k \in \mathbb{N}$. The *basic verification system (BVS)* associated to $N$ and $\mathscr{P}$ is

$$\begin{array}{l} \text{Variables: } X, M \text{ integer} \\ \qquad M = M_0 + \mathbb{C} \cdot X \\ \quad M, X \geq 0 \\ \quad A \cdot M \geq B. \end{array}$$

We argued that feasibility of BVS is necessary for a violation to the property.

**Proposition 2.3.** *Consider a Petri net $N$ and a property $\mathscr{P}$. If the associated BVS is infeasible, then $\mathscr{P}$ holds for $N$.*

Basic verification systems are too weak for the analysis of concurrent programs that communicate via shared variables. Programs typically rely on tests of the form

$$c_0;\ \textbf{if } x = 0 \textbf{ then } c_1;\ \ldots\ \textbf{else } \ldots$$

to determine the flow of control. These tests are canonically modelled by loops in Petri nets. There is a transition $t$ leading from a place for command $c_0$ to a place for command $c_1$. This transition has arcs from and to a place $s$ that reflects the valuation $x = 0$. In consequence, the connectivity matrix has entry $\mathbb{C}(s,t) = W(t,s) - W(s,t) = 0$. Therefore, the connectivity matrix cannot distinguish the test from the absence of a test. As a result, Proposition 2.3 often is not applicable and a proof for unreachability of $c_1$ fails. Indeed, the BVS does not change for program $c_0; c_1$ where the latter command is reachable.

To strengthen the verification approach, we refine the set of constraints in BVS. We add inequalities that reflect the trap property: all initially marked traps have to remain marked in the marking that solves the mixed programming problem. The resulting *enhanced verification system* is sensitive to guards.

To incorporate traps, we construct for a given marking $M$ a *trap inequality*. It has a rational solution if and only if $M$ satisfies the trap property. Note that it is not obvious how to check a universal quantifier (*all* initially marked traps remain marked in $M$) by means of feasibility (*there is* a solution to the trap inequality). The idea is to state the reverse. We set up a constraint system that is *feasible* iff *there is a trap* for which $M$ violates the trap property. Then we use Farkas' lemma to capture by means of feasibility the negation of this statement: *for all traps $M$* satisfies the trap property. We briefly explain the steps in our construction.

1. We exploit the linear algebraic characterization of traps to set up a system of inequalities. This so-called primal system is feasible if and only if $M$ violates the trap property for some trap.

2. By Farkas' lemma we then construct a *dual system of inequalities* that is feasible if and only if the primal system is infeasible. Together with the first statement, the dual system is thus feasible if and only if $M$ satisfies the trap property for all traps.
3. In combination with the marking equation, $M$ becomes variable which leads to non-linearity of the resulting constraints. We manipulate the constraint system to deal with this.

The primal system is a reformulation of the trap property.

**Definition 2.12 (Primal system).** Consider Petri net $N = (S, T, W, M_0)$ with trap matrix $\mathbb{C}_Q \in \mathbb{Z}^{|S| \times |S||T|}$. Let $M \in \mathbb{N}^{|S|}$ be some marking. The *primal system* is

$$\begin{aligned} &\text{Variables: } Y \text{ rational} \\ Y^T \cdot \mathbb{C}_Q &\geq 0 \\ Y &\geq 0 \\ Y^T \cdot M_0 &> 0 \\ Y^T \cdot M &= 0. \end{aligned}$$

By the first two inequalities, $Y$ forms a trap. The strict inequality then requires $Y$ to be initially marked, and the equality finds $Y$ unmarked at $M$. As a result, $M$ violates the trap property for $Q_Y$ from Proposition 2.2.

**Lemma 2.8.** *The primal system is feasible if and only if $M$ violates the trap property.*

For the second phase of our construction, we briefly recall Farkas' lemma. Certain systems of inequalities, so-called *primal systems*, have a *dual system* that enjoys the following equivalence. The primal system is infeasible if and only if the dual system is feasible.

**Lemma 2.9 (Farkas 1894).** *One and only one of the following linear programming problems is feasible:*

$$\begin{array}{ccc} \textit{Variables: } X \textit{ rational} & \qquad & \textit{Variables: } Y \textit{ rational} \\ A \cdot X \leq B & & Y^T \cdot A \geq 0 \\ X \geq 0 & & Y^T \cdot B < 0 \\ & & Y \geq 0. \end{array}$$

The system from Definition 2.12 is not quite in the form on the right hand side. We apply several transformations to obtain an equivalent constraint system of the required shape. Equivalent here means that the solutions do not change. To begin with, note that $M \in \mathbb{N}^{|S|}$ and thus $M \geq 0$. Moreover, we require $Y \geq 0$. Hence, we have $Y^T \cdot M = 0$ if and only if $Y^T \cdot M \leq 0$. Changing the signs inverts the inequality, i.e., $Y^T \cdot M \leq 0$ holds if and only if $Y^T \cdot (-M) \geq 0$. We treat $M_0$ similarly and rewrite the system from Definition 2.12 to

$$\text{Variables: } Y \text{ rational}$$
$$\begin{aligned} Y^T \cdot \mathbb{C}_Q &\geq 0 \\ Y &\geq 0 \\ Y^T \cdot (-M_0) &< 0 \\ Y^T \cdot (-M) &\geq 0. \end{aligned}$$

A last step in constructing the desired shape is to extend $\mathbb{C}_Q$ by a column for $-M$, denoted by $(\mathbb{C}_Q \quad -M)$. This summarize the first and the last inequality. Indeed, we have $Y^T \cdot \mathbb{C}_Q \geq 0$ and $Y^T \cdot (-M) \geq 0$ if and only if $Y^T \cdot (\mathbb{C}_Q \quad -M) \geq 0$.

$$\text{Variables: } Y \text{ rational}$$
$$\begin{aligned} Y^T \cdot (\mathbb{C}_Q \quad -M) &\geq 0 \\ Y^T \cdot (-M_0) &< 0 \\ Y &\geq 0. \end{aligned}$$

To this system, we apply Farkas' lemma.

**Definition 2.13 (Dual system).** Given Petri net $N$ with trap matrix $\mathbb{C}_Q \in \mathbb{Z}^{|S| \times |S||T|}$ and a marking $M \in \mathbb{N}^{|S|}$, the *dual system* is

$$\text{Variables: } X \text{ rational}$$
$$\begin{aligned} (\mathbb{C}_Q \quad -M) \cdot X &\leq -M_0 \\ X &\geq 0. \end{aligned}$$

Combining Lemma 2.8 with Farkas' lemma immediately shows:

**Lemma 2.10.** *The dual system is feasible if and only if $M$ satisfies the trap property: all initially marked traps remain marked at $M$.*

Up to now, $M$ was assumed constant. The goal of the enhanced verification system, however, is to overapproximate all reachable markings that satisfy the trap property. To this end, we combine the dual system with the marking equation. The problem in this construction is in the product $(-M) \cdot X$ that is non-linear, and hence out of scope for linear programming techniques. The solution is again to manipulate the constraint system. We turn to the technicalities of the third phase.

Since $-M$ is added to the trap matrix $\mathbb{C}_Q \in \mathbb{Z}^{|S| \times |S||T|}$, the dimension of $X$ is $|S||T| + 1$. Hence, vector $X$ is the composition $(X' \quad x')^T$ with $X' \in \mathbb{Q}^{|S||T|}$ and $x' \in \mathbb{Q}$. The product $(\mathbb{C}_Q \quad -M) \cdot X \leq -M_0$ is thus equivalent to $x'M \geq M_0 + \mathbb{C}_Q \cdot X'$. We rewrite the dual system accordingly:

$$\text{Variables: } X', x' \text{ rational}$$
$$\begin{aligned} x'M &\geq M_0 + \mathbb{C}_Q \cdot X' \\ X' &\geq 0 \\ x' &\geq 0. \end{aligned}$$

Since $M \geq 0$ the system is solvable with $x' = 0$ if and only if there is a solution with $x' > 0$. This allows us to divide the first and the second inequality by $x'$. Note also that $x' > 0$ if and only if $\frac{1}{x'} > 0$:

$$\begin{aligned} &\text{Variables: } X', x' \text{ rational} \\ &M \geq \tfrac{1}{x'}\, M_0 + \mathbb{C}_Q \cdot (\tfrac{1}{x'}\, X') \\ &\tfrac{1}{x'}\, X' \geq 0 \\ &\tfrac{1}{x'} > 0. \end{aligned}$$

If we set $\frac{1}{x'}$ to be the rational variable $z$ and use $Z$ for $\frac{1}{x'}\, X'$, we obtain the desired trap inequality.

**Definition 2.14 (Trap inequality).** Consider Petri net $N = (S, T, W, M_0)$ with trap matrix $\mathbb{C}_Q \in \mathbb{Z}^{|S| \times |S||T|}$. Let $M \in \mathbb{N}^{|S|}$ be a vector. The *trap inequality* is

$$\begin{aligned} &\text{Variables: } Z, z \text{ rational} \\ &M \geq z\, M_0 + \mathbb{C}_Q \cdot Z \\ &Z \geq 0 \\ &z > 0. \end{aligned}$$

**Proposition 2.4.** *Consider Petri net $N$ and $M \in \mathbb{N}^{|S|}$. Marking $M$ satisfies the trap property if and only if the trap inequality is feasible.*

We are now prepared to combine the trap inequality with the basic verification system from Definition 2.11 to a mixed programming problem.

**Definition 2.15 (Enhanced verification system).** Let Petri net $N$ have connectivity matrix $\mathbb{C} \in \mathbb{Z}^{|S| \times |T|}$ and trap matrix $\mathbb{C}_Q \in \mathbb{Z}^{|S| \times |S||T|}$. Moreover, let $\mathscr{P}$ be a co-linear property on $N$ defined by $A \cdot X \geq B$ with $A \in \mathbb{Q}^{k \times |S|}$ and $B \in \mathbb{Q}^k$ for some $k \in \mathbb{N}$. The associated *enhanced verification system (EVS)* is

$$\text{Variables: } M, X \text{ integer} \quad Z, z \text{ rational}$$

$$\begin{aligned} M &= M_0 + \mathbb{C} \cdot X \\ M, X &\geq 0 \end{aligned} \tag{2.3}$$

$$\begin{aligned} M &\geq z\, M_0 + \mathbb{C}_Q \cdot Z \\ Z &\geq 0 \\ z &> 0 \end{aligned} \tag{2.4}$$

$$A \cdot M \geq B. \tag{2.5}$$

Equality 2.3 is the marking equation. It states that $M$ is reachable from $M_0$ via Parikh vector $X$. The trap inequality is given as 2.4. By Proposition 2.4 it holds for $M$ iff all initially marked traps remain marked in $M$. Therefore, the enhanced verification system is a more precise approximation to the Petri net's state space than BVS. By definition of co-linearity, the last Inequality 2.5 captures a violation to the property.

Since we overapproximate the state space, checking EVS for infeasibility proves correctness. Phrased differently, the analysis is sound.

**Theorem 2.3.** *Consider a Petri net $N$ and a co-linear property $\mathscr{P}$. If the associated enhanced verification system is infeasible, then $\mathscr{P}$ holds for $N$.*

Mixed programming only solves non-strict inequalities $z \geq 0$ and thus cannot handle $z > 0$ in Inequality 2.4. To overcome this problem, the idea is to use the objective function. We relax EVS to $z \geq 0$ and look for a solution that maximizes $z$. Then EVS is infeasible if and only if the optimal solution is $z = 0$.

# Chapter 3
# Unfoldings

**Abstract** Partial order representations of Petri net state spaces.

When linear algebraic verification techniques fail, we have to analyse the Petri net's state space. We develop here a compact representation of these state spaces, called a *finite and complete unfolding prefix*. We also provide suitable operations to evaluate analysis problems like reachability of marking on such prefixes.

The key idea of unfoldings is to store markings as distributed objects, so-called *cuts*. With this distribution, we can determine the effect of transitions locally, i.e., we only change the marking of the surrounding places. The difference to rechability graphs is remarkable. There, a transition firing always yields an overall new marking, even if the token count is changed only in one place.

From a computational complexity point of view, unfolding prefixes trade size for computational hardness of analysis problems like reachability. Indeed, in terms of size and hardness unfolding prefixes lie in between the original Petri net and its reachability graph. The unfolding prefix is larger than the Petri net but more compact than the reachability graph, often exponentially more succinct. As a result, reachability becomes easier for unfoldings than for Petri nets: NP-complete in the size of the unfolding in contrast to PSPACE-complete in the size of the Petri net. In turn, the problem is NL-complete in the size of a given reachability graph.

Technically, unfolding prefixes are themselves Petri nets that have a simpler structure than the original net. They are acyclic and forward branching, i.e., places have a unique input transition. This ease in structure justifies NP-completeness. In fact, on unfolding prefixes reachability queries can be answered by means of off-the-shelf SAT-solvers.

The unfolding is also interesting from a semantical point of view. It preserves more information about the behaviour of the original net than the reachability graph does. It makes explicit *causal dependencies* between transitions, *conflicts* that arise from competitions about tokens, and finally the independence of transitions, also known as *concurrency*. This information is lost in the reachability graph. Indeed, from an unfolding prefix, one can recompute the reachability graph. The reverse does not hold as long as we only take the graph structure into account.

One intuition to the definition of unfoldings stems from finite automata. One can unwind a finite automaton into a computation tree as is done in Algorithm 1.1. This unwinding can be stopped at any moment, yielding different trees. However, if we continue the process with a fair selection of transitions, e.g., by choosing a breadth first processing, then we obtain a unique usually infinite tree. Unfoldings mimick this procedure. To unroll the Petri net, the algorithm first adds places for each token in the input marking. Then it generates a copy of each transition that is fired and adds a fresh place for every token that is produced. If the process is continued as long as enabled transitions exist, the result is a unique structure similar to the computation tree. It is called the *unfolding* of the Petri net. The unfolding is typically infinite, stopping it earlier yields an *unfolding prefix*. The main contrubtion of this section is an algorithm that determines a *finite* prefix of the unfolding that is *complete*. This means the algorithm stops unrolling so that the resulting prefix is finite but yet contains all information about the full unfolding.

## 3.1 Branching Processes

The following definition will only be applied to acyclic Petri nets.

**Definition 3.1 (Causality, conflict, and concurrency relation).** Let $N = (S,T,W)$ be a Petri net that we consider here as a graph $(S \cup T, W)$. Two vertices $x, y \in S \cup T$ are in *causal relation*, denoted by $x \leq y$, if there is a (potentially empty) path from $x$ to $y$. They are in *conflict relation*, denoted by $x \# y$, if there are distinct transitions $t_1, t_2 \in T$ so that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$ and $t_1 \leq x$ and $t_2 \leq y$. The vertices $x$ and $y$ are called *concurrent*, denoted by $x\ co\ y$, if neither $x \leq y$ nor $y \leq x$ nor $x \# y$.

The subclass of Petri nets used for unfolding is the following.

**Definition 3.2 (Occurrence nets).** An *occurrence net* is a Petri net $O = (B,E,G)$ with places $B$, transitions $T$, and weight function $G : (B \times E) \cup (E \times B) \to \{0,1\}$ that satisfies the following constraints.

**(O1)** $O$ is *acyclic*.
**(O2)** $O$ is *finitely preceeded*: the set $\{y \in B \cup E \mid y \leq x\}$ is finite for all $x \in B \cup E$.
**(O3)** $O$ is *forward branching*: for all $b \in B$ we have $|{}^\bullet b| = 1$.
**(O4)** $O$ is *free from self conflicts*: for all $y \in B \cup E$ we do not have $y \# y$.

We assume the existence of a unique $\leq$-minimal element $e_\perp \in E$. The $\leq$-minimal places are denoted by $Min(O)$.

In occurrence nets, places are typically called *conditions* and transitions are called *events*. Note that by the requirement for acyclicity $(B \cup E, \leq)$ is a partial order.

**Lemma 3.1.** *Consider an occurrence net $O = (B, E, G)$. For two vertices $x, y \in B \cup E$ one and only one of the following holds: $x = y$, $x < y$, $y < x$, $x \# y$, or $x$ co $y$.*

We are interested in occurrence nets that result from unrolling the original Petri net. We establish the relationship between the two by labelling the occurrence net with the places and transitions of the original net.

**Definition 3.3 (Folding homomorphism, branching process).** Let $O = (B, E, G)$ be an occurrence net and $N = (S, T, W, M_0)$. A *folding homomorphism from $O$ to $N$* is a mapping $h : B \cup (E \setminus \{e_\perp\}) \to S \cup T$ that satisfies the following constraints.

**(F1)** Conditions are labelled by places and events represent transitions: $h(B) \subseteq S$ and $h(E \setminus \{e_\perp\}) \subseteq T$.
**(F2)** Transition environments are preserved: $h(e^\bullet) = h(e)^\bullet$ and $h(^\bullet e) = {}^\bullet h(e)$.
**(F3)** Minimal elements represent the initial marking: $h(Min(O)) = M_0$.
**(F4)** No redundancy: for all $e, f \in E$ with $^\bullet e = {}^\bullet f$ and $h(e) = h(f)$ we have $e = f$.

The pair $(O, h)$ is a *branching process of $N$*.

The auxiliary event $e_\perp \in E$ is not mapped. It helps us shorten formal statements about unfoldings, but has no semantical meaning when relating an occurrence net to the original Petri net.

Branching processes differ in how much they unfold the original Petri net. The prefix relation captures this notion of *unfolding more than* in a formal way.

**Definition 3.4 (Prefix relation).** Consider two branching processes $(O, h)$ with $O = (B, E, G)$ and $(O', h')$ with $O' = (B', E', G')$. Then $(O, h)$ is a *prefix of* $(O', h')$, denoted by $(O, h) \sqsubseteq (O', h')$, if $O$ is a subnet of $O'$ and the following holds.

**(P1)** If $b \in B$ and $(e, b) \in G'$ then $e \in E$.
**(P2)** If $e \in E$ and $(b, e) \in G'$ or $(e, b) \in G'$ then $b \in B$.
**(P3)** We have $h = h' \cap (B \cup E)$.

By our requirement on a unique minimal element, we find $e_\perp$ in both $E$ and $E'$. The notion of subnet is defined by inclusion, $B \subseteq B'$, $E \subseteq E'$, and $G = G' \cap ((B \times E) \cup (E \times B))$. Requirement **(P1)** states that the predecessors of conditions in $O$ according to $O'$ have to be in $O$. For events, **(P2)** requires that we preserve the full environment of conditions in the pre- and in the postset. Finally, **(P3)** states that the labelling of $O$ is the labelling of $O'$ restricted to the conditions and events in $O$.

The unfolding is a branching process that unrolls the given Petri net as much as possible — a procedure that usually does not terminate. The proof that this object is unique is out of the scope of the techniques we discuss in this lecture.

**Theorem 3.1 (Engelfriet 1991).** *Every Petri net $N$ has an up to isomorphism (renaming of conditions and events) unique and $\sqsubseteq$-maximal branching process. It is called* the unfolding of *$N$ and denoted by $Unf(N)$.*

The unfolding keeps the initial marking in terms of $\leq$-minimal conditions. It also has a representative for each transition that occurs in a firing sequence. Therefore, intuitively the reachable markings in the unfolding should coincide, via the folding homomorphism, with the reachable markings of the original Petri net. Since the unfolding is an infinite but unmarked Petri net with a distinguished transition $e_\perp \in E$, we have to first have to define the notion of reachability for $\mathit{Unf}(N)$. We assume that every minimal condition carries precisely one token. Transition $e_\perp$ never executes. All remaining transitions fire as it is defined for finite Petri nets.

**Theorem 3.2 (Engelfriet 1991).** *Let $\mathit{Unf}(N) = (O,h)$ with $O = (B,E,G)$. We have $R(N) = h(R(\mathit{Unf}(N)))$. Moreover, for $M_1, M_2 \in R(\mathit{Unf}(N))$ and all $e \in E$ we have $M_1[e\rangle M_2$ if and only if $h(M_1)[h(e)\rangle h(M_2)$.*

## 3.2 Configurations and Cuts

The result stated above understands the unfolding as a Petri net. It relies on the classical sequential semantics defined in terms of transition sequences as they are represented in interleaving structures like the reachability graph. But this view does not take the partial order of events into account. In an unfolding, the counterpart of a transition sequence is called a *configuration*. A configuration is a set of events that usually allows for different sequential executions. This means a single configuration reflects multiple transition sequences in the unfolding and, with Theorem 3.2, also in the Petri net. Configurations are at the heart of why unfolding-based approaches to verification scale well with an increasing degree of concurrency, whereas reachability graph exploration suffers from the state space explosion problem.

**Definition 3.5 (Configuration).** A *configuration of* $(O,h)$ with $O = (B,E,G)$ is a non-empty set $C \subseteq E$ of events that is

**C1** *causally closed*: if $f \in C$ and $e \leq f$ then $e \in C$ and
**C2** *conflict free*: for all $e, f \in C$ we do not have $e \# f$.

By $\mathscr{C}_{\mathit{fin}}(O,h)$ we denote the set of all *finite configurations* of $(O,h)$.

Transition sequences lead to markings. For configurations, the analogue is called a *cut* of the branching process.

**Definition 3.6 (Cut).** Consider $(O,h)$ with $O = (B,E,G)$. A set $B' \subseteq B$ of conditions is *concurrent* if $b_1$ *co* $b_2$ for all $b_1, b_2 \in B'$. A *cut* is an $\subseteq$-maximal concurrent set.

The relationship between cuts and markings is again given via folding.

**Lemma 3.2 (and definition).** *Let $(O,h)$ be a branching process of Petri net $N$ and let $C \in \mathscr{C}_{\mathit{fin}}(O,h)$. Then $C^\bullet \setminus {}^\bullet C$ is a cut, denoted by $\mathit{Cut}(C)$. The* final marking of $C$ *is $\mathit{Mark}(C) := h(\mathit{Cut}(C))$. A marking is said to be* represented in $(O,h)$ *if there is a configuration $C \in \mathscr{C}_{\mathit{fin}}(O,h)$ with $M = \mathit{Mark}(C)$.*

A transition sequence $M_0[\sigma\rangle M$ yields a finite configuration $C$ in the unfolding that represents the marking, $M = \mathit{Mark}(C)$. In turn, every configuration can be linearized to a transition sequence. As a result, final markings are reachable.

**Lemma 3.3.** *Every $M \in R(N)$ is represented in $Unf(N)$. Every marking represented in a branching process is reachable.*

**Definition 3.7 (Extension).** Given configuration $C \in \mathscr{C}_{fin}(O,h)$ and set of events $E$. We denote by $C \oplus E$ the fact that $C \cup E$ is a configuration and $C \cap E = \emptyset$. We call $C \oplus E$ the *extension* of $C$. Moreover, $E$ is also called the *suffix* of $C$.

**Lemma 3.4.** *If $C \subsetneq C'$ then there is a non-empty suffix $E$ of $C$ so that $C \oplus E = C'$.*

## 3.3 Finite and Complete Prefixes

We study algorithmic aspects related to unfoldings. To this end, we first develop a *data structure for branching processes*. Consider branching process $(O,h)$ with $O = (B,E,G)$ of Petri net $N = (S,T,W,M_0)$. We represent $(O,h)$ as a list $\{n_1,\dots,n_k\}$ of nodes. The list contains both, conditions and events. More precisely, a condition $b \in B$ yields a record node $b = (s,e)$. It contains the place $s \in S$ that labels $b$, which means $h(b) = s$. Moreover, $e$ is the input event of $b$, ${}^\bullet b = \{e\}$. Events $e \in E$ are stored similarly as record nodes $e = (t,X)$ with $h(e) = t$ and ${}^\bullet e = X \subseteq B$. So again the first entry is the label and the second entry is a set of pointers to the conditions in the preset. Note that the list representation contains the weight function as well as the labelling. This means we can use $(O,h)$ and $\{n_1,\dots,n_k\}$ interchangably.

We describe the events that can be added to a branching process.

**Definition 3.8 (Possible extensions).** Let $(O,h)$ with $O = (B,E,G)$ be a branching process of Petri net $N$. A pair $(t,X)$ with $t \in T$ and $X \subseteq B$ is a *possible extension of* $(O,h)$ if $h(X) = {}^\bullet t$ and $(t,X)$ does not already belong to $(O,h)$. We denote by $Pe(O,h)$ the set of possible extensions of $(O,h)$.

**Lemma 3.5.** *Let $(O,h) = \{n_1,\dots,n_k\}$ be a branching process of Petri net $N$. Let $t \in T$ have postset $t^\bullet = \{s_1,\dots,s_n\}$. If $e = (t,X)$ is a possible extension of $(O,h)$ then $\{n_1,\dots,n_k,e,(s_1,e),\dots,(s_n,e)\}$ is a branching process of $N$.*

The algorithm to compute the unfolding is given in Figure 3.1. The procedure is initialized with the minimal conditions. It keeps adding possible extensions together with their outputs as long as there are some. The unfolding computation terminates if and only if $N$ terminates, i.e., the net does not enable an infinite run. Moreover, for correctness of the procedure we have to impose the following fairness requirement: every event $e \in pe$ is eventually chosen to extend the unfolding.

### *3.3.1 Constructing a finite and complete prefix*

For algorithmic analyses, we require a finite object that allows for an exhaustive analysis. We now construct a *finite prefix* $(O,h)$ of the unfolding of $N$ that is still

$Unf := \{e_\perp, (s_1, e_\perp), \ldots, (s_n, e_\perp)\}$
$pe := Pe(Unf)$
**while** $pe \neq \emptyset$ **do**
  **add to** *Unf* event $e = (t, X) \in pe$
  **add to** *Unf* condition $(s, e)$ **for all** $s \in t^\bullet$
  $pe := Pe(Unf)$
**end while**

**Fig. 3.1** Unfolding procedure.

*complete*: it contains as much information as $Unf(N)$. Technically, the notion of completeness that we rely on is the following.

**Definition 3.9 (Complete Prefix).** Let $N = (S, T, W, M_0)$ be a Petri net and $(O, h)$ one of its branching processes. We say $(O, h)$ is *complete* or *a complete prefix of* $Unf(N)$ if for all $M \in R(N)$ there is a configuration $C \in \mathscr{C}_{fin}(O, h)$ so that

- $Mark(C) = M$ and
- for all $t \in T$ with $M[t\rangle$ there is $C \oplus \{e\} \in \mathscr{C}_{fin}(O, h)$ with $h(e) = t$.

The first requirement states that every marking $M$ reachable in the Petri net is represented by a configuration $C$ in the complete prefix. The second requirement asks this configuration to also preserve the transitions. If $t \in T$ is enabled in $M$, then a corresponding event can be appended to the configuration without leaving the complete prefix. Note that a marking may be represented by several configurations, but only one of them needs to reflect the transition environment.

Note that the unfolding can be reconstructed from a complete prefix. Indeed, by definition all markings together with their firings are present in this smaller object. It can be shown that the preservation of reachable markings themselves is not sufficient to obtain the unfolding, simply because some transitions may be forgotten if there are several paths leading to a marking.

The key observation to the theory that we develop is the following. Since Petri net $N$ is assumed to be safe, it has finitely many reachable markings. This means, the unfolding eventually starts repeating markings. Therefore, intuitively it should contain a complete prefix that is yet *finite*. We give a procedure for computing such a finite and complete unfolding prefix. We reuse the procedure in Figure 3.1 but identify events at which the computation can be stopped without loosing information. These events are called *cut-offs* and their detection is at the heart of the unfolding theory.

# Chapter 4
# Coverability

**Abstract** Decidability of coverability

We develop a decision procedure for the *coverability problem*. The problem takes as input a Petri net $N$ and a marking $M \in \mathbb{N}^S$. The question is whether there is an $M' \in R(N)$ that dominates $M$, $M' \geq M$. If the state space of the Petri net is finite, an immediate solution is to compute the reachable states and look for a covering marking $M'$. If the state space is infinite, however, the problem is non-trivial. The reachability graph cannot be used for the analysis as it is no longer finite. Moreover, also an analysis by means of linear algebraic techniques may fail.

## 4.1 Coverability Graphs

The solution is to define a *finite* structure, the so-called coverability graph of a Petri net, that an algorithm can analyze exhaustively. Coverability graph are similar to reachability graphs in that they reflect the firing of transitions along markings. But different from reachability graphs, coverability graphs may abstract away the precise token count. They use entries $\omega$ in a marking to indicate that a place may carry arbitrarily many tokens.

Technically, we first generalize the natural numbers $\mathbb{N}$ to $\mathbb{N}_\omega := \mathbb{N} \cup \{\omega\}$. With the number of tokens in mind, the new element $\omega$ stands for *unbounded*. To extend the operations $<$ and $+$ to $\mathbb{N}_\omega$, we set

$$m < \omega \quad \text{and} \quad \omega + m := \omega =: \omega - m \qquad \text{for all } m \in \mathbb{N}.$$

We do not define the subtraction $\omega - \omega$ and will not need it for the development in this section.

**Definition 4.1 (Generalized marking).** Consider Petri net $N = (S, T, W, M_0)$. The set of *generalized markings* is $\mathbb{N}_\omega^S$. For every marking $M_\omega \in \mathbb{N}_\omega^S$, we denote by $\Omega(M_\omega) := \{s \in S \mid M_\omega(s) = \omega\}$ the set of places marked $\omega$. The operations on $\mathbb{N}_\omega^S$ are taken componentwise, so also the following notions are defined:

$$\begin{aligned} M_\omega[t\rangle \qquad & \text{if } M_\omega \geq W(-,t) \\ M_\omega[t\rangle M'_\omega \qquad & \text{if } M_\omega \geq W(-,t) \quad \text{and} \quad M'_\omega = M_\omega - W(-,t) + W(t,-). \end{aligned}$$

Note that firing a transition does not remove $\omega$-entries. This means, $M_\omega(s) = \omega$ and $M_\omega[t\rangle M'_\omega$ implies $M'_\omega(s) = \omega$ for all $M_\omega, M'_\omega \in \mathbb{N}_\omega^S$, $s \in S$, and $t \in T$.

The coverability graph is computed (and defined) by the algorithm in Figure 4.1. It introduces $\omega$ whenever a path strictly increases the token count. To make the outcome of the computation deterministic, we use a FIFO buffer for the work list and an ordering on the transitions. Without this restriction, the resulting coverability graph would depend on the processing order for markings and transitions.

**Lemma 4.1 (Finiteness).** *For every Petri net $N$, $Cov(N)$ is finite.*

The proof bears similarities to the decision procedure for boundedness discussed in Section 1.2. To turn coverability graphs into a decision procedure for coverability, we need an equivalence of the following form. Marking $M$ is coverable in $N$ if and only if there is $M_\omega \in Cov(N)$ with $M \leq M_\omega$. The next lemmas provide the required implications.

**Lemma 4.2 (From $N$ to $Cov(N)$).** *Consider Petri net $N = (S, T, W, M_0)$ and a transition sequence $\sigma \in T^*$ with $M_0[\sigma\rangle M$ for some $M \in R(N)$. Then there is a $\sigma$-labelled path $M_0 \xrightarrow{\sigma} M_\omega$ in $Cov(N)$ that leads to $M_\omega \geq M$.*

Thus, if a marking $M$ is coverable in $N$ then there is a larger marking $M_\omega \geq M$ in the coverability graph. The following lemma states the reverse. Larger markings in the coverability graph indeed indicate coverability in $N$.

**Lemma 4.3 (From $Cov(N)$ to $N$).** *Consider Petri net $N = (S, T, W, M_0)$. For every $M_\omega \in Cov(N)$ and every $k \in \mathbb{N}$ there is a marking $M \in R(N)$ with $M(s) \geq k$ for all $s \in \Omega(M_\omega)$ and $M(s) = M_\omega(s)$ for all $s \in S \setminus \Omega(M_\omega)$.*

The lemma states that the number of tokens on $\omega$-marked places can exceed any bound $k \in \mathbb{N}$. The remaining places receive the exact token count as it is required by the given marking $M_\omega$. The proof exploits the fact that sequences which introduce $\omega$-entries in the coverability graph can be repeated arbitrarily. Consider

$$M_0 \xrightarrow{\tau} M_\omega^1 \xrightarrow{\sigma} M_\omega^2 \qquad \text{with} \qquad M_\omega^1 \lneq M_\omega^2.$$

By repeating $\sigma$, an arbitrary token count can be generated on the places $s \in S$ with $M_\omega^1(s) < M_\omega^2(s)$. The proof is by induction on the length of the shortest path leading to $M_\omega$. It requires some effort in case a new $\omega$ is introduced in the induction step.

**input** : $N = (S, T, W, M_0)$

**begin**
  $V := \{M_0\}$ //Set of vertices in the coverability graph
  $L := M_0$ //Work list of vertices to be processed
  $E := \emptyset$ //Set of edges in the coverability graph
  **while** $L \neq \emptyset$ **do**
    **let** $L = M^1_\omega.L'$
    $L := L'$
    **for all** $t = t_1, \ldots t_n \in T$ with $M^1_\omega[t\rangle$ **do** //Process the enabled transitions in order
      $M^2_\omega := \tilde{M}^2_\omega$ where $M^1_\omega[t\rangle\tilde{M}^2_\omega$
      **for all** $M_\omega$ on a path from $M_0$ to $M^1_\omega$ that satisfy $M_\omega \lneq \tilde{M}^2_\omega$ **do**
        $M^2_\omega(s) := \omega$ for all $s \in S$ with $M_\omega(s) < \tilde{M}^2_\omega(s)$
      **end for all**
      **if** $M^2_\omega \notin V$ **then**
        $V := V \cup \{M^2_\omega\}$
        $L := L.M^2_\omega$
      **end if**
      $E := E \cup \{(M^1_\omega, t, M^2_\omega)\}$
    **end for all**
  **end while**
**end**

**output** : $Cov(N) := (V, E, M_0)$ the *coverability graph of* $N$.

**Fig. 4.1** Coverability graph computation.

**Theorem 4.1 (Decision procedure for coverability and place boundedness).** *Given Petri net* $N = (S, T, W, M_0)$.

1. *Marking* $M \in \mathbb{N}^S$ *is coverable if and only if there is* $M_\omega$ *in* $Cov(N)$ *with* $M_\omega \geq M$.
2. *Place* $s \in S$ *is unbounded if and only if there is* $M_\omega$ *in* $Cov(N)$ *with* $M_\omega(s) = \omega$.

# Part II
# Network Protocols and Lossy Channel Systems

Network protocols define the interaction among finite state components that communicate asynchronously by package transfer. We introduce a corresponding model of lossy channel systems and investigate algorithms for the automatic verification of network protocols. Decidability of the analysis follows from monotonicity of the models' behaviour with respect to an ordering on the configurations. We extend this insight towards a theory of well structured transition systems.

# Chapter 5
# Introduction to Lossy Channel Systems

**Abstract** Lossy Channel Systems

Lossy channel systems (LCS) formalize network protocols like the alternating bit protocol or more general sliding window protocols that are located at the data link layer of the ISO OSI reference model. In recent developments, LCS have also proven adequate for modelling programs running on relaxed memory models like total store ordering used in x86 processors.

Technically, LCS are finite state programs that communicate via asynchronous message transfer over unbounded FIFO channels. Without restrictions, such a model of channel systems is Turing complete. Channels immediately reflect the tape of a Turing machine. The restriction we impose is inspired by the following observation about the application domain of our analysis. Network protocols are designed to operate correctly in the presence of package loss. Therefore, a weaker model with unreliable channels should be sufficient for their verification. Lossy channel systems formalize unreliability by lossiness: channels may drop packages at any moment. This weakness indeed yields decidability of the resulting model.

## 5.1 Syntax and Semantics

**Definition 5.1 (Lossy Channel Systems).** A *lossy channel system (LCS)* is a tuple $L = (Q, q_0, C, M, \rightarrow)$ where $Q$ is a finite set of *states* with *initial state* $q_0 \in Q$. Moreover, $C$ is a finite set of *channels* over which we transfer *messages* in the finite set $M$. *Transitions* in $\rightarrow \subseteq\ Q \times OP \times Q$ perform *operations* in $OP := C \times \{!, ?\} \times M$.

A transition $(q_1, op, q_2) \in \rightarrow$, typically denoted by $q_1 \xrightarrow{op} q_2$ yields a change in the control state from $q_1$ to $q_2$ while performing operation $op$. A *send operation* $c!a$ in $OP$ appends message $a$ to the current content of channel $c$. A *receive operation* $c?a \in OP$ removes message $a$ from the head of channel $c$. Therefore, the two operations indeed define a FIFO channel.

In our examples, we often represent LCS by several automata. This matches the above formal definition by taking as set of states the Cartesian product of the states in the single automata. The initial state is the tuple of initial states. Every transitions represents the state change in a single automaton.

Like every automaton model, the semantics of LCS relies on a notion of *state at runtime*. For LCS, they are called configurations and should be understood as analogue of markings in Petri nets.

**Definition 5.2 (Configuration).** Let $L = (Q, q_0, C, M, \rightarrow)$. A *configuration of* $L$ is a pair $\gamma = (q, W) \in Q \times M^{*C}$. It consists of a state $q \in Q$ and a function $W \in M^{*C}$ that assigns to each channel $c \in C$ a finite word $W(c) \in M^*$. The *initial configuration of* $L$ is $\gamma_0 := (q_0, \varepsilon)$ where $\varepsilon$ assigns the empty word ($\varepsilon$) to every channel.

Transitions change the channel content. We capture this by update operations on vectors of words. Lossiness is formalized by an ordering on configurations. For the definition of this ordering, we first compare words by Higman's subword ordering. It sets $u \preceq^* v$ if $u$ is a not necessarily contiguous subword of $v$. With a componentwise definition, we lift the ordering to vectors of words, $W_1 \preceq^* W_2$. For configurations, we pose the additional requirement that the states coincide.

**Definition 5.3 (Updates and $\preceq$ on configurations).** *Updates* take the form $[c := x]$ with $c \in C$ and $x \in M^*$. They are applied to channel contents $W \in M^{*C}$. The result of this application is a new content $W[c := x] \in M^{*C}$ defined by $W[c := x](c) := x$ and $W[c := x](c') := W(c')$ for all $c' \neq c$ with $c' \in C$.

For the definition of the *subword ordering* $\preceq^* \subseteq M^* \times M^*$, let $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$ in $M^*$. We have $u \preceq^* v$ if there are indices $1 \leq i_1 < \dots < i_m \leq n$ with $u_j = v_{i_j}$ for all $1 \leq j \leq m$. For $W_1, W_2 \in M^{*C}$, we set $W_1 \preceq^* W_2$ if $W_1(c) \preceq^* W_2(c)$ for all $c \in C$. Finally, for configurations $(q_1, W_1), (q_2, W_2) \in Q \times M^{*C}$ we have $(q_1, W_1) \preceq (q_2, W_2)$ if $q_1 = q_2$ and $W_1 \preceq^* W_2$.

The semantics of LCS is defined in terms of transitions between configurations.

**Definition 5.4 (Transition relation between configurations).** Consider the LCS $L = (Q, q_0, C, M, \rightarrow)$. It defines a transition relation $\rightarrow \subseteq (Q \times M^{*C}) \times (Q \times M^{*C})$ between configurations as follows:

$$\begin{aligned} (q_1, W) &\rightarrow (q_2, W[c := W(c).m]) && \text{if } q_1 \xrightarrow{c!m} q_2 \\ (q_1, W[c := m.W(c)]) &\rightarrow (q_2, W) && \text{if } q_1 \xrightarrow{c?m} q_2 \\ \gamma_1' &\rightarrow \gamma_2' && \text{if } \gamma_1' \succeq \gamma_1 \rightarrow \gamma_2 \succeq \gamma_2' \end{aligned}$$

for some configurations $\gamma_1, \gamma_2 \in Q \times M^{*C}$.

For a lossy transition $(q_1, W_1') \to (q_2, W_2')$ that is derived with the third condition, we already have a transition $(q_1, W_1) \to (q_2, W_2)$ with $W_1' \succeq^* W_1$ and $W_2 \succeq^* W_2'$. Intuitively, the messages in $W_1'$ but outside $W_1$ are lost immediately before the transition and the messages in $W_2$ but outside $W_2'$ are lost immediately afterwards.

Interestingly, for LCS the notions of reachability and coverability coincide. However, as refer to coverability in the context of well structured transition systems, we define both notions.

**Definition 5.5 (Reachability and Coverability).** Let $L = (Q, q_0, C, M, \to)$ be an LCS and $\gamma_1, \gamma_2 \in Q \times M^{*C}$. We say $\gamma_2$ *is reachable from* $\gamma_1$ if $\gamma_1 \to^* \gamma_2$. The set of *all configurations reachable from* $\gamma_1$ is $R(\gamma_1) := \{\gamma \in Q \times M^{*C} \mid \gamma_1 \to^* \gamma\}$. We denote the *reachable configurations of* $L$ by $R(L) := R(\gamma_0)$. Configuration $\gamma_2$ is *coverable from* $\gamma_1$ if there is $\gamma \in R(\gamma_1)$ with $\gamma \succeq \gamma_2$.

# Chapter 6
# Well Structured Transition Systems

**Abstract** Well Structured Transition Systems

## 6.1 Well Quasi Orderings

In computer science, quasi orderings that are not partial orderings result from syntactically different representations of semantically equivalent elements. To give an example, let $\leq$ denote language inclusion. Then the regular expressions $a+b$ and $b+a$ can be ordered by $a+b \leq b+a$ as well as $b+a \leq a+b$. The terms, however, do not coincide.

Formally, a *quasi ordering (qo)* is a reflexive and transitive relation $\leq \subseteq A \times A$. We also call the pair $(A, \leq)$ a quasi ordering. In a qo, we write $a > b$ for $a \geq b$ and $b \not\geq a$. Note that $a \geq b$ and $b \geq a$ need not imply $a = b$. In this case, $\leq$ is called a *partial ordering*. In the theory of well structured transition systems, so-called well quasi orderings (wqos) play a key role. In a wqo, every infinite sequence contains two comparable elements.

**Definition 6.1 (Well quasi ordering).** A qo $(A, \leq)$ is a *well quasi ordering (wqo)* if for every infinite sequence $(a_i)_{i\in\mathbb{N}}$ in $A$ there are indices $i < j$ with $a_i \leq a_j$.

We exploit the unavoidability of repetitions to establish termination of verification algorithms. Indeed, classical termination proofs rely on well founded relations that decrease with every transition. Recall that a quasi ordering $(A, \leq)$ is *well founded* if it does not contain infinite sequences $(a_i)_{i\in\mathbb{N}}$ that strictly decrease, $a_0 > a_1 > \ldots$ Wqos additionally impose the absence of antichains. An *antichain* is a set $B \subseteq A$ of incomparable elements, $a \not\leq b$ for all $a, b \in B$.

**Theorem 6.1 (Characterization of wqos).** *Consider the qo $(A, \leq)$. The following statements are equivalent:*

1. *$(A, \leq)$ is a wqo.*
2. *Every infinite sequence $(a_i)_{i\in\mathbb{N}}$ in $A$ contains an infinite non-decreasing subsequence $(a_{\varphi(i)})_{i\in\mathbb{N}}$ with $a_{\varphi(i)} \leq a_{\varphi(i+1)}$ for all $i \in \mathbb{N}$.*
3. *There is no infinite strictly decreasing sequence and no infinite antichain in $A$.*

*Proof.* $(\mathbf{1}) \Rightarrow (\mathbf{2})$ Consider an infinite sequence $(a_i)_{i\in\mathbb{N}}$ in $A$. Take the subsequence $(a_{nd(i)})_{i\in\mathbb{N}}$ of elements that are not dominated by successors, i.e., for all $a_{nd(i)}$ there is no $a_j$ with $nd(i) < j$ and $a_{nd(i)} \leq a_j$. The sequence has to be finite by the well quasi ordering assumption. Let $n := nd(k)$ be the maximal index in this sequence. Starting from $n+1$ one finds an infinite non-decreasing subsequence since every element after $a_n$ is dominated by $(\leq)$ some successor.

$(\mathbf{2}) \Rightarrow (\mathbf{3})$ By definition.

$(\mathbf{3}) \Rightarrow (\mathbf{1})$ Consider an infinite sequence $(a_i)_{i\in\mathbb{N}}$. We show that there are $i < j$ with $a_i \leq a_j$. The idea is to descend strictly decreasing sequences and gather the least elements in an antichain. By the well foundedness assumption and the absence of infinite antichains, the procedure terminates and finds two comparable elements.

Consider the first element $a_0$. If there is a successor $a_j$ with $a_0 \leq a_j$ we are done. Otherwise, we find the first successor $a_{\varphi(1)}$ with $a_0 > a_{\varphi(1)}$. We repeat the argumentation for $a_{\varphi(1)}$. If there is a successor $a_j$ with $a_{\varphi(1)} \leq a_j$, we found two comparable elements. Otherwise, we find the first successor $a_{\varphi(2)}$ with

$$a_0 > a_{\varphi(1)} > a_{\varphi(2)}.$$

The search eventually terminates because there are no infinite strictly decreasing sequences. Let $a_{\varphi(n_0)}$ be the element that has no successor $a_j$ with $a_{\varphi(n_0)} \leq a_j$ and no successor $a_j$ with $a_{\varphi(n_0)} > a_j$. Add $a_{\varphi(n_0)}$ as first element to an antichain.

We proceed with $a_{\varphi(n_0)+1}$. Again we search for $a_{\varphi(n_1)}$ that has no successor $a_j$ with $a_{\varphi(n_1)} \leq a_j$ and no successor $a_j$ with $a_{\varphi(n_1)} > a_j$. By construction

$$a_{\varphi(n_0)} \not\leq a_{\varphi(n_1)} \qquad \text{and} \qquad a_{\varphi(n_0)} \not> a_{\varphi(n_1)}.$$

Hence, the set $\{a_{\varphi(n_0)}, a_{\varphi(n_1)}\}$ is an antichain of size two. Repeating the procedure indefinitely yields an infinite antichain. A contradiction to the assumption that no infinite antichains exists. We have to find $i < j$ with $a_i \leq a_j$. □

A reader familiar with Ramsey's theorem will find a more elegant proof of the last implication $(\mathbf{3}) \Rightarrow (\mathbf{1})$, in fact even of the stronger statement $(\mathbf{3}) \Rightarrow (\mathbf{2})$. Ramsey's theorem considers infinite complete graphs where the edges are labelled by finitely many colors. It states that such a graph contains an infinite complete subgraph that is labelled by a single color. To apply the theorem, note that an infinite sequence $(a_i)_{i\in\mathbb{N}}$ induces the infinite complete graph where the elements $a_i$ are the vertices. The edges are labelled by the relations $\{\leq, >, \mathit{incomparable}\}$. Let $i < j$ and consider the edge between $a_i$ and $a_j$. We label it by $\leq$ if $a_i \leq a_j$. We label it by $>$ if $a_i > a_j$. Otherwise, we label it by *incomparable*. Ramsey's theorem applies and yields an infinite complete subgraph labelled by a single color. By the assumptions, the color cannot be $>$ and not *incomparable*. Hence, we found an infinite non-decreasing subsequence.

## 6.2 Upward and downward closed sets

In wqos, every set $B$ contains finitely many minimal elements $min(B)$. Minimal elements are interesting as they represent, in a precise way, so-called upward closed sets.

**Definition 6.2 (Minimal elements).** Let $(A,\leq)$ be a wqo and let $B\subseteq A$. A *set of minimal elements* is a subset $min(B)\subseteq B$ that contains for every $b\in B$ an element $m\in min(B)$ with $m\leq b$ and that is an antichain.

**Lemma 6.1 (Existence and finiteness of minimal elements).** *Let $(A,\leq)$ be a wqo and $B\subseteq A$. There is a finite set of minimal elements $min(B)$.*

*Proof.* To the contrary, assume there is no finite set of minimal elements. We form an infinite sequence $(b_i)_{i\in\mathbb{N}}$ starting with some $b_0\in B$. As $b_{i+1}$ we choose an element that is no larger than any predecessor, $b_j\not\leq b_{i+1}$ for all $0\leq j\leq i$. Such an element exists, otherwise we can construct a finite set of minimal elements from $\{b_0,\ldots,b_i\}$. The resulting infinite sequence $(b_i)_{i\in\mathbb{N}}$ violates the wqo assumption. ☐

Note that $min(B)$ need not be unique as antisymmetry is missing. With an algorithmic point of view, the lemma can be understood as follows. Sets $min(B)$ of minimal elements are good candidates for finite representations of infinite sets. The sets that can be captured precisely by their minimal elements are upward closed.

**Definition 6.3 (Upward and downward closure).** Let $(A,\leq)$ be a wqo. A set $I\subseteq A$ is *upward closed* if $x\in I$ and $a\geq x$ for $a\in A$ implies $a\in I$. The *upward closure* of a set $B\subseteq A$ is $B\uparrow := \{a\in A \mid a\geq b \text{ for some } b\in B\}$. Similarly, a set $D\subseteq A$ is *downward closed* if $x\in D$ and $a\leq x$ for $a\in A$ implies $a\in D$. The *downward closure* of $B\subseteq A$ is $B\downarrow := \{a\in A \mid a\leq b \text{ for some } b\in B\}$.

**Lemma 6.2 (Representation of upward closed sets by minimal elements).** *Let $(A,\leq)$ be a wqo and consider an upward closed set $I\subseteq A$. Let $min(I)$ be a finite set of minimal elements. Then $I=min(I)\uparrow$.*

The decision procedure for coverability in wsts deals with increasing sequences of upward closed sets. The wqo assumption guarantees that these sequences stabilize, which in turn ensures termination of the algorithm.

**Theorem 6.2 (Chains of upward closed sets stabilize).** *Consider a qo $(A,\leq)$. The following statements are equivalent:*

1. *$(A,\leq)$ is a wqo.*
2. *For every infinite increasing sequence $I_0\subseteq I_1\subseteq I_2\subseteq\ldots$ of upward closed sets $I_j\subseteq A$ there is a $k\in\mathbb{N}$ with $I_k=I_{k+1}$.*

3. *For every infinite increasing sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq \dots$ of upward closed sets $I_j \subseteq A$ there is an $l \in \mathbb{N}$ with $I_l = I_{l+1} = I_{l+2} = \dots$*

*Proof.* (**1**) $\Rightarrow$ (**2**) Towards a contradiction, assume there is an infinite sequence $I_0 \subsetneq I_1 \subsetneq I_2 \subsetneq \dots$ Then there are elements $a_0 \in I_1 \setminus I_0$, $a_1 \in I_2 \setminus I_1$, $a_2 \in I_3 \setminus I_2$, $\dots$ Since the sets $I_j$ are upward closed, we can conclude $a_i \not\leq a_j$ for all $i, j \in \mathbb{N}$ with $i < j$. The sequence $(a_i)_{i\in\mathbb{N}}$ violates the wqo assumption.

(**2**) $\Rightarrow$ (**3**) Again we proceed by contradiction and assume (3) does not hold. This means there is an infinite sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq \dots$ so that for every $k \in \mathbb{N}$ there is $k_1$ with $k < k_1$ and $I_k \subsetneq I_{k_1}$. For $k_1$ there is again a later $k_1 < k_2$ with $I_{k_1} \subsetneq I_{k_2}$ etc. We single out this infinite strictly increasing subsequence.

$$I_k \subsetneq I_{k_1} \subsetneq I_{k_2} \subsetneq \dots$$

By assumption (2), the sequence contains $I_{k_j} = I_{k_{j+1}}$. A contradiction.

(**3**) $\Rightarrow$ (**1**) Consider sequence $(a_i)_{i\in\mathbb{N}}$ in $A$. We define a sequence of upward closed sets: $I_0 := \{a_0\}\uparrow$, $I_1 := \{a_0, a_1\}\uparrow$, etc. Since $I_0 \subseteq I_1 \subseteq \dots$ there is a smallest $l \in \mathbb{N}$ with $I_l = I_{l+1} = \dots$ This means there is $j < l+1$ with $a_j \leq a_{l+1}$. ☐

## 6.3 Constructing well quasi orderings

The importance of well structured transition systems stems from the fact that many sets are well quasi ordered. This in turn is based on the observation that wqos can be composed into new ones. We present an algebraic toolkit to derive the wqos needed in this lecture. The list is not complete. We skip Kruskal's theorem on a well quasi ordering on trees and also the graph minor theorem.

Every finite set is well quasi ordered by equality. Moreover, the natural numbers are well quasi ordered by $\leq$.

**Lemma 6.3.** *If $A$ is finite, then $(A, =)$ is a wqo. Moreover, $(\mathbb{N}, \leq)$ is a wqo.*

Well quasi orderings are stable under Cartesian products.

**Lemma 6.4.** *Consider two wqos $(A, \leq_A)$ and $(B, \leq_B)$. Then $(A \times B, \leq_{A\times B})$ is a wqo where $(a_1, b_1) \leq_{A\times B} (a_2, b_2)$ if $a_1 \leq_A a_2$ and $b_1 \leq_B b_2$.*

*Proof.* Consider an infinite sequence $(a_i, b_i)_{i\in\mathbb{N}}$ in $A \times B$. As $(a_i)_{i\in\mathbb{N}}$ is an infinite sequence in $A$ and $A$ is a wqo by the assumption, there is (Theorem 6.1) an infinite non-decreasing subsequence $(a_{\varphi(i)})_{i\in\mathbb{N}}$ with $a_{\varphi(i)} \leq_A a_{\varphi(i+1)}$ for all $i \in \mathbb{N}$.

Consider the sequence $(a_{\varphi(i)}, b_{\varphi(i)})_{i\in\mathbb{N}}$. As $(b_{\varphi(i)})_{i\in\mathbb{N}}$ is an infinite sequence in $B$, by the wqo assumption there are $i < j$ with $b_{\varphi(i)} \leq_B b_{\varphi(j)}$. By the definition of subsequences, $i < j$ implies $\varphi(i) < \varphi(j)$. So we found indices $\varphi(i) < \varphi(j)$ with

$$a_{\varphi(i)} \leq_A a_{\varphi(j)} \qquad \text{and} \qquad b_{\varphi(i)} \leq_B b_{\varphi(j)}.$$

We conclude $(a_{\varphi(i)}, b_{\varphi(i)}) \leq_{A\times B} (a_{\varphi(j)}, b_{\varphi(j)})$ as required. ☐

Words can be understood as an unbounded version of Cartesian products. Higman has shown that also words are well quasi ordered (by the subword relation).

**Lemma 6.5 (Higman 1952).** *If $(A, \leq)$ is a wqo, so is $(A^*, \leq^*)$. Here, $u \leq^* v$ with $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$ if there are $1 \leq i_1 < \dots < i_m \leq n$ with $u_j \leq v_{i_j}$ for all $1 \leq j \leq m$.*

*Proof.* To the contrary, assume there are infinite sequences that are bad, i.e., that do not contain comparable elements. We rely on a combinatorial construction to derive the contradiction. It forms an infinite bad sequence $(u_i)_{i\in\mathbb{N}}$ that is particularly small as follows. Select the shortest word $u_0$ that starts a bad sequence. Assume we constructed the sequence $u_0, \dots, u_n$. We then append the shortest word $u_{n+1}$ so that the result $u_0, \dots, u_{n+1}$ still forms a prefix of a bad sequence.

The infinite sequence $(u_i)_{i\in\mathbb{N}}$ is bad. Let $u_i = a_i.v_i$ with $a_i \in A$ and $v_i \in A^*$. By the well quasi ordering assumption on $A$ and Theorem 6.1, sequence $(a_i)_{i\in\mathbb{N}}$ contains an infinite non-decreasing subsequence $(a_{\varphi(i)})_{i\in\mathbb{N}}$. Consider now the sequence

$$u_0, \dots, u_{\varphi(0)-1}, v_{\varphi(0)}, v_{\varphi(1)}, \dots$$

Since $v_{\varphi(0)}$ is strictly shorter than $u_{\varphi(0)}$, the sequence has to be good (otherwise we would have selected $v_{\varphi(0)}$ instead of $u_{\varphi(0)}$). This means, there are two comparable elements. They cannot be among $u_0, \dots, u_{\varphi(0)-1}$, otherwise the sequence $(u_i)_{i\in\mathbb{N}}$ would have been good. Moreover, the ordering cannot be between $u_i$ and $v_{\varphi(j)}$. Otherwise, we had $u_i \leq^* v_{\varphi(j)} \leq^* u_{\varphi(j)}$ and so $u_i \leq^* u_{\varphi(j)}$. Again a contradiction to the assumption that $(u_i)_{i\in\mathbb{N}}$ is bad. Hence, we have $v_{\varphi(i)} \leq^* v_{\varphi(j)}$ with $i < j$. By monotonicity, this means $\varphi(i) < \varphi(j)$. But since also $a_{\varphi(i)} \leq a_{\varphi(j)}$, we derive $u_{\varphi(i)} = a_{\varphi(i)}.v_{\varphi(i)} \leq^* a_{\varphi(j)}.v_{\varphi(j)} = u_{\varphi(j)}$. A contradiction. □

## 6.4 Well Structured Transition Systems

Well structured transition systems are a framework for the automatic verification of infinite state systems. The concept was found independently by Alain Finkel (Cachan) and Parosh Abdulla (Uppsala) when they worked on generalizations of decision procedures that were known for particular models. Finkel strived for an extension of coverability graphs in order to decide termination and boundedness problems. Abdulla was interested in coverability and simulation problems for lossy channel systems.

Technically, wsts are (usually infinite) transition systems where the configurations are equipped with a well quasi ordering. This wqo has to be compatible with the transitions, i.e., larger configurations can imitate the transitions of smaller ones. Imitation is formalized by so-called simulation relations.

**Definition 6.4 (Well structured transition system (wsts), simulation relation).** A *transition systems* is a triple $TS = (\Gamma, \gamma_0, \rightarrow)$ with a (typically infinite) set of *configurations* $\Gamma$, an *initial configuration* $\gamma_0 \in \Gamma$, and a *transition relation* $\rightarrow \subseteq$

$\Gamma \times \Gamma$. The transition system is *well structured* if there is $\leq\, \subseteq \Gamma \times \Gamma$ that is a wqo and a simulation relation. We also write $TS = (\Gamma, \gamma_0, \rightarrow, \leq)$ for a wsts.

Recall that $\leq\, \subseteq \Gamma \times \Gamma$ is a *simulation (relation)* if for all $\gamma_1, \gamma_2, \gamma_3 \in \Gamma$ with $\gamma_1 \rightarrow \gamma_2$ and $\gamma_1 \leq \gamma_3$ there is $\gamma_4 \in \Gamma$ with $\gamma_3 \rightarrow \gamma_4$ and $\gamma_2 \leq \gamma_4$.

With the ordering $\preceq\, \subseteq (Q \times M^{*C}) \times (Q \times M^{*C})$ from Definition 5.3, lossy channel systems are indeed well structured.

**Theorem 6.3 (Lcs are wsts).** *Consider the lcs $L = (Q, q_0, C, M, \rightarrow)$. The transition system $(Q \times M^{*C}, \gamma_0, \rightarrow, \preceq)$ is well structured.*

For the proof, it remains to be shown that $\preceq$ is a wqo and a simulation.

## 6.5 Abdulla's Backwards Search

Our goal is to decide coverability in lossy channel systems. Recall that configuration $(q, W) \in Q \times M^{*C}$ is coverable in $L = (Q, q_0, C, M, \rightarrow)$ if there is a configuration $\gamma \in R(L)$ with $\gamma \succeq (q, W)$. With upward closed sets, the problem can be rephrased as follows. Is the upward closed set $\{(q, W)\}\uparrow$ reachable?

We present an algorithm that solves *reachability of upward closed sets* in a wsts. Formally, the problem takes as input a wsts $TS = (\Gamma, \gamma_0, \rightarrow, \leq)$ and an upward closed set $I \subseteq \Gamma$. The question is whether $I$ is reachable from $\gamma_0$. More precisely, is there an element $\gamma \in I$ with $\gamma_0 \rightarrow^* \gamma$. We first discuss the general decision procedure for wsts and then instantiate it to lossy channel systems.

Before we plunge into the details, we sketch the procedure and mention the key arguments. The idea is to perform the reachability analysis *backwards*. We start with the set $I_0 = I$ of interest. Then we compute the set of configurations $I_1$ that reach $I$ in at most one step. We continue with the configurations $I_2$ that lead to $I$ in up to two steps and so on. The procedure allows us to reformulate reachability as follows. The set $I$ is reachable form $\gamma_0$ if and only if $\gamma_0 \in \bigcup_{j \geq 0} I_j$.

The sets $I_j$ can be shown to be upward closed. Moreover, they form an infinite chain $I_0 \subseteq I_1 \subseteq I_2 \subseteq \ldots$ Therefore, Theorem 6.2 applies and states that the chain stabilizes in some $k \in \mathbb{N}$: $I_k = I_{k+1} = I_{k+2} = \ldots$ With reference to the infinite union above, we get $\bigcup_{j \geq 0} I_j = I_k$. This equation suggests the following procedure to decide upward closed reachability:

- Generate the sequence of upward closed sets $I_0 \subseteq I_1 \subseteq I_2 \subseteq \ldots$
- Check for stabilization, $I_k = I_{k+1}$.
- If the sequence stabilized, check for membership $\gamma_0 \in I_k$.

The problem is that the sets $I_j$ are infinite. This means neither equality $I_k = I_{k+1}$ nor membership $\gamma_0 \in I_k$ can be checked algorithmically without further assumptions on the $I_j$. The solution is to represent these sets symbolically by means of minimal elements $M_j$ and exploit the equation $I_j = M_j\uparrow$. This allows us to store and update only finite sets.

Overview: $I$ is reachable from $\gamma_0$ iff $\gamma_0 \in I_k$ with $I_k = I_{k+1}$ iff $\gamma_0 \geq \gamma$ with $\gamma \in M_k$ and $M_k\!\uparrow = M_{k+1}\!\uparrow$

**Part I: *I* is reachable from $\gamma_0$ iff $\gamma_0 \in I_k$ with $I_k = I_{k+1}$**

Consider a wsts $(\Gamma, \gamma_0, \to, \leq)$ and an upward closed set $I \subseteq \Gamma$ to be checked for reachability. In the construction of the sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq \ldots$ the following observation is important. In wsts, upward closed sets are closed under computing predecessors: if $I$ is upward closed so is $pre(I)$.[1] This follows immediately from the requirement that $\leq$ is a simulation. Interestingly, the fact that upward closure is preserved under predecessors characterizes simulation relations.

**Lemma 6.6 ($pre(I)$ is upward closed).** *Consider a transition system $(\Gamma, \gamma_0, \to)$ and a relation $\leq\, \subseteq \Gamma \times \Gamma$. Then $\leq$ is a simulation if and only if $pre(I)$ is upward closed for every upward closed set $I \subseteq \Gamma$.*

We define the sequence

$$I_0 := I \qquad \text{and} \qquad I_{j+1} := I \cup pre(I_j) \;\text{ for all } j \in \mathbb{N}.$$

Denote by $pre^l(I)$ the set obtained by $l \in \mathbb{N}$ applications of $pre(-)$ to $I$:

$$pre^l(I) := \underbrace{pre(\ldots pre(I))}_{l-\text{times}}. \qquad \text{Then the equality} \qquad I_j = \bigcup_{l=0}^{j} pre^l(I) \tag{6.1}$$

holds and gives rise to the following lemma.

**Lemma 6.7.** *Consider wsts $(\Gamma, \gamma_0, \to, \leq)$, $I \subseteq \Gamma$ upward closed, $\gamma \in \Gamma$, and $n \in \mathbb{N}$. Then $I$ is reachable from $\gamma$ in at most $n$ steps if and only if $\gamma \in I_n$.*

As a consequence, set $I$ is reachable from the initial configuration $\gamma_0$ if and only if $\gamma_0 \in pre^*(I)$ where $pre^*(I) := \bigcup_{j \in \mathbb{N}} I_j$. The union is not really infinite. Equation 6.1 shows the inclusions $I_0 \subseteq I_1 \subseteq I_2 \subseteq \ldots$ With Lemma 6.6, the sets $I_j$ are upward closed for all $j \in \mathbb{N}$. Theorem 6.2 applies and yields a first index $k \in \mathbb{N}$ that satisfies $I_k = I_{k+1}$. By definition of the sets $I_j$, we obtain $I_k = I_{k+1} = I_{k+2} = \ldots$

**Theorem 6.4.** *Consider a wsts $(\Gamma, \gamma, \to, \leq)$ and an upward closed set $I \subseteq \Gamma$. Then $I$ is reachable from $\gamma_0$ if and only if $\gamma_0 \in pre^*(I) = \bigcup_{j \in \mathbb{N}} I_j = I_k$ with $I_k = I_{k+1}$.*

[1] Here, $pre(I) := \{\gamma \in \Gamma \mid \gamma \to \gamma' \in I\}$ is the set of *predecessors of I*. Those configurations that lead to $I$ in a single step.

**Part II:** $\gamma_0 \in I_k$ **with** $I_k = I_{k+1}$ **iff** $\gamma_0 \geq \gamma$ **with** $\gamma \in M_k$ **and** $M_k\uparrow = M_{k+1}\uparrow$

It remains to decide equality $I_k = I_{k+1}$ and membership $\gamma_0 \in I_k$. The trick is to define, in accordance with the $I_j$, a sequence of minimal elements:

$$M_0 := min(I) \qquad \text{and} \qquad M_{j+1} := min(M_0 \cup \bigcup_{\gamma \in M_j} minpre(\gamma)) \;\; \text{for all } j \in \mathbb{N}.$$

The definition relies on a function $minpre(-)$ that returns a set of minimal elements $min(pre(\{\gamma\}\uparrow))$ for the predecessors of $\{\gamma\}\uparrow$. Computability of $minpre(-)$ does not follow from the requirements on wsts but has to be shown for every instantiation of the framework. We say a wsts *has computable minimal predecessors* if the set $minpre(\gamma)$ is computable for every $\gamma \in \Gamma$. The $M_j$ are indeed sets of minimal elements for the $I_j$.

**Lemma 6.8.** $I_j = M_j\uparrow$ *for all* $j \in \mathbb{N}$.

*Proof.* We proceed by induction where the base case $I_0 = min(I)\uparrow = M_0\uparrow$ follows from Lemma 6.2. For the induction step, assume we already have $I_j = M_j\uparrow$ for $j \in \mathbb{N}$. We consider $I_{j+1}$ and derive

$$\begin{aligned}
I_{j+1} &= I \cup pre(I_j)\\
\{\text{ Induction hypothesis }\} &= I \cup pre(\bigcup_{\gamma \in M_j} \{\gamma\}\uparrow)\\
\{\text{ Distributivity of } pre(-) \text{ over } \cup \} &= I \cup \bigcup_{\gamma \in M_j} pre(\{\gamma\}\uparrow)\\
\{\ pre(\{\gamma\}\uparrow) \text{ upward closed }\} &= M_0\uparrow \cup \bigcup_{\gamma \in M_j} min(pre(\{\gamma\}\uparrow))\uparrow\\
\{\text{ Distributivity}\uparrow \text{ over } \cup \} &= \big(M_0 \cup \bigcup_{\gamma \in M_j} min(pre(\{\gamma\}\uparrow))\big)\uparrow\\
\{\text{ Definition minimal elements }\} &= min(M_0 \cup \bigcup_{\gamma \in M_j} min(pre(\{\gamma\}\uparrow)))\uparrow = M_{j+1}\uparrow
\end{aligned}$$

□

Note that the definition of the $M_j$ does not rely on the upward closed sets $I_j$. The relationship is given only by Lemma 6.8. We now have

$$pre^*(I) = \bigcup_{j \in \mathbb{N}} I_j = I_k = M_k\uparrow$$

where $k \in \mathbb{N}$ is the first index with $I_k = I_{k+1}$ or equivalently $M_k\uparrow = M_{k+1}\uparrow$. The latter equality $M_k\uparrow = M_{k+1}\uparrow$ is decidable provided the wqo $\leq$ is decidable: one just compares the minimal elements.

**Theorem 6.5 (Decidability of upward closed reachability, Abdulla 1996).** *Let* $(\Gamma, \gamma_0, \rightarrow, \leq)$ *be a wsts with computable minimal predecessors and decidable* $\leq$.

*Consider the upward closed set $I \subseteq \Gamma$ given by its minimal elements $min(I)$. Then it is decidable whether $I$ is reachable from $\gamma_0$.*

*Proof.* The algorithm computes the sequence of minimal elements as defined above. When it finds $M_k\uparrow = M_{k+1}\uparrow$, it terminates as now $pre^*(I) = M_k\uparrow$. By Theorem 6.4, $\gamma_0$ reaches $I$ iff $\gamma_0 \geq \gamma$ with $\gamma \in M_k$. $\square$

To instantiate the algorithm to LCS, we need a suitable $minpre(-)$ function. Let $L = (Q, q_0, C, M, \rightarrow)$. We take as $minpre(q_2, W_2) := min(T)$, where $T$ is the smallest set so that

$$(q_1, W_1) \in T \quad \text{if } q_1 \xrightarrow{c!m} q_2 \text{ and } W_2 = W_1[c := W_1.m]$$

$$(q_1, W_2) \in T \quad \text{if } q_1 \xrightarrow{c!m} q_2 \text{ and the last element of } W_2(c) \neq m \text{ (or } W_2(c) \text{ is empty)}$$

$$(q_1, W_1) \in T \quad \text{if } q_1 \xrightarrow{c?m} q_2 \text{ and } W_1 = W_2[c := m.W_2(c)]$$

**Lemma 6.9.** *Consider LCS $L = (Q, q_0, C, M, \rightarrow)$ and configuration $\gamma \in Q \times M^{*C}$. Then $minpre(\gamma) = min(pre(\{\gamma\}\uparrow))$.*

One may be skeptical about $(q_1, W_2) \in T$ if $q_1 \xrightarrow{c!m} q_2$ and the last element of $W_2(c)$ is different from $m$. We have $(q_1, W_2) \rightarrow (q_2, W_2[c := W_2(c).m]) \geq (q_2, W_2)$. Hence, $(q_1, W_2) \in pre(\{(q_2, W_2)\}\uparrow)$.

There is no configuration $(q_1, W_2')$ with $W_2' \prec^* W_2$ (in case last $W_2(c) \neq m$). One adds $m$ in order to take the transition. The letter is lost to get $W_2(c)$. Since only a single letter can be added, $W_2(c)$ cannot be constructed from $W_2'(c) \prec^* W_2(c)$.

# Chapter 7
# Simple Regularity and Symbolic Forward Analysis

**Abstract** Symbolic forward analysis of lossy channel systems

## 7.1 Simple Regular Expressions and Languages

Recall that the *regular expressions* over an alphabet $M$ are defined by finite unions, concatenation, and Kleene star of single letters:

$$re ::= \emptyset \mid \varepsilon \mid a \mid re_1 + re_2 \mid re_1.re_2 \mid re^* \qquad \text{where } a \in M.$$

Regular expressions $re$ denote languages $\mathscr{L}(re) \subseteq M^*$ in the standard way:

$$\begin{array}{ll} \mathscr{L}(\emptyset) := \emptyset & \mathscr{L}(re_1 + re_2) := \mathscr{L}(re_1) \cup \mathscr{L}(re_2) \\ \mathscr{L}(\varepsilon) := \{\varepsilon\} & \mathscr{L}(re_1.re_2) := \mathscr{L}(re_1).\mathscr{L}(re_2) \\ \mathscr{L}(a) := \{a\} & \mathscr{L}(re^*) := \mathscr{L}(re)^* := \bigcup_{j \in \mathbb{N}} \mathscr{L}(re)^j \end{array}$$

Here, $\mathscr{L}(re)^j$ denotes $j \in \mathbb{N}$ concatenations of $\mathscr{L}(re)$ where we fix $\mathscr{L}(re)^0 := \varepsilon$. Simple regular expressions are designed to represent languages that are downward closed wrt. Higman's subword ordering $\preceq^*$. Therefore, every occurrence of a letter also offers the choice of loss.

**Definition 7.1 (Simple regular expression).** Consider some underlying alphabet $M$. *Atomic expressions* $e$ allow for choices among letters and form the base case. They are concatenated to *products* $p$. *Simple regular expressions (sres)* $r$ are then choices among products:

$$e ::= (a + \varepsilon) \mid (a_1 + \ldots + a_m)^* \qquad p ::= \varepsilon \mid e.p \qquad r ::= \emptyset \mid p + r$$

where $a, a_1, \ldots, a_m \in M$. A language $\mathscr{L} \subseteq M^*$ is *simple regular* if there is a simple regular expression $r$ with $\mathscr{L} = \mathscr{L}(r)$.

Haines showed that downward closed languages are regular. We first establish this result and then sharpen it as follows. Downward closed languages are precisely the languages represented by sres.

**Theorem 7.1 (Haines '69).** *Let $\mathscr{L} \subseteq M^*$ be any language. Then $\mathscr{L}\downarrow$ is regular.*

*Proof.* Since $\mathscr{L}\downarrow$ is downward closed, the complement $\overline{\mathscr{L}\downarrow}$ is upward closed. Since Higman's ordering $\preceq^*$ is a wqo, this upward closed language can be represented by its (finitely many) minimal elements:

$$\overline{\mathscr{L}\downarrow} \;=\; min(\overline{\mathscr{L}\downarrow})\uparrow = \bigcup_{w \in min(\overline{\mathscr{L}\downarrow})} \{w\}\uparrow . \tag{7.1}$$

Note that the upward closure of a word $w = w_1 \ldots w_n$ is the language

$$\{w\}\uparrow = \;\{y \in M^* \;\mid\; w \preceq^* y\} \;=\; \mathscr{L}(M^*.w_1.M^* \ldots M^*.w_n.M^*)$$

where $M^*$ denotes the choice $\Sigma_{m \in M} m$. This means $\{w\}\uparrow$ is regular. Since $min(\overline{\mathscr{L}\downarrow})$ is finite by Lemma 6.1 and since regular languages are closed under finite unions, we conclude with Equation 7.1 that $\overline{\mathscr{L}\downarrow}$ is regular. Regular languages are also closed under complementation, so $L\downarrow = \overline{\overline{\mathscr{L}\downarrow}}$ is regular. $\square$

The result is indeed surprising. If we define the language of a Turing machine to contain all sequences of transitions that lead to a halting state, we get that $\mathscr{L}(TM)\downarrow$ is regular. This in turn means that the downward closure of languages cannot be computable in general. In the example of Turing machines, we would obtain

$$TM \text{ halts} \quad \text{iff} \quad \mathscr{L}(TM)\downarrow \neq \emptyset \quad \text{iff} \quad \varepsilon \in \mathscr{L}(TM)\downarrow .$$

So there is no algorithm to compute a representation of $\mathscr{L}(TM)\downarrow$ (more precisely, no representation which allows us to evaluate emptiness or membership of $\varepsilon$).

There are interesting classes of languages for which the downward closure is computable. Van Leeuwen has shown in 1978 that the downward closure of context free languages is effectively computable. For Petri nets, the problem remained open until 2010 when it was solved positively by Habermehl, Wimmel, and the author. Establishing such computability results is a beautiful theoretical challenge that finds applications in decidability results for asynchronous hardware. Indeed, consider a shared memory architecture with a writer and a reader. The reader always sees the downward closure of the writers actions. If the reader process is slower than the writer, it may miss intermediary instructions. It was Ahmed Bouajjani who realized this applications of downward closed languages in modelling and verification.

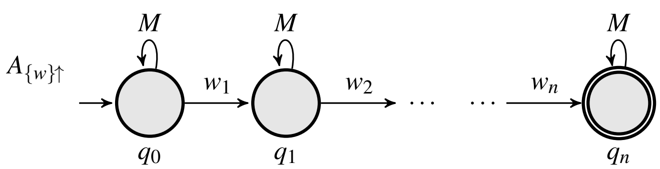
**Theorem 7.2 (Bouajjani '98).** *Language $\mathscr{L}$ is downward closed if and only if it is simple regular.*

*Proof.* For the if direction, recall that simple regularity means $\mathscr{L} = \mathscr{L}(r)$ for some sre $r$. Therefore, an induction along the structure of sres is sufficient that shows $\mathscr{L}(r)$ is downward closed for all sres.

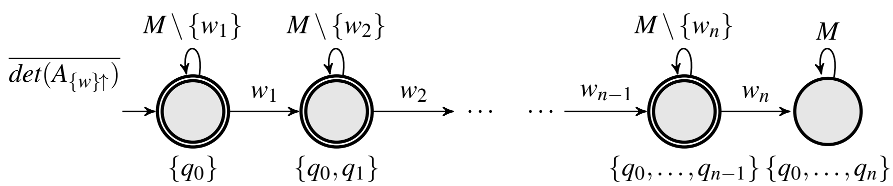
For the only if direction, we apply Haines' theorem and obtain

$$\mathscr{L} \;=\; \overline{\overline{\mathscr{L}}} \;=\; \overline{\bigcup_{w \in min(\overline{\mathscr{L}})} \{w\}\uparrow} \;=\; \bigcap_{w \in min(\overline{\mathscr{L}})} \overline{\{w\}\uparrow}. \tag{7.2}$$

To find an sre for $\overline{\{w\}\uparrow}$, we take a detour and represent $\{w\}\uparrow$ by a finite automaton. This allows us to apply the standard construction for complementation, which hints to the required expression. Let $w = w_1 \ldots w_n$ with $M$ as underlying alphabet. The language $\{w\}\uparrow$ is accepted by




The $M$ labelled loops denote $|M|$ loops, one for each letter in $M$. We determinize the automaton with the powerset construction of Rabin and Scott. Switching then final and non-final states yields an automaton for the complement language:




The automaton operations are known to reflect the operations on languages. Thus, the language of $\overline{det(A_{\{w\}\uparrow})}$ is as desired:

$$\mathscr{L}(\overline{det(A_{\{w\}\uparrow})}) \;=\; \overline{\mathscr{L}(det(A_{\{w\}\uparrow}))} \;=\; \overline{\mathscr{L}(A_{\{w\}\uparrow})} \;=\; \overline{\{w\}\uparrow}.$$

The language is characterized by the sre

$$(M \setminus \{w_1\})^*.(w_1 + \varepsilon).(M \setminus \{w_2\})^*.(w_2 + \varepsilon)\ldots(w_{n-1} + \varepsilon).(M \setminus \{w_n\})^*$$

According to Equation 7.2, we need an sre for the intersection $\bigcap_{w \in min(\overline{\mathscr{L}})} \overline{\{w\}\uparrow}$. On automata, this intersection is reflected by a parallel composition

$$\prod_{w \in min(\overline{\mathscr{L}})} \overline{det(A_{\{w\}\uparrow})}.$$

The result is an, up to loops, acyclic automaton. We decompose it into its maximal paths and reuse the above construction. □

## 7.2 Inclusion among simple regular languages

We intend to use sres to represent sets of configurations in a lossy channel system. More precisely, we develop a concept of symbolic configurations $(q,R)$ where $R$ is a function that assings to each channel $c$ an sre $R(c)$. Based on such symbolic configurations, we again develop a fixed point algorithm of the form

$$I_0 \subseteq I_1 \subseteq I_2 \subseteq \dots \quad \text{until} \quad I_{k+1} = I_k \text{ for some } k \in \mathbb{N}.$$

As the sequence already is increasing, we only need to check $I_{k+1} \subseteq I_k$. This calls for an inclusion check $\mathscr{L}(r_1) \subseteq \mathscr{L}(r_2)$ among simple regular languages. The following result is key to making this inclusion check efficient. It states that we only need to compare products with products.

**Lemma 7.1.** *Consider a product $p$ and an sre $r = p_1 + \dots + p_k$. If $\mathscr{L}(p) \subseteq \mathscr{L}(r)$ then $\mathscr{L}(p) \subseteq \mathscr{L}(p_i)$ for some $1 \leq i \leq k$.*

*Proof.* The proof approach is interesting. We devise a single word $y \in \mathscr{L}(p)$ that is demanding enough so as to ensure inclusion of the full language $\mathscr{L}(p)$. More precisely, the word will guarantee that

$$y \in \mathscr{L}(p_i) \quad \text{implies} \quad \mathscr{L}(p) \subseteq \mathscr{L}(p_i)$$

for every product $p_i$ with $1 \leq i \leq k$. This proves the lemma as

$$y \ \in \ \mathscr{L}(p) \ \subseteq \ \mathscr{L}(r) \ = \ \bigcup_{i=1}^{k} \mathscr{L}(p_i)$$

yields $y \in \mathscr{L}(p_i)$ for some $1 \leq i \leq k$. With the above implication, we conclude $\mathscr{L}(p) \subseteq \mathscr{L}(p_i)$ for this $p_i$. All that remains is to give the construction of $y$.

Let $p = e_1 \dots e_n$ and let $j$ be the maximal number of atomic expressions in the products in $r = p_1 + \dots + p_k$. The goal is to enforce $\mathscr{L}(p) \subseteq \mathscr{L}(p_i)$ if $y \in \mathscr{L}(p_i)$. We set $y = y_1 \dots y_n$ with

$$y_i := a \quad \text{if } e_i = (a+\varepsilon) \qquad\qquad y_i := (a_1 \dots a_m)^{j+1} \quad \text{if } e_i = (a_1 + \dots + a_m)^*.$$

This means we have a word for every atomic expression. For a choice $e_i = (a+\varepsilon)$ we select $y_i = a$ to demand the occurrence of letter $a$. For $e_i = (a_1 + \dots a_m)^*$ we apply the pigeonhole principle. Let the longest product in $r$ be $e'_1 \dots e'_j$. We choose $y_i = (a_1 \dots a_m)^{j+1}$. This means at least two iterations of $a_1 \dots a_m$ have to be in the language of a same expression, $(a_1 \dots a_m)^2 \in \mathscr{L}(e'_l)$ for some $1 \leq l \leq j$. This implies $e'_l = (\dots + a_1 + \dots + a_m)^*$ and guarantees $\mathscr{L}(e_i) \subseteq \mathscr{L}(e'_l)$. Inclusion of the full product $p = e_1 \dots e_n$ iterates the argument for single expressions. $\square$

We now develop a recursive algorithm that checks inclusion among products in linear time. If one of the products is the empty word, we have $\mathscr{L}(\varepsilon) \subseteq \mathscr{L}(p)$ for every product $p$ and $\mathscr{L}(p) \not\subseteq \mathscr{L}(\varepsilon)$ for all $p \neq \varepsilon$. For atomic expressions, we have

$$\begin{aligned}
\mathscr{L}(a+\varepsilon) &\subseteq \mathscr{L}((a_1+\ldots+a_m)^*) \quad \text{if } a \in \{a_1,\ldots,a_m\}\\
\mathscr{L}((a_1+\ldots+a_m)^*) &\subseteq \mathscr{L}((b_1+\ldots+b_n)^*) \quad \text{if } \{a_1,\ldots,a_m\} \subseteq \{b_1,\ldots,b_n\}.
\end{aligned}$$

It remains to set up the recursion for proper products $e_1.p_1$ and $e_2.p_2$. We return $\mathscr{L}(e_1.p_1) \subseteq \mathscr{L}(e_2.p_2)$ if one of the following holds:

$$\begin{aligned}
&\mathscr{L}(e_1) \nsubseteq \mathscr{L}(e_2) \quad \text{and} \quad \mathscr{L}(e_1.p_1) \subseteq \mathscr{L}(p_2)\\
&\mathscr{L}(e_1) = \mathscr{L}(e_2) = \mathscr{L}(a+\varepsilon) \quad \text{and} \quad \mathscr{L}(p_1) \subseteq \mathscr{L}(p_2)\\
&\mathscr{L}(e_1) \subseteq \mathscr{L}(e_2) = \mathscr{L}((a_1+\ldots+a_m)^*) \quad \text{and} \quad \mathscr{L}(p_1) \subseteq \mathscr{L}(e_2.p_2).
\end{aligned}$$

**Lemma 7.2.** *Inclusion among products can be checked in linear time.*

To check inclusion $\mathscr{L}(p_1+\ldots+p_m) \subseteq \mathscr{L}(p'_1+\ldots+p'_n)$ among sres, we compare each product $p_i$ with every product $p'_j$ until we find $\mathscr{L}(p_i) \subseteq \mathscr{L}(p'_j)$. This local check among products is sufficient according to Lemma 7.1.

**Lemma 7.3.** *Inclusion among sres can be checked in quadratic time.*

## 7.3 Computing the Effect of Transitions

The result of applying an operation like $c!a$ to an sre $r$ should again be an sre. We show how to compute this sre. In the next section, we obtain a similar computability result for the application of iterated sequences of operations.

We fix the channel $c$ to which we apply the operations and write $!a$ and $?a$ instead of $c!a$ and $c?a$. Let $M$ be the alphabet of messages that are sent and received. We define the effect of performing a send operation $!a$ on $\mathscr{L} \subseteq M^*$ to be the language $\mathscr{L}\oplus!a := \{y \in M^* \mid y = x.a \text{ for some } x \in \mathscr{L}\}$. Similarly, the effect of receiving from $\mathscr{L}$ is defined by $\mathscr{L}\oplus?a := \{y \in M^* \mid x = a.y \text{ for some } x \in \mathscr{L}\}$. The languages $\mathscr{L}$ we are concerned with are represented by sres $r$. The following lemma shows how to compute an sre that represents $\mathscr{L}(r) \oplus op$.

**Lemma 7.4.** *Consider an sre $r$ and an operation $op \in \{!a, ?a\}$. There is an sre $r \oplus op$ with $\mathscr{L}(r \oplus op) = \mathscr{L}(r) \oplus op$. Moreover, $r \oplus op$ can be computed in linear time.*

*Proof.* We first consider products. For send operations, we set $p\oplus!a := p.(a+\varepsilon)$. For receive operations, the base case is $\varepsilon\oplus?a := \emptyset$. In the induction step, we have

$$(e.p)\oplus?a := \begin{cases} e.p & \text{if } e = (a_1+\ldots+a_m)^* \text{ and } a \in \{a_1,\ldots,a_m\}\\ p & \text{if } e = (a+\varepsilon)\\ p\oplus?a & \text{otherwise.}\end{cases}$$

This means the operation is applied to the remaining product $p$ provided letter $a$ cannot be served by the first atomic expression $e$.

For an sre $r = p_1 + \ldots + p_k$ we set $r \oplus op := (p_1 \oplus op) + \ldots + (p_k \oplus op)$ to apply the operation to all products. It is readily checked that language equality holds and that $r \oplus op$ can be computed in time linear in the size of $r$. □

## 7.4 Computing the Effect of Loops

Our goal is to accelerate the coverability analysis in lossy channel systems. The term *acceleration* means we determine the effect of arbitrary iterations of control loops in a single computation, rather than calculating the effect of every transition.

Technically, a *control loop* is a sequence of transitions that starts and ends in a same state:

$$q_0 \xrightarrow{op_1} q_1 \xrightarrow{op_2} \ldots \xrightarrow{op_n} q_n \quad \text{with} \quad q_0 = q_n.$$

We assume that all operations in $ops = op_1 \ldots op_n$ act on the same channel $c$.[1] The main contribution is an algorithm which, given an sre $r$ and sequence $ops$, computes a new sre $r \oplus ops^*$. The latter reflects the effect of arbitrary iterations of $ops$ on $r$.

The key insight is that the effect of loops stabilizes. For every sre $r$ and sequence $ops$, there is an $n \in \mathbb{N}$ that satisfies the following. The language obtained by at least $n$ iterations of $ops$ on $r$ is characterized by an sre $r \oplus ops^{\geq n}$. As a consequence, the effect of arbitrary iterations of $ops$ on $r$ can be captured by the sre

$$r \oplus ops^* := r + (r \oplus ops) + \ldots + (r \oplus ops^{n-1}) + (r \oplus ops^{\geq n}).$$

**Theorem 7.3 (Bouajjani 1998).** *Consider product $p$ and a sequence of operations $ops$. There is an $n \in \mathbb{N}$ and a product $p \oplus ops^{\geq n}$ so that either $\mathscr{L}(p \oplus ops^n) = \emptyset$ or $\mathscr{L}(p \oplus ops^{\geq n}) = \bigcup_{j \geq n} \mathscr{L}(p \oplus ops^j)$. The value of $n$ is linear in the size of $p$ and $p \oplus ops^{\geq n}$ can be computed in quadratic time.*

Before we turn to the proof, we define notions that help us shorten the presentation. Consider a sequence $ops$ of operations $!a$ or $?a$ where $a$ is in the alphabet $M$. We denote by $ops?$ the subword of receive operations. Similarly, $ops!$ yields the subword of send operations. We assume the symbol of operation to be removed. So for $ops = !a.?b.?c.!d$ we have $ops? = b.c$ and $ops! = a.d$. We use $M(ops?)$ and $M(ops!)$ to restrict alphabet $M$ to the letters in $ops?$ and $ops!$, respectively.

We extend Higman's ordering into two directions. We use the *growing subword ordering* $x \preceq^*_{grow} y$ with $x, y \in M^*$ to indicate that there is $m \in \mathbb{N}$ so that $x^{m+1} \preceq^* y^m$. Not only does $x \preceq^*_{grow} y$ imply $x \preceq^* y$. It indeed shows that $y$ is large enough so as to accommodate several instances of $x$. If both words are iterated $m+1$ times, then $x^{m+1}$ fits into only $m$ iterations of $y$. The $(m+1)$st iteration of $y$ is left untouched by the subword ordering. Ordering $\preceq^*_{grow}$ can be checked in quadratic time.

If we iterate a control loop, $ops$ can be understood as a cycle. Indeed, when the last operation $op_n$ in the control loop is reached, the word will continue with

[1] If this is not the case, we first decompose $ops$ into the subwords $ops_c$ for the single channels.

the first operation $op_1$. The *cyclic subword ordering* $x \preceq^*_{cyc} y$ requires that there is a decomposition $x = x_1.x_2$ so that $x_2.x_1 \preceq^* y$. Intuitively, the ordering rotates the cyclic word by $x_1$. Note that $x \preceq^*_{cyc} y$ implies $x \preceq^* y^2$. Moreover, $\preceq^*_{cyc}$ can be checked in quadratic time as well.

*Proof.* We distinguish four cases. In the first two, the control loop can be iterated an unbounded number of times so that the channel content grows unboundedly. In the third case, the loop can be iterated an unbounded number of times but the channel content stabilizes. Finally, a deadlock may occur because a receive fails.

**Case (1)** $\mathscr{L}((ops?)^*) \subseteq \mathscr{L}(p)$  If $ops? = \varepsilon$ we set $n := 0$ and $p \oplus ops^{\geq n} := p.M(ops!)^*$. If $ops? \neq \varepsilon$, there is a first atomic expression $e = (a_1 + \ldots + a_m)^*$ in $p = p_1.e.p_2$ that satisfies $M(ops?) \subseteq \{a_1, \ldots, a_m\}$. We set $n := |p_1|$ and $p \oplus ops^{\geq n} := e.p_2.M(ops!)^*$.

**Case (2)** $\mathscr{L}((ops?)^*) \not\subseteq \mathscr{L}(p)$ and $ops? \preceq^*_{grow} ops!$ and $p \oplus ops \neq \emptyset$  We set $n := |p|$ and $p \oplus ops^{\geq n} := M(ops!)^*$.

**Case (3)** $\mathscr{L}((ops?)^*) \not\subseteq \mathscr{L}(p)$ and $ops? \not\preceq^*_{grow} ops!$ and $ops? \preceq^*_{cyc} ops!$ and $p \oplus ops^2 \neq \emptyset$  We set $n := |p| + 1$ and $p \oplus ops^{\geq n} := p \oplus ops^n$.

**Case (4)** where (1)-(3) do not apply  We set $n := |p| + 1$ and have $p \oplus ops^n = \emptyset$. □

In Case (1), there is an atomic expression $e = (a_1 + \ldots + a_m)^*$ in product $p = p_1.e.p_2$. It serves all receive operations in *ops* once $p_1$ has been consumed. This means *ops* can be iterated an arbitrary number of times. Sequence *ops*? is always received from $e$ and *ops*! is appended after $p_2$. Let $ops! = b_1 \ldots b_n$. Due to lossiness, the downward closure of $\mathscr{L}((ops!)^*)$ is

$$\mathscr{L}((ops!)^*)\!\downarrow = (b_1 + \ldots + b_n)^* = M(ops!)^*.$$

In the second case we do not have $\mathscr{L}((ops?)^*) \subseteq \mathscr{L}(p)$. Therefore, the original channel content will be consumed after at most $n = |p|$ iterations of the control loop. But as $ops? \preceq^*_{grow} ops!$ we have $(ops?)^{m+1} \preceq^* (ops!)^m$ for some $m \in \mathbb{N}$. This means the channel content grows by *ops*! every $m+1$ iterations of the loop. So we can have any number of *ops*! sequences at the end of the channel. The downward closure is again $M(ops!)^*$. Condition $p \oplus ops \neq \emptyset$ ensures the first iteration of the control loop is executable. The remaining iterations can be performed since $ops? \preceq^*_{grow} ops!$ implies $ops? \preceq^* ops!$.

In the third case, the channel content is again lost after $|p|$ iterations of the control loop. Afterwards, the send operations in *ops*! serve the receive operations in *ops*? in a way that forbids the channel content to grow. We require two iterations of *ops* to be feasible on $p$ to guarantee executability of arbitrary iterations. The reason is that $x \preceq^*_{cyc} y$ implies $X \preceq^* y^2$ but does not imply $x \preceq^* y$. A counterexample to why $p \oplus ops \neq \emptyset$ is not sufficient for feasibility of arbitrary iterations is the following:

$$p = (b + \varepsilon).(a + \varepsilon) \quad ops = ?b.?a.!a.!b.$$

Whe have $p \oplus ops = (a + \varepsilon).(b + \varepsilon)$ and $p \oplus ops^2 = \emptyset$.

If cases (1) to (3) fail, the loop can be iterated at most $|p|$ times. Then the channel is empty and the next iteration enters a deadlock as a receive fails.

## 7.5 A Symbolic Forward Algorithm for Coverability

Consider an lcs $L = (Q, q_0, C, M, \rightarrow)$ and a set of configurations $\Gamma_F \subseteq Q \times M^{*C}$. Harnessing the algorithms from Section 7.2 to 7.4, we now design a procedure that checks reachability of $\Gamma_F$ from the initial configuration of $L$. Different from the backwards search from Section 6.5, the new algorithm no longer stores minimal elements to describe upward closed sets. The idea is to store *symbolic configurations* $(q, R)$ where function $R$ assigns an sre $R(c)$ to every channel $c \in C$. The symbolic configuration denotes the set of (standard) configurations

$$\mathscr{L}((q,R)) := \{(q,W) \in Q \times M^{*C} \mid W(c) \in \mathscr{L}(R(c)) \text{ for all } c \in C\}.$$

The overall verification algorithm is given in Figure 7.1. It maintains a set $V$ of symbolic configurations computed so far. When we calculate the effect of transitions and control loops, we find new symbolic configurations $\gamma$. We add $\gamma$ to $V$ provided it denotes new standard configurations that are not represented by $V$ so far. When a configuration in $\Gamma_F$ is found, the algorithm returns *reachable*. When no more symbolic configurations are found ($V_0 \subseteq V_1 \subseteq \ldots \subseteq V_k = V_{k+1}$), the procedure returns *unreachable*. The algorithm expects a finite set *loops* of control loops to be accelerated. A canonical choice for *loops* are the *simple* control loops that do not repeat states.

The algorithm requires two comments. First, note that a symbolic configuration $\gamma$ may be added to $V$ and $L$ although it does not represent new configurations. This happens if

$$\mathscr{L}(\gamma) \subseteq \bigcup_{\gamma' \in V} \mathscr{L}(\gamma') \quad \text{but there is no single } \gamma' \in V \text{ so that } \mathscr{L}(\gamma) \subseteq \mathscr{L}(\gamma').$$

We stick with the local comparison as it can be checked in polynomial time.

A second comment is that the algorithm is sound but incomplete. If it returns *reachable* or *unreachable* the answer is correct. But the procedure may run forever. More precisely, with a breadth-first processing of configurations the algorithm is a semidecider for reachable instances: if $\Gamma_F$ is reachable, it will find a configuration in $\Gamma_F$ and terminate. (Indeed the algorithm only finds reachable configurations.) However, it may fail to terminate when $\Gamma_F$ is unreachable. We discuss the underlying computability theoretic reasons in the following chapter.

**input** : $L = (Q, q_0, C, M, \rightarrow)$, $\Gamma_F \subseteq Q \times M^{*C}$,
*loops* a finite set of control loops to be accelerated

**begin**
  $V := \{\gamma_0\}$
  $L := \{\gamma_0\}$
  **while** $L \neq \emptyset$ **do**
    **let** $\gamma_1 = (q_1, R_1) \in L$
    $L := L \setminus \{\gamma_1\}$
    **for all** transitions $q_1 \xrightarrow{c,op} q_2$ **do**
      $\gamma := (q_2, R_1[c := R_1(c) \oplus op])$
      **if** $\mathscr{L}(\gamma) \not\subseteq \mathscr{L}(\gamma')$ for all $\gamma' \in V$ **then**
        $V := V \cup \{\gamma\}$
        $L := L \cup \{\gamma\}$
      **end if**
    **end for all**
    **for all** control loops $q_1 \xrightarrow{ops} q_1$ with $ops \in loops$ **do**
      $\gamma := (q_1, R)$ where $R(c) := R_1(c) \oplus ops_c^*$ for all $c \in C$
      **if** $\mathscr{L}(\gamma) \not\subseteq \mathscr{L}(\gamma')$ for all $\gamma' \in V$ **then**
        $V := V \cup \{\gamma\}$
        $L := L \cup \{\gamma\}$
      **end if**
    **end for all**
    **if** $\mathscr{L}(V) \cap \mathscr{L}(\Gamma_F) \neq \emptyset$ **then**
      **return** reachable
    **end if**
  **end while**
**return** unreachable

**Fig. 7.1** Symbolic forward algorithm for coverability in LCS.

# Chapter 8
# Undecidability Results for Lossy Channel Systems

**Abstract** Undecidability and non-computability results for lossy channel systems.

We establish a fundamental undecidability result for lossy channel systems. As main consequence, we derive incompleteness of the previous acceleration algorithm. The problem we consider asks for whether a given control state in a lossy channel system can be visited infinitely often. Consider an LCS $L = (Q, q_0, C, M, \rightarrow)$ and a state $q \in Q$. More formally, the *recurrent state problem (RSP)* asks for whether there is an infinite sequence of configurations $\gamma_0 \rightarrow \gamma_1 \rightarrow \dots$ with $\gamma_i = (q_i, W_i)$ so that $q_i = q$ for infinitely many $i \in \mathbb{N}$.

We show that RSP is undecidable. This is interesting for several reasons. First, the infinite repetition of a designated state corresponds to the acceptance condition in *Büchi automata*.[1] Like finite automata serve as monitors for safety properties, Büchi automata act as observers for liveness properties: is a desirable situation guaranteed to happen? Undecidability of RSP rules out decidability of liveness properties for LCS. As a consequence, liveness properties are undecidable for general WSTS. Surprisingly, liveness properties can be shown to be decidable for Petri nets. It is an open research problem to find a natural subclass of WSTS that has a decidable liveness problem. A good restriction to general WSTS should extend and at best explain the positive result for Petri nets, and illustrate the strength of LCS.

As second consequence of this undecidability result for RSP, we prove that the channel content is not computable for LCS. This shows Bouajjani's acceleration approach has to be incomplete.

We obtain undecidability by a reduction from the *cyclic Post's correspondence problem (CPCP)*. It takes as input a finite alphabet $M$ and a finite list of pairs

[1] Syntactically, Büchi automata are finite state automata. Their semantics, however, is defined in terms of infinite words.

$(x_1,y_1),\dots,(x_n,y_n)$ with $x_i,y_i \in M^*$. The question is whether there is a finite and non-empty sequence of indices $i_1,\dots,i_m \in \{1,\dots,n\}$ so that

$$x_{i1}\dots x_{im} =_{cyc} y_{i1}\dots y_{im}.$$

Here, $x =_{cyc} y$ if there are $x',x'' \in M^*$ so that $x = x'.x''$ and $y = x''.x'$. Intuitively, $x$ and $y$ are equal when considered as circles.

**Theorem 8.1 (Ruohonen 1983).** *CPCP is undecidable.*

We reduce CPCP to RSP in order to establish

**Theorem 8.2 (Abdulla, Jonsson).** *RSP is undecidable.*

*Proof.* Consider an instance of CPCP with alphabet $M$ and list $(x_1,y_1),\dots,(x_n,y_n)$. We construct an LCS $L = (Q,q_0,C,M,\rightarrow)$ with a designated state $q \in Q$ so that the following equivalence holds: CPCP has a solution if and only if $L$ has a transition sequence $\gamma_0 \rightarrow \gamma_1 \rightarrow \dots$ that visits $q$ infinitely often. The construction, illustrated in Figure 8.1, is as follows.

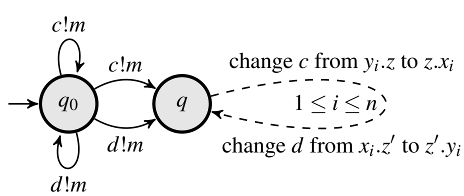



**Fig. 8.1** Sketch of the lossy channel system in the encoding of CPCP. There are loops labelled by $c!m$ and $d!m$ for every $m \in M$, and similar for the transitions from $q_0$ to $q$.

The LCS takes as messages the alphabet $M$ of the CPCP instance. It has two channels $\{c,d\} =: C$. In the initial state $q_0$, the LCS guesses channel contents for $c$ and $d$ via loops labelled by $c!m$ and $d!m$ for every $m \in M$. The channel contents are supposed to solve the CPCP instance. With a transition from $q_0$ to the state $q$ of interest, the LCS stops guessing and starts validating the proposed solution. To this end, it has a cycle for every pair $(x_i,y_i)$ with $1 \leq i \leq n$. Cycle $i$ changes the content of channel $c$ from $y_i.z$ to $z.x_i$ for every $z \in M^*$. A similar change is performed on $d$. It is immediate to implement the changes via sequences of receive and send operations. We now argue that the CPCP instance has a solution iff $L$ admits a transition sequence that visits $q$ infinitely often.

$\Rightarrow$ Assume that $i_1,\dots,i_m$ solves the CPCP instance. The words are

$$x := x_{i1}\dots x_{im} \quad y := y_{i1}\dots y_{im} \quad \text{so that} \quad x =_{cyc} y.$$

By definition of $=_{cyc}$, we have $x = x'.x''$ so that $y = x''.x'$ for some $x',x'' \in M^*$. We construct a transition sequence that visits $q$ infinitely often. In state $q_0$, we send $y.x''$

to channel $c$ and $x.x'$ to channel $d$. With this channel content, we move to $q$. The $i1$st cycle transforms

$$\begin{array}{rll} y.x'' = y_{i1}.y_{i2}\ldots y_{im}.x'' & \text{into} \quad y_{i2}\ldots y_{im}.x''.x_{i1} & \text{for channel c} \\ x.x' = x_{i1}.x_{i2}\ldots x_{im}.x' & \text{into} \quad x_{i2}\ldots x_{im}.x'.y_{i1} & \text{for channel d.} \end{array}$$

Then we continue with the $i2$nd cycle in the expected way. Eventually, channel $c$ contains $x''.x$ and $d$ holds $x'.y$. We now observe that

$$x''.x = x''.x'.x'' = y.x''$$
$$x'.y = x'.x''.x' = x.x'.$$

This means $m$-iterations of the cycles recreate the initial channel contents $y.x''$ for $c$ and $x.x'$ for $d$. To visit $q$ infinitely often, we repeat the $m$-iterations infinitely often. Note that the transition sequence we chose does not loose messages.

$\Leftarrow$ For the reverse direction, one can show that if CPCP has no solution, then every transition sequence that leads to $q$ eventually deadlocks. ☐

The above proof relies on two channels, and one may ask whether LCS with a single channel have a decidable RSP. The answer is negative and sheds some light on the expressiveness of LCS.

**Lemma 8.1.** *RSP is undecidable even for LCS with one channel.*

*Idea.* Consider the above LCS $L$ with two channels $c$ and $d$. We construct a new LCS $L'$ with a single channel $s$. The new alphabet is $C \times M$. This means the new messages $(c,m)$ and $(d,m)$ keep track of the channel $c$ or $d$ that message $m$ stems from. The configurations $\gamma = (q,(w_c,w_d))$ of $L$ are imitated in the new LCS by configurations $\gamma' = (q,w)$. So the state $q$ coincides, but the content of $\gamma'$ is a shuffle $w \in ((\{c\}\times w_c) \sqcup\!\sqcup (\{d\}\times w_d))$ of the contents in both channels.[2] A send action $c!m$ of $L$ yields $s!(c,m)$ in $L'$. Imitating a receive $c?m$ of $L$ is more delicate. The problem is that, due to shuffling, $L'$ may not have $(c,m)$ at the head of channel $s$. The rotation construction from the exercises solves this problem. ☐

Theorem 8.2 yields our main result. The channel content in LCS is not computable. Therefore, the acceleration procedure from Chapter 7 has to be incomplete.

**Theorem 8.3.** *For an LCS $L = (Q,q_0,C,M,\rightarrow)$ with state $q \in Q$ and channel $c \in C$ there is no algorithm to compute an SRE that represents*

$$W(q,c) := \{w \in M^* \mid \gamma_0 \rightarrow^* (q,W) \text{ with } W(c) = w\}.$$

Note, however, that $W(q,c)$ is simply regular by Theorem 7.2.

[2] The *shuffle operator* is well known in formal language theory. Consider $M$ as underlying alphabet. The operator is defined inductively by $w \sqcup\!\sqcup \varepsilon := w =: \varepsilon \sqcup\!\sqcup w$ for all $w \in M^*$ and $a_1.w_1 \sqcup\!\sqcup a_2.w_2 := a_1.(w_1 \sqcup\!\sqcup a_2.w_2) \cup a_2.(a_1.w_1 \sqcup\!\sqcup w_2)$ for all $a_1,a_2 \in M$, $w_1,w_2 \in M^*$.

*Proof.* The result follows from a reduction of RSP. Consider as instance the LCS $L = (Q, q_0, C, M, \rightarrow)$ and state $q \in Q$. We construct a modified LCS $L'$ with a new channel $n \in C$ so that the content of $n$ reflects the repetition of $q$ as follows. There is a transition sequence $\gamma_0 \rightarrow \gamma_1 \rightarrow \ldots$ that visits $q$ infinitely often iff $W(q,c)$ is infinite. Essentially, $L'$ keeps track of when $q$ is entered by adding a message to the new channel $n$. More precisely, $L'$ adds to $L$ a new state $q'$. Every transition that leads to $q$ is redirected to $q'$. From $q'$, a single transition labelled $n!x$ leads to $q$. Here, $x \in M$ is an arbitrary but fixed message. The remaining transitions of $L$ are left unchanged in $L'$. In particular there are no transitions that consume messages from the new channel $n$. Figure 8.2 illustrates the construction.

We show that there is a run visiting $q$ infinitely often if and only if $W(q,n)$ is infinite. The direction from left to right is immediate. For the reverse, one forms a tree of all transition sequences that end in a configuration $(q,W)$. Since language $W(q,n)$ is infinite, there are infinitely many transition sequences leading to $q$. Thus, the tree is infinite. Moreover, the tree is finitely branching. König's lemma applies and yields an infinite path $\gamma_0 \rightarrow \gamma_1 \rightarrow \ldots$ in the tree. State $q$ is visited infinitely often on this path. To see this, assume there was a last configuration with state $q$. Then, by construction, the path would end in this configuration. A contradiction.

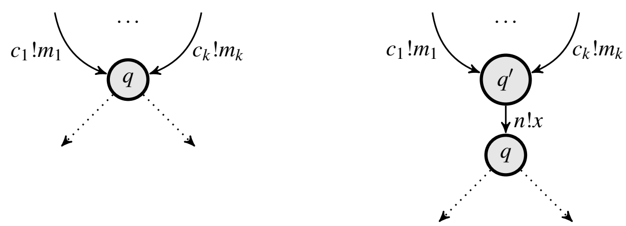



**Fig. 8.2** Reduction from RSP to the computation of channel contents. The original transitions in $L$ are given to the left, the modification in $L'$ is depicted to the right.

We now derive the desired non-computability result for channel contents. If an SRE was computable for $W(q,n)$, then we could also decide finiteness of $W(q,n)$ using this SRE. With the previous reduction, this decides RSP. Hence, such an SRE is not computable. $\square$

The proof shows more. No representation of $W(q,n)$ is computable that allows us to decide finiteness of the language.

# Chapter 9
# Expand, Enlarge, and Check

**Abstract** EEC

We are still looking for a forward algorithm that solves upward closed reachability in WSTS in a complete way. The strong motivation for forward algorithms is in their efficiency. Backwards algorithms often encounter search trees with high outdegree. Forward algorithms are more deterministic. Moreover, verification techniques like partial order reduction immediately apply to forward algorithms while their design is difficult for backwards searches.

To circumvent the non-computability result from Theorem 8.3, we refrain from computing the precise set of reachable configurations in a WSTS. We rather employ two sequences of *approximations*

$$Under(TS,\Gamma_0), Under(TS,\Gamma_1),\ldots \quad \text{and} \quad Over(TS,\Gamma_0,L_0), Over(TS,\Gamma_1,L_1),\ldots$$

Sequence $Under(TS,\Gamma_0), Under(TS,\Gamma_1),\ldots$ provides more and more precise *underapproximations* of the WSTS *TS*. They are used to decide the *positive instances* of upward closed reachability: if the upward closed set is reachable from the initial configuration, some underapproximation $Under(TS,\Gamma_i)$ will report this. The second sequence gives more and more precise *overapproximations* of the WSTS. They will decide the *negative instances*. Since we have two semi-decision procedures, the combination of both algorithms decides upward closed reachability.

The construction enjoys a beautiful analogy. Abdulla's algorithm is a *backwards* search that manipulates *upward closed sets* represented by *minimal elements*. EEC in turn is a *forward* algorithm that manipulates *downward closed sets*. The following section shows how to represent downward closed sets by *limit elements*.

## 9.1 Domains of Limits

Consider the WQO $(C, \leq)$. Upward closed sets $I \subseteq C$ are finitely represented by their minimal elements: $min(I)\uparrow = I$. The representation is effective. Membership in and inclusion among upward closed sets can be checked via $\leq$. What is a finite and effective representation of downward closed sets? We propose to use limit elements $l \notin C$. The idea is to reflect infinite non-decreasing sequences

$$c_0 \leq c_1 \leq c_2 \leq \ldots$$

For example, the sequence $(0,0),(0,1),(0,2),\ldots$ in $\mathbb{N}^2$ is represented by $(0,\omega)$. Similarly, the sequence of languages

$$\mathscr{L}(a+\varepsilon), \mathscr{L}((a+\varepsilon).(b+\varepsilon)), \mathscr{L}((a+\varepsilon).(b+\varepsilon).(a+\varepsilon)), \ldots$$

yields as limit the language $\mathscr{L}((a+b)^*)$. To be useful in a decision procedure for upward closed reachability, limit elements should satisfy some constraints.

**Definition 9.1 (Adequate domain of limits).** Let $(C, \leq)$ be a WQO. A pair $(L, r)$ consisting of a set of *limit elements* $L$ with $L \cap C = \emptyset$ and a *representation function* $r : L \cup C \to \mathbb{P}(C)$ is called an *adequate domain of limits (ADL) for* $(C, \leq)$ provided the following conditions hold.

**(L1)** For $l \in L$, $r(l)$ is downward closed. Moreoever, $r(c) := \{c\}\downarrow$ f.a. $c \in C$.
**(L2)** There is a *top element* $\top \in L$ with $r(\top) = C$.
**(L3)** For any downward closed set $D \subseteq C$ there is a *finite* set $D' \subseteq C \cup L$ with $r(D') = D$. Condition **(L3)** is also called *completeness*.

The domain of limits has to be compatible with the transition relation in the WSTS.

**Definition 9.2 (Effectiveness).** A WSTS $(\Gamma, \gamma, \to, \leq)$ and an ADL $(L, r)$ for $(\Gamma, \leq)$ are called *effective* if

**(E1)** For all $d \in \Gamma \cup L$, finite $D \subseteq \Gamma \cup L$, inclusion $suc(r(d)) \subseteq r(D)$ is decidable.
**(E2)** For all finite $D_1, D_2 \subseteq \Gamma \cup L$, inclusion $r(D_1) \subseteq r(D_2)$ is decidable.

We observed that Petri nets are WSTS. The limit elements are extended markings in $\mathbb{N}_\omega^{|S|}$. That this domain is adequate and effective is not hard to check. For LCS, the symbolic configurations from Chapter 7 form an ADL that can be shown to be effective. Recall that symbolic configurations assign an SRE to every channel.

Consider WSTS $TS = (\Gamma, \gamma_0, \to, \leq)$ and an upward closed set $I \subseteq \Gamma$. To solve upward closed reachability means to decide $R(TS) \cap I = \emptyset$. The following lemma shows that downward closed sets are sufficient for this task.

**Lemma 9.1.** *We have $R(TS) \cap I = \emptyset$ if and only if $R(TS)\downarrow \cap I = \emptyset$.*

Note that Definition 9.1 yields a finite representation for $R(TS)\downarrow$. By **(L3)**, there is a finite set $CS(TS) \subseteq \Gamma \cup L$ so that $r(CS(TS)) = R(TS)\downarrow$. We call $R(TS)\downarrow$ the *covering set* of $R(TS)$. The finite set $CS(TS)$ is the *coverability set* of $R(TS)$.

Combined with Lemma 9.1 this finiteness brings us closer to a decision procedure for coverability. Indeed, the term coverability set is chosen intentionally: the EEC algorithm can be understood as an advanced version of coverability graphs. But how to circumvent the non-computability of coverability sets for LCS? The trick is to approximate $CS(TS)$ rather than to compute it precisely.

## 9.2 Underapproximation

We construct an underapproximation of a transition system $TS = (\Gamma, \gamma_0, \rightarrow)$ wrt. a *finite* subset of configurations $\Gamma' \subseteq \Gamma$. The idea is to reflect the transition sequences that visit configurations in $\Gamma'$, only.

**Definition 9.3 (Underapproximation wrt. $\Gamma'$).** Let $TS = (\Gamma, \gamma_0, \rightarrow)$ and consider a finite set $\Gamma' \subseteq \Gamma$ with $\gamma_0 \in \Gamma'$. The *underapproximation of TS wrt.* $\Gamma'$ is the transition system $Under(TS, \Gamma') := (\Gamma', \gamma_0, \rightarrow \cap (\Gamma' \times \Gamma'))$.

With $\Gamma'$ large enough, this underapproximation decides the positive instances of upward closed reachability. To begin with, we argue that the underapproximation reports correctly on reachability. If it finds an upward closed set $I \subseteq \Gamma$ reachable, then the set is reachable in the original transition system.

**Lemma 9.2 (Soundness).** *If $R(Under(TS, \Gamma')) \cap I \neq \emptyset$ then $R(TS) \cap I \neq \emptyset$.*

Moreover, if set $I$ is reachable in $TS$ then some underapproximation will detect this.

**Lemma 9.3 (Completeness).** *If $R(TS) \cap I \neq \emptyset$ then there is a finite set $\Gamma' \subseteq \Gamma$ with $\gamma_0 \in \Gamma'$ so that $R(Under(TS, \Gamma')) \cap I \neq \emptyset$.*

## 9.3 Overapproximation

For the following development, we assume that the WSTS $TS = (\Gamma, \gamma_0, \rightarrow, \leq)$ to be approximated is *deadlock free*: for all $\gamma_1 \in \Gamma$ there is $\gamma_2 \in \Gamma$ so that $\gamma_1 \rightarrow \gamma_2$. In the case of LCS, deadlock freeness can always be achieved by adding a loop to each state that sends to a fresh channel.

The underapproximation of $TS$ is parameterized by a finite set of configurations $\Gamma' \subseteq \Gamma$. The overapproximation $Over(TS, \Gamma', L')$ additionally relies on a *finite set of limit elements* $L'$ from an ADL $(L, r)$. Intuitively, transition sequences that stay within $\Gamma'$ are represented precisely by $Over(TS, \Gamma', L')$. When we encounter a configuration outside $\Gamma'$, we overapproximate it using limits from $L'$.

The problem is in the choice of limits. There may be two sets $E_1, E_2 \subseteq \Gamma' \uplus L'$ that overapproximate $suc(r(d))$ with $d \in \Gamma' \uplus L'$. This means $suc(r(d)) \subseteq r(E_1)$ and $suc(r(d)) \subseteq r(E_2)$. If the sets are incomparable, $r(E_1) \not\subseteq r(E_2)$ and $r(E_2) \not\subseteq r(E_1)$, both overapproximation are reasonable. The trick is to avoid a choice but consider all overapproximations. As a result, $Over(TS, \Gamma', L')$ will be an and-or graph rather than a transition system.

### *9.3.1 And-Or Graphs*

And-or graphs are bipartite graphs with an initial or-vertex.

**Definition 9.4 (And-or graph).** An *and-or graph* is a graph $G = (V_A \uplus V_O, v_O, \rightarrow)$ with disjoint sets of *and vertices* $V_A$, *or vertices* $V_0$ with *initial vertex* $v_O \in V_O$, and edges $\rightarrow \subseteq (V_A \times V_O) \cup (V_O \times V_A)$. We assume that for every $v_1 \in V_A \uplus V_O$ there is $v_2 \in V_O \uplus V_A$ with $v_1 \rightarrow v_2$.

For and-or graphs, the analogue of a transition sequence is an execution tree.

**Definition 9.5 (Execution tree).** Consider $G = (V_A \uplus V_O, v_O, \rightarrow)$. An *execution tree of* $G$ is an infinite tree $T = (N, n_r, \rightsquigarrow, \lambda)$ with node labelling $\lambda : N \rightarrow V_A \uplus V_O$ that satisfies the following compatibility requirements:

(i) $\lambda(n_r) = v_O$

(ii) For all $n_1 \in N$ with $\lambda(n_1) \in V_O$ there is precisely one $n_2 \in N$ with $n_1 \rightsquigarrow n_2$. Moreover, the nodes satisfy $\lambda(n_1) \rightarrow \lambda(n_2)$.

(iii) For all $n_1 \in N$ with $\lambda(n_1) \in V_A$ we have that

(a) for all $v_2 \in V_O$ with $\lambda(n_1) \rightarrow v_2$ there is precisely one $n_2 \in N$ with $n_1 \rightsquigarrow n_2$ and $\lambda(n_2) = v_2$.

(b) for all $n_2 \in N$ with $n_1 \rightsquigarrow n_2$ there is $v_2 \in V_O$ with $\lambda(n_1) \rightarrow v_2$ and $\lambda(n_2) = v_2$.

We relate unreachability of upward closed sets in WSTS to the *avoidability problem* in and-or graphs. The problem takes as input an and-or graph $G = (V_A \uplus V_O, v_O, \rightarrow)$ and a set of vertices $E \subseteq V_A \uplus V_O$. The question is whether there is an execution tree $T = (N, n_r, \rightsquigarrow, \lambda)$ so that $\lambda(N) \cap E = \emptyset$. In this case, we say *$E$ is avoidable in $G$*. The avoidability problem can be shown to be complete for polynomial time P.

### *9.3.2* $Over(TS, \Gamma', L')$

Let $TS = (\Gamma, \gamma_0, \rightarrow, \leq)$ be a WSTS with ADL $(L, r)$ wrt. $(\Gamma, \leq)$. Consider *finite* sets $\Gamma' \subseteq \Gamma$ with $\gamma_0 \in \Gamma'$ and $L' \subseteq L$ with $\top \in L'$.

**Definition 9.6 (Overapproximation wrt. $\Gamma'$ and $L'$).** The *overapproximation of TS wrt. $\Gamma'$ and $L'$* is the and-or graph $Over(TS, \Gamma', L') := (V_A \uplus V_O, v_O, \rightarrow)$ defined by

**(A1)** $V_0 := \Gamma' \uplus L'$

**(A2)** $V_A := \{E \subseteq \Gamma' \uplus L' \mid E \neq \emptyset \text{ and } \nexists\, d_1, d_2 \in E : r(d_1) \subseteq r(d_2)\}$

**(A3)** $v_O := \gamma_0$

**(A4)** For all $v_1 \in V_A$ and $v_2 \in V_O$ we have $v_1 \rightarrow v_2$ iff $v_2 \in v_1$.

**(A5)** For all $v_1 \in V_O$ and $v_2 \in V_A$ we have $v_1 \rightarrow v_2$ iff $suc(r(v_1)) \subseteq r(v_2)$ and there is no $v \in V_A$ with $suc(r(v_1)) \subseteq r(v) \subsetneq r(v_2)$.

By Condition **(A1)**, or-nodes are configurations in $\Gamma'$ or limits in $L'$. And-nodes, defined by **(A2)**, are sets of configurations and limit elements. Sets arise for two reasons. First, due to non-determinism a configuration may have several successors. Second, as discussed above it may be unclear which limit elements to choose for the overapproximation of successors. By Condition **(A5)**, we select the most precise overapproximations of $suc(r(v_1))$.

The definition of and-or graphs requires each vertex to have a successor. To see that $Over(TS,\Gamma',L')$ obeys this constraint, consider an and-node. By definition, this is a non-empty set $E \subseteq \Gamma' \uplus L'$. By Condition **(A4)**, the transitions leaving and-nodes just select an element from $E$. For an or-node, observe that $\{\top\}$ is an and-node. It can be used to overapproximate $suc(r(v)) \neq \emptyset$ for any or-node $v \in V_O$. Non-emptiness holds by deadlock freedom.

To link unreachability of $I \subseteq \Gamma$ to avoidability in $Over(TS,\Gamma',L')$, we define the *set of vertices $V_I \subseteq V_A \uplus V_O$ that represent elements in $I$*:

$$V_I := \{v \in V_A \uplus V_O \mid r(v) \cap I \neq \emptyset\}.$$

To prove the overapproximation sound, we first show that it imitates the behaviour of $TS$. Note that the following statements holds for any choice of $\Gamma'$ and $L'$.

**Lemma 9.4.** *Let $\gamma_0 \to \ldots \to \gamma_k$ in TS. Then in every execution tree $T = (N, n_r, \rightsquigarrow, \lambda)$ of $Over(TS,\Gamma',L')$ there is a path $n_r \rightsquigarrow n_1 \rightsquigarrow \ldots \rightsquigarrow n_{2k}$ so that $\gamma_i \in r(\lambda(n_{2i}))$.*

So we use or-vertices $\lambda(n_{2i})$ to reflect configurations.

**Theorem 9.1 (Soundness).** *If $V_I$ is avoidable in $Over(TS,\Gamma',L')$ then $R(TS) \cap I = \emptyset$.*

*Proof.* We proceed by contraposition and assume $R(TS) \cap I \neq \emptyset$. Then there is a path $\gamma_0 \to^+ \gamma_k$ with $\gamma_k \in I$ in $TS$. By Lemma 9.4, every execution tree $T = (N, n_r, \rightsquigarrow, \lambda)$ of $Over(TS,\Gamma',L')$ contains a path $n_r \rightsquigarrow^+ n_{2k}$ with $\gamma_i \in r(\lambda(n_{2i}))$. We conclude $r(\lambda(n_{2k})) \cap I \neq \emptyset$ and so $\lambda(N) \cap V_I \neq \emptyset$. $\square$

The overapproximation is actually complete. In case of unreachability, the sets $\Gamma'$ and $L'$ can be chosen precise enough so as to avoid $V_I$. Precise enough here means that $CS(TS) \subseteq \Gamma' \uplus L'$. This is the key observation that distinguishes EEC from the acceleration approach. It is sufficient to overapproximate the coverability set, it is not necessary to compute it precisely.

**Theorem 9.2 (Completeness).** *Let $CS(TS) \subseteq \Gamma' \uplus L'$. If $R(TS) \cap I = \emptyset$ then $V_I$ is avoidable in $Over(TS,\Gamma',L')$.*

*Proof.* We compute an execution tree $T = (N, n_r, \rightsquigarrow, \lambda)$ so that all $n \in N$ satisfy

$$r(\lambda(n)) \subseteq r(CS(TS)).$$

Since $r(CS(TS)) = R(TS)\downarrow$ and since $R(TS)\downarrow \cap I = \emptyset$ if and only if $R(TS) \cap I = \emptyset$, we conclude $r(\lambda(n)) \cap I = \emptyset$. This means $\lambda(N) \cap V_I = \emptyset$, tree $T$ avoids $V_I$.

We construct the tree by induction on the number of layers of or- and and-vertices. In the base case, we start from the root with

$$r(\lambda(n_r)) = r(v_O) = r(\gamma_0) = \{\gamma_0\}\downarrow \subseteq r(CS(TS)).$$

We have to determine an and-vertex $v \in V_A$ with $v_0 \to v$ so that $r(v) \subseteq r(CS(TS))$. With such an and-vertex, we extend the execution tree by $n_r \rightsquigarrow n$ so that $\lambda(n) = v$.

To find a suitable and-vertex, it is sufficient to show that

$$suc(r(\gamma_0)) \subseteq r(CS(TS)).$$

Since and-vertices are most precise overapproximations, there is $v \in V_A$ with $v_O \to v$ that satisfies $r(v) \subseteq CS(TS)$. In the worst case, we select $CS(TS)$ itself.

To establish the inclusion, consider $\gamma \in r(\gamma_0) = \{\gamma_0\}\downarrow$ that takes a transition $\gamma \to \gamma'$ for some $\gamma' \in \Gamma$. By definition of WSTS, $\leq$ is a simulation relation and so $\gamma_0$ can imitate the transition. There is $\gamma'' \in \Gamma$ with $\gamma_0 \to \gamma''$ and $\gamma'' \geq \gamma'$. Since $\gamma'' \in r(CS(TS))$ and since $r(CS(TS))$ is downward closed, we have $\gamma' \in r(CS(TS))$.

The induction step is along similar lines. □

## 9.4 Overall Algorithm

EEC expects as input a WSTS $(\Gamma, \gamma_0, \to, \leq)$ with an ADL $(L, r)$ that are effective. For the iterative construction of under- and overapproximations, we additionally require $\Gamma$ and $L$ to be recursively enumerable. As a consequence of this, there is an infinite sequence of *finite* sets of configurations

$$\Gamma_0 \subseteq \Gamma_1 \subseteq \dots$$

with $\gamma_0 \in \Gamma_0$ that satisfies the following. For every $\gamma \in \Gamma$ there is $i \in \mathbb{N}$ so that $\gamma \in \Gamma_i$. Likewise, there is an infinite sequencce of finite sets of limits

$$L_0 \subseteq L_1 \subseteq \dots$$

so that $\top \in L_0$ and for every $l \in L$ there is $i \in \mathbb{N}$ so that $l \in L_i$. Then for any finite $\Gamma' \uplus L' \subseteq \Gamma \uplus L$ there is $j \in \mathbb{N}$ so that

$$\Gamma' \uplus L' \subseteq \Gamma_j \uplus L_j.$$

**Theorem 9.3.** *EEC terminates and returns reachable if $R(TS) \cap I \neq \emptyset$ and unreachable otherwise.*

*Proof.* Provided $\to$ is decidable, $Under(TS, \Gamma_i)$ is computable due to finiteness of $\Gamma_i$. With a decidable $\leq$, the test $R(Under(TS, \Gamma_i)) \cap I \neq \emptyset$ is also decidable. Similarly, $Over(TS, \Gamma_i, L_i)$ and $V_I$ are computable due to effectiveness. Avoidability of $V_I$ can then be checked in polynomial time.

**input** :

Finite representation of WSTS $TS = (\Gamma, \gamma_0, \rightarrow, \leq)$ with ADL $(L, r)$ for $(\Gamma, \leq)$ that are effective
Upward closed set $I \subseteq \Gamma$ represented by $min(I)$
Infinite sequence $\Gamma_0 \subseteq \Gamma_1 \subseteq \ldots$ of finite subsets of $\Gamma$ as discussed above
Infinite sequence $L_0 \subseteq L_1 \subseteq \ldots$ of finite subsets of $L$ as discussed above

```
begin
    i := 0
    while  true  do
        Compute Under(TS, Γ_i)                          //Expand
        Compute Over(TS, Γ_i, L_i)                      //Enlarge
        if  R(Under(TS, Γ_i)) ∩ I ≠ ∅  then             //Check
            return reachable
        else if  V_I avoidable in Over(TS, Γ_i, L_i)  then
            return unreachable
        end if
        i := i + 1
    end while
end
```

**Fig. 9.1** Expand, Enlarge, and Check.

For correctness, let $R(TS) \cap I \neq \emptyset$. Then $V_I$ is not avoidable in all $Over(TS, \Gamma_i, L_i)$ by soundness of overapproximation (applied in contraposition). By completeness of underapproximation, there is $j \in \mathbb{N}$ so that $R(Under(TS, \Gamma_j)) \cap I \neq \emptyset$. EEC returns reachable.

Let $R(TS) \cap I = \emptyset$. We have $R(Under(TS, \Gamma_i)) \cap I = \emptyset$ for all $i \in \mathbb{N}$ by soundness of underapproximation. But there is $j \in \mathbb{N}$ with $CS(TS) \subseteq L_j \uplus \Gamma_j$. By completeness of overapproximation, $V_I$ is avoidable in $Over(TS, \Gamma_j, L_j)$. EEC returns unreachable as desired. $\square$

# Part III
# Dynamic Networks and $\pi$-Calculus

Text.

# Chapter 10
# Introduction to $\pi$-Calculus

**Abstract** $\pi$-Calculus

The $\pi$-Calculus is a process algebra for modelling dynamic networks. The origins of process algebras date back to the 1970s with Hoare's *Communicating Sequential Processes (CSP)* and Milner's *Calculus of Communicating Systems (CCS)*. Both lines of research were devoted to the study of the semantics of concurrency — with the following observation. *Communication*, sending and simultaneous receiving of messages, is the fundamental computation mechanism in concurrent systems. More complex mechanisms, e.g., semaphores, can be derived from communications.

Communications exchange messages over channels. To transmit its IP address to a server located at some URL, a client uses the *output action* $\overline{url}\langle ip\rangle$. It sends the message *ip* on the channel *url*. The *input action* $url(x)$ of the server listens on channel *url* and replaces variable $x$ by the incoming message. The key idea is to let message and channel have the same type: they are just *names*. Therefore, a message that is received in one communication may serve as the channel in the following. We extend the model of the server to $S = url(x).\overline{x}\langle ses\rangle$. The server receives a channel $x$ on *url* from the client. As a reply it sends a session *ses* on the received channel, i.e., to the client. We also extend the client to receive the session: $C = \overline{url}\langle ip\rangle.ip(y)$.

Concurrent execution of client and server is reflected by *parallel composition*. In the scenario, the parallel composition is $C \mid S = \overline{url}\langle ip\rangle.ip(y) \mid url(x).\overline{x}\langle ses\rangle$. Since a communication of $C$ and $S$ forms a computation step, we derive the transition

$$\overline{url}\langle ip\rangle.ip(y) \mid url(x).\overline{x}\langle ses\rangle \rightarrow ip(y) \mid \overline{ip}\langle ses\rangle.$$

Note that the communication changes the link structure. While in $C \mid S$ client and server share channel *url*, they are connected by *ip* in the next step. The number of entities in the system stays constant. To also model *object creation*, the parallel composition can be nested under action prefixes. Therefore, the two characteristic features of dynamic networks are well-reflected in $\pi$-Calculus.

We focus on the *computational expressiveness* of dynamic networks, i.e., we study restrictions of $\pi$-Calculus that yield system classes with decidabile verification problems. Interestingly, dynamic networks require a new correctness criterion. They ask for proper connections among entities, different from the earlier systems where we focussed on proper interaction. As we shall see, also linkage problems relate to coverability.

The main insight is that, despite the unbounded number of components and links that may be generated, dynamic networks often feature a strong similarity inside its configurations. There often is a finite set of connection patterns that all components make use of. We exploit this observation to derive finite representation of dynamic networks. As we shall see, the requirement can also be weakened. We later consider architectures where only certain dependency chains are bounded, similar to what is the case in $n$-tier architectures.

## 10.1 Syntax

The basic elements of processes are *names* $a, b, x, y$ in the infinite *set of names* $\mathcal{N}$. They are used as channels and messages. The previously introduced output and input actions are *prefixes*

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

The *silent prefix* $\tau$ performs an internal action.

Let $\tilde{a}$ abbreviate a *finite sequence of names* $a_1, \ldots, a_n$. To define parameterized recursion, we use *process identifiers* $K, L$. A process identifier represents a process $P$ via a recursive definition $K(\tilde{x}) := P$, where the elements in $\tilde{x}$ are pairwise distinct. The term $K\lfloor\tilde{a}\rfloor$ is a *call* to the process identifier, which results in the process $P$ with the names $\tilde{x}$ replaced by $\tilde{a}$. The remaining operators are as follows.

Symbol $\mathbf{0}$ is the *stop process* without behaviour. A *prefixed process* $\pi.P$ offers $\pi$ for communication and behaves like $P$ when $\pi$ is consumed. The *choice* between prefixed processes is represented by $\pi.P + M$. If $\pi.P$ is chosen, the alternatives in $M$ are forgotten. In a *parallel composition* $P \mid Q$, the processes $P$ and $Q$ communicate via pairs of send and receive prefixes. The *restriction* operator $\nu a.P$ converts the name $a$ in $P$ into a private name. It is different from all other names.

**Definition 10.1.** The *set of all $\pi$-Calculus processes* $\mathcal{P}$ is defined inductively by

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

Every process relies on finitely many process identifiers $K$, each of which defined by an equation $K(\tilde{x}) := Q$.

We write $\pi$ instead of $\pi.\mathbf{0}$. A sequence of restrictions $\nu a_1 \dots \nu a_n.P$ is abbreviated by $\nu\tilde{a}.P$ with $\tilde{a} := a_1,\dots,a_n$. To avoid brackets, we define that (1) prefix $\pi$ and restriction $\nu a$ bind stronger than choice composition $+$ and (2) choice composition binds stronger than parallel composition $\mid$.

## 10.2 Names and Substitutions

We mentioned that a restricted name $\nu a$ is different from all other names in the process $P \in \mathscr{P}$ under consideration. To ensure this, we define $\nu$ to *bind* the name $a$. We then allow for renaming bound names by $\alpha$-conversion. Similarly, in a prefixed process $a(y).Q$ the receive action $a(y)$ *binds* the name $y$. Intuitively, $y$ is a variable which has not yet received a concrete value and therefore should be different from all other names in $P$. We refer to the set of bound names by $bn(P)$. A name that is not bound is said to be *free* and we denote the set of free names in $P$ by $fn(P)$.

Of particular interest to the theory we develop are the restricted names that are not covered by a prefix in the syntax tree. We call them *active restricted names* and denote them by $arn(P)$. In

$$\nu a.(\overline{a}\langle b\rangle.\nu c.\overline{a}\langle c\rangle \mid a(x) \mid K\lfloor b\rfloor)$$

the restriction $\nu a$ is active while $\nu c$ is not as it is covered by the prefix $\overline{a}\langle b\rangle$. Note that active restricted names are bound, $arn(P) \subseteq bn(P)$. Active restrictions connect the processes that use the name. In the example, $\nu a$ connects $\overline{a}\langle b\rangle.\nu c.\overline{a}\langle c\rangle$ and $a(x)$, but not $K\lfloor b\rfloor$. We formalise the idea of connecting processes by active restrictions in Section 11.1. Formally, we say process $P$ *uses name* $a$ if $a \in fn(P)$.

Since we will permit $\alpha$-conversion of bound names, the following constraints (1) and (2) can always be achieved.

We assume wlog. (1) that all bound names are different and (2) that bound names and free names do not interfere. (3) Defining equations $K(\tilde{x}) := P$ should not contribute names. Therefore, we require that $fn(P) \subseteq \tilde{x}$.

Technically, $\alpha$-conversion of a bound name $a$ to $c$ means changing $\nu a.P$ to $\nu c.P'$, where every free occurrence of $a$ in $P$ is replaced by $c$ in $P'$. For example, $\nu a.a(x)$ is $\alpha$-converted to $\nu c.c(x)$. To rename free names in a process, we use *substitutions*.

**Definition 10.2 ($\sigma : \mathscr{N} \to \mathscr{N}$).** A *substitution* is a mapping from names to names, $\sigma : \mathscr{N} \to \mathscr{N}$. Let $x\sigma$ denote the image of $x$ under $\sigma$. If we give domain and codomain, $\sigma : A \to B$ with $A, B \subseteq \mathscr{N}$, we demand $x\sigma \in B$ if $x \in A$ and $x\sigma = x$ otherwise. An explicitly defined substitution $\sigma = \{a_1,\dots,a_n/x_1,\dots,x_n\}$ maps $x_i$ to $a_i$, i.e., $\sigma : \{x_1,\dots,x_n\} \to \{a_1,\dots,a_n\}$ with $x_i\sigma = a_i$.

An application of a substitution $\sigma : A \to B$ to a process $P$ results in a new process $P\sigma$, where all free names in $P$ are changed according to $\sigma$.

To ensure that substitution $\sigma : A \to B$ does not introduce new bindings in process $P \in \mathscr{P}$, we assume that the names in $\sigma$ do not interfere with the bound names: $(A \cup B) \cap bn(P) = \emptyset$.

We formalize the application of substitutions.

**Definition 10.3 (Application of Substitutions).** Consider $\sigma : A \to B$ and $P \in \mathscr{P}$ with $(A \cup B) \cap bn(P) = \emptyset$. The *application of* $\sigma$ *to* $P$ yields $P\sigma \in \mathscr{P}$ defined by

$$\begin{aligned}
\mathbf{0}\sigma &:= \mathbf{0} & (\pi.P + M)\sigma &:= (\pi\sigma).(P\sigma) + (M\sigma)\\
\tau\sigma &:= \tau & K\lfloor\tilde{a}\rfloor\sigma &:= K\lfloor\tilde{a}\sigma\rfloor\\
x(y)\sigma &:= x\sigma(y) & (P \mid Q)\sigma &:= P\sigma \mid Q\sigma\\
\overline{x}\langle y\rangle\sigma &:= \overline{x\sigma}\langle y\sigma\rangle & (\nu a.P)\sigma &:= \nu a.(P\sigma).
\end{aligned}$$

## 10.3 Structural Congruence

To give an operational semantics to a process algebra, the behaviour of every process has to be defined. To keep the definition of the transition relation simple, Berry and Boudol suggested to define only the transitions of representative terms and use a second relation to link processes with representatives. By definition, a process then behaves like its representative. Intuitively, the definition of the operational semantics is factorized into the definition of a transition and a structural relation.

Berry and Boudol called the approach *chemical abstract machine* with the following idea. Processes are chemical molecules that change their structure. Changing the structure heats molecules up or cools them down. Only heated molecules react with one another, which changes their state.

The $\pi$-Calculus semantics that exploits the chemical abstract machine idea was introduced by Milner. He called the relation to identify processes with representatives *structural congruence* and the name is still in use. Many results in this part of the lecture exploit invariance of the transition relation under structural rewriting.

Before we turn to the definition of structural congruence $\equiv \subseteq \mathscr{P} \times \mathscr{P}$, we recall that a *congruence relation* is an equivalence which is compatible with the operators of the algebra under study. That $\equiv$ is an *equivalence* means we have

$$\begin{aligned}
&\forall P \in \mathscr{P} : P \equiv P && \text{(Reflexivity)}\\
&\forall P, Q \in \mathscr{P} : P \equiv Q \text{ implies } Q \equiv P && \text{(Symmetry)}\\
&\forall P, Q, R \in \mathscr{P} : P \equiv Q \text{ and } Q \equiv R \text{ implies } P \equiv R. && \text{(Transitivity)}
\end{aligned}$$

That structural congruence is a congruence means it is preserved under composition, using any of the operators:

$$\forall P,Q,M \in \mathscr{P} : \forall \pi : P \equiv Q \text{ implies } \pi.P + M \equiv \pi.Q + M$$
$$\forall P,Q,R \in \mathscr{P} : P \equiv Q \text{ implies } P \mid R \equiv Q \mid R$$
$$\forall P,Q \in \mathscr{P} : \forall a \in \mathscr{N} : P \equiv Q \text{ implies } \nu a.P \equiv \nu a.Q.$$

**Definition 10.4 (Structural Congruence).** *Structural congruence* $\equiv \subseteq \mathscr{P} \times \mathscr{P}$ is the least congruence on processes which allows for $\alpha$-converting bound names

$$\nu x.P \equiv \nu y.(P\{y/x\}) \qquad\qquad a(x).P \equiv a(y).(P\{y/x\}),$$

where in both cases $\{y\} \cap (fn(P) \cup bn(P)) = \emptyset$. Moreover, $+$ and $\mid$ are commutative and associative with $\mathbf{0}$ as neutral element,

$$\begin{array}{rr} M + \mathbf{0} \equiv M & M_1 + M_2 \equiv M_2 + M_1 \\ M_1 + (M_2 + M_3) \equiv (M_1 + M_2) + M_3 & \\ P \mid \mathbf{0} \equiv P & P_1 \mid P_2 \equiv P_2 \mid P_1 \\ P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3, & \end{array}$$

and restriction is a commutative quantifier that is absorbed by $\mathbf{0}$ and whose scope can be shrunk and extruded over processes not using the quantified name:

$$\begin{array}{rr} \nu x.\nu y.P \equiv \nu y.\nu x.P & \nu x.\mathbf{0} \equiv \mathbf{0} \\ \nu x.(P \mid Q) \equiv P \mid (\nu x.Q), \text{ if } x \notin fn(P). & \end{array}$$

The latter law is called *scope extrusion.*

Structural congruence preserves the free names in a process.

**Lemma 10.1 (Invariance of *fn* under $\equiv$).** *$P \equiv Q$ implies $fn(P) = fn(Q)$.*

## 10.4 Transition Relation

To define the behaviour of $\pi$-Calculus processes, we employ the *structural approach to operational semantics*. Plotkin argues that the states of a transition system, like that of a program or that of a $\pi$-Calculus process, have a syntactic structure. They are compositions of basic elements using a set of operators. He then proposes to define transitions between these structured states by a proof system: a transition exists iff it is provable in the proof system. In order to define the behaviour of every state, the proof system uses induction on their structure. It comprises (1) axioms that define the transitions of basic elements and (2) proof rules that define the transitions of composed states from the transitions of the operands. The benefit of structural operational semantics is their simplicity and elegance, combined with the ability to establish properties of transitions by induction on the derivations.

**Definition 10.5 (Transition Relation and System).** The *transition relation* $\rightarrow \subseteq \mathscr{P} \times \mathscr{P}$ is defined by the rules in Table 10.1. For a process $P \in \mathscr{P}$, we define the *set* *of reachable processes* to be $R(P) := \{Q \in \mathscr{P} \mid P \to^* Q\}$. The *transition system of P* factorizes along structural congruence, $T(P) := (R(P)/_{\equiv}, \to, [P])$ where $[Q] \to [Q']$ iff $Q \to Q'$.

$$\text{(Tau)} \quad \tau.P + M \to P$$

$$\text{(React)} \quad x(y).P + M \mid \overline{x}\langle z\rangle.Q + N \to P\{z/y\} \mid Q$$

$$\text{(Const)} \quad K\lfloor\tilde{a}\rfloor \to P\{\tilde{a}/\tilde{x}\}, \text{ if } K(\tilde{x}) := P$$

$$\text{(Par)} \quad \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad\qquad \text{(Res)} \quad \frac{P \to P'}{\nu a.P \to \nu a.P'}$$

$$\text{(Struct)} \quad \frac{P \to P'}{Q \to Q'}, \text{ if } P \equiv Q \text{ and } P' \equiv Q'.$$

**Table 10.1** Rules defining the transition relation $\to \subseteq \mathscr{P} \times \mathscr{P}$.

Different from Plotkin's classical approach where the proof system only relies on the transition relation, Definition 10.5 makes use of the chemical abstract machine idea (cf. Section 10.3). All rules except for (Struct) define the transitions of representative processes. Rule (Struct) then postulates that a process can do all transitions of the representative it is related to by structural congruence.

# Chapter 11
# A Petri Net Translation of $\pi$-Calculus

**Abstract** From $\pi$-Calculus to Petri nets

We develop a translation of $\pi$-Calculus processes into Petri nets. This allows us to reuse the techniques and tools that have been developed for the analysis of Petri nets for the verification of dynamic networks. The $\pi$-Calculus is Turing complete while finite Petri nets are not. Therefore, the translation yields infinite Petri nets for some processes. This means we relax the definition of Petri nets $N = (S, T, W, M_0)$ in that $S$, $T$, or $W$ may be infinite sets.

For process algebras, the investigation of automata-theoretic models has a long standing tradition. The classic question was to find representations that reflect the concurrency of processes. The translation considered here exploits the connections induced by restricted names, instead. Rather than understanding a process as a set of programs running concurrently, we understand it as a graph where the references to restricted names connect processes. We call this translation a *structural semantics* to distinguish it from classical concurrency semantics. The benefit of taking the viewpoint of structure instead of concurrency are finite net representations for processes with unboundedly many restricted names and unbounded parallelism. We outline the intuition behind the translation.

The graph interpretation of a $\pi$-Calculus process $P \in \mathscr{P}$ is a hypergraph $\mathscr{G}(P)$ that makes the use of active restricted names explicit. A hypergraph is a graph where several vertices may be connected to a single so-called hyperedge. The interpretation of a process is obtained as follows. We draw a vertex labelled by $Q$ for every process $Q = M$ with $M \neq \mathbf{0}$ and for every $Q = K\lfloor \tilde{a} \rfloor$ in $P$. We then add a hyperedge labelled by $a$ for every active restricted name $\nu a$. An arc is inserted between a vertex $Q$ and an edge $a$ if the name is free in the process, $a \in fn(Q)$. Due to process creation, process destruction, and name passing this graph structure changes during system execution. We illustrate the interpretation on an example.

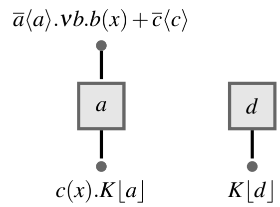



**Fig. 11.1** Graph interpretation of a $\pi$-Calculus process

*Example 11.1.* Let $P = \nu a.(\overline{a}\langle a\rangle.\nu b.b(x)+\overline{c}\langle c\rangle \mid c(x).K\lfloor a\rfloor \mid \nu d.K\lfloor d\rfloor)$. The graph interpretation $\mathscr{G}(P)$ is given in Figure 11.1. Note that the choice $\overline{a}\langle a\rangle.\nu b.b(x)+\overline{c}\langle c\rangle$ is represented by one vertex which is connected with $a$, although the alternative $\overline{c}\langle c\rangle$ does not contain $a$ as a free name. Furthermore, there is no hyperedge $c$ as the name is free in the process.

In the example, process $P$ is represented by two unconnected graphs. This means dynamic networks consist of independent parts that only communicate over public channels. The idea of the structural semantics is to represent each such graph by a place in a Petri net. We then obtain the current process by putting tokens on the places, one for each occurrence of the corresponding graph. Technically, we do not work with graphs but transform every process into a normal form.

## 11.1 Restricted Form

The restricted form captures the intuition of unconnected graphs discussed above. It serves the definition of the structural semantics and also helps in the definition of the characteristic functions depth and breadth. The idea of the restricted form is to minimize the scopes of active restricted names. This results in a process where the topmost parallel components correspond to the unconnected graphs. We call them *fragments*. The decomposition function that we define in Section 11.2 then counts how often a fragment occurs in a process in restricted form. It acts as a marking in the structural semantics.

**Definition 11.1 (Fragments and Restricted Form).** *Fragments* in the set $\mathscr{P}_{fg}$ are defined inductively by

$$F ::= M \mid K\lfloor\tilde{a}\rfloor \mid \nu a.(F_1 \mid \ldots \mid F_n),$$

where $M \neq \mathbf{0}$ and $a \in fn(F_i)$ for all $1 \leq i \leq n$. A process $P^{rf} = \Pi_{i\in I}F_i$ is *in restricted form*. The *set of all processes in restricted form* is $\mathscr{P}_{rf}$.

In case the index set is empty, we define $P^{rf} = \Pi_{i\in\emptyset}F_i$ to be $\mathbf{0}$. This means $\mathbf{0} \in \mathscr{P}_{rf}$. Function $fg\left(P^{rf}\right)$ determines the set of fragments in a process in restricted form.

For $\Pi_{i\in I}F_i$, we often refer (1) to the fragments $F_i$ that contain some name $a$ and (2) to those that are structurally congruent with a given fragment $F$. To determine these fragments, we define subsets $I_a$ and $I_F$ of the index set $I$.

**Definition 11.2 ($I_a, I_F$).** Consider process $\Pi_{i\in I}F_i$ in $\mathscr{P}_{rf}$. For every name $a \in \mathscr{N}$, we define the index set $I_a \subseteq I$ by $i \in I_a$ if and only if $a \in fn(F_i)$. For every fragment $F \in \mathscr{P}_{fg}$, we define $I_F \subseteq I$ by $i \in I_F$ if and only if $F \equiv F_i$.

To transform a process into restricted form via structural congruence, we employ the recursive function $rf : \mathscr{P} \to \mathscr{P}_{rf}$. It uses the rule for scope extrusion to shrink the scopes of restrictions and removes parallel compositions of stop processes $\mathbf{0}$.

**Definition 11.3 ($rf : \mathscr{P} \to \mathscr{P}_{rf}$).** The function $rf$ in Table 11.1 computes for any process $P \in \mathscr{P}$ a process $rf(P)$ in restricted form, i.e., $rf(P) \in \mathscr{P}_{rf}$. We call $rf(P)$ the *restricted form of P*.

$$rf(M) := M \qquad rf(K\lfloor\tilde{a}\rfloor) := K\lfloor\tilde{a}\rfloor$$

$$rf(P \mid Q) := \begin{cases} \mathbf{0}, & \text{if } rf(P) = \mathbf{0} = rf(Q) \\ rf(P), & \text{if } rf(P) \neq \mathbf{0} = rf(Q) \\ rf(Q), & \text{if } rf(P) = \mathbf{0} \neq rf(Q) \\ rf(P) \mid rf(Q), & \text{if } rf(P) \neq \mathbf{0} \neq rf(Q) \end{cases}$$

$$rf(\nu a.P) := \begin{cases} rf(P), & \text{if } a \notin fn(P) \\ \nu a.rf(P), & \text{if } a \in fn(P) \text{ and (1)} \\ \nu a.\,(\Pi_{i\in I_a}F_i) \mid \Pi_{i\in I\setminus I_a}F_i, & \text{if } a \in fn(P) \text{ and (2)} \end{cases}$$

**Table 11.1** Definition of function $rf$. With $rf(P) = \Pi_{i\in I\neq\emptyset}F_i$, condition (1) requires that $I_a = I$ and (2) states that $I_a \neq I$.

The following lemma states that $rf(P)$ is in fact in restricted form and structurally congruent with $P$.

**Lemma 11.1.** *For process $P \in \mathscr{P}$ we have $rf(P) \in \mathscr{P}_{rf}$ and $rf(P) \equiv P$.*

The restricted form is only invariant under structural congruence up to reordering and rewriting of fragments. So $P \equiv Q$ does not imply $rf(P) = rf(Q)$ but it implies $rf(P) \equiv_{rf} rf(Q)$. Relation $\equiv_{rf}$ is defined as follows.

**Definition 11.4 (Restricted Equivalence).** The *restricted equivalence relation* $\equiv_{rf} \subseteq \mathscr{P}_{rf} \times \mathscr{P}_{rf}$ is the smallest equivalence on processes in restricted form that satisfies commutativity and associativity of parallel compositions,

$$P_1^{rf} \mid P_2^{rf} \equiv_{rf} P_2^{rf} \mid P_1^{rf} \qquad P_1^{rf} \mid (P_2^{rf} \mid P_3^{rf}) \equiv_{rf} (P_1^{rf} \mid P_2^{rf}) \mid P_3^{rf},$$

and that replaces fragments by structurally congruent ones,

$$F \mid P^{rf} \equiv_{rf} G \mid P^{rf},$$

where $F \equiv G$ and $P^{rf}$ is optional.

We illustrate the indicated relationship between $P \equiv Q$ and $rf(P)$ and $rf(Q)$ on an example.

*Example 11.2 (Invariance of rf under $\equiv$ up to $\equiv_{rf}$).* Consider the processes

$$\begin{aligned} P &= \nu a.(\overline{a}\langle a\rangle.\nu b.b(x)+\overline{c}\langle c\rangle \mid c(x).K\lfloor a\rfloor \mid \nu d.K\lfloor d\rfloor) \\ &\equiv \nu a.(c(x).K\lfloor a\rfloor \mid \overline{a}\langle a\rangle.\nu b.b(x)+\overline{c}\langle c\rangle \mid \nu d.K\lfloor d\rfloor) = Q. \end{aligned}$$

We compare the restricted forms:

$$\begin{aligned} rf(P) &= \nu a.(\overline{a}\langle a\rangle.\nu b.b(x)+\overline{c}\langle c\rangle \mid c(x).K\lfloor a\rfloor) \mid \nu d.K\lfloor d\rfloor \\ &\equiv_{rf} \nu a.(c(x).K\lfloor a\rfloor \mid \overline{a}\langle a\rangle.\nu b.b(x)+\overline{c}\langle c\rangle) \mid \nu d.K\lfloor d\rfloor = rf(Q). \end{aligned}$$

We have $rf(P) \neq rf(Q)$ but $rf(P) \equiv_{rf} rf(Q)$.

Proposition 11.1 states invariance of the restricted form up to restricted equivalence, $P \equiv Q$ implies $rf(P) \equiv_{rf} rf(Q)$. Even more, restricted equivalence characterizes structural congruence, i.e., also $rf(P) \equiv_{rf} rf(Q)$ implies $P \equiv Q$.

**Proposition 11.1 (Characterisation of $\equiv$ with $\equiv_{rf}$).** *For $P,Q \in \mathscr{P}$ we have $P \equiv Q$ if and only if $rf(P) \equiv_{rf} rf(Q)$.*

*Proof.* To show the implication from right to left we observe that all rules making up equivalence $\equiv_{rf}$ also hold for structural congruence. Thus, $rf(P) \equiv_{rf} rf(Q)$ implies $rf(P) \equiv rf(Q)$. Combined with $P \equiv rf(p)$ from Lemma 11.1, we get $P \equiv Q$ by transitivity of structural congruence. The reverse direction uses an induction on the derivations of structural congruence. ☐

## 11.2 Structural Semantics

We assign to every process $P$ a Petri net $N(P)$ as illustrated in Example 11.4. The places of the net are the fragments of all reachable processes. More precisely, we deal with classes of fragments under structural congruence.

We use two disjoint sets of transitions. Transitions of the first kind are pairs $([F],[Q])$ of places $[F]$ and processes $[Q]$, with the condition that $F \rightarrow Q$. These transitions reflect communications inside fragments. The second set of transitions contains pairs $([F_1 \mid F_2],[Q])$ where $[F_1]$ and $[F_2]$ are places and $F_1 \mid F_2 \rightarrow Q$. These transitions represent communications between fragments using public channels.

There is an arc from place $[G]$ to transition $([F],[Q])$ provided $G \equiv F$. If $G$ is structurally congruent with $F_1$ and $F_2$, there is an arc weighted two from place $[G]$ to transition $([F_1 \mid F_2],[Q])$. This models a communication of fragment $F_1$ with the structurally congruent fragment $F_2$ on a public channel. If $G$ is structurally congruent with $F_1$ or $F_2$, there is an arc weighted one from place $[G]$ to transition $([F_1 \mid F_2],[Q])$. In case $F_1 \not\equiv G \not\equiv F_2$, there is no arc.

The number of arcs from $([F],[Q])$ to place $[G]$ is determined by the number of occurrences of $G$ in the decomposition of $Q$. Similarly, the initial marking of the net is determined by the decomposition of the initial process $P$.

To capture the notion of process decomposition we define function $dec(P^{rf})$. It counts how many fragments of class $[F]$ are present in $P^{rf}$. For $P^{rf} = F \mid G \mid F'$ with $F \equiv F'$ and $F \not\equiv G$ we have $(dec(P^{rf}))([F]) = 2$, $(dec(P^{rf}))([G]) = 1$, and $(dec(P^{rf}))([H]) = 0$ with $F \not\equiv H \not\equiv G$.

**Definition 11.5** ($dec : \mathscr{P}_{rf} \to \mathbb{N}^{\mathscr{P}_{fg}/\equiv}$)**.** Consider $P^{rf} = \Pi_{i \in I} F_i$. We assign to $P^{rf}$ the function $dec(P^{rf}) : \mathscr{P}_{fg}/_{\equiv} \to \mathbb{N}$ via $(dec(P^{rf}))([F]) := |I_F|$.

The support of $dec(P^{rf})$ is always finite. This ensures process

$$\Pi_{[H] \in supp(dec(P^{rf}))} \Pi^{(dec(P^{rf}))([H])} H$$

is defined. Intuitively, the term selects a representative for each fragment and then rearranges the fragments so that the same representatives lie next to each other.

*Example 11.3 (Elementary Equivalence).* For process $P^{rf} = F \mid G \mid F'$ we choose $F$ as representative for $F \equiv F'$ and let $G \not\equiv F$ represent itself. We then have

$$F \mid G \mid F' \equiv_{rf} \Pi^2 F \mid \Pi^1 G = \Pi^{(dec(P^{rf}))([F])} F \mid \Pi^{(dec(P^{rf}))([G])} G.$$

This relationship holds in general.

**Lemma 11.2 (Elementary Equivalence).** *For $P^{rf} \in \mathscr{P}_{rf}$ we have*

$$P^{rf} \equiv_{rf} \Pi_{[H] \in supp(dec(P^{rf}))} \Pi^{(dec(P^{rf}))([H])} H \ .$$

That *dec* is invariant under restricted equivalence ensures the structural semantics is well-defined. That it even characterizes restricted equivalence is exploited in the proof of Theorem 11.1.

**Lemma 11.3.** $P^{rf} \equiv_{rf} Q^{rf}$ *if and only if* $dec(P^{rf}) = dec(Q^{rf})$.

We are now prepared to define the structural semantics.

**Definition 11.6.** The *structural semantics* translates process $P$ into the Petri net $N(P)$ as defined in Table 11.2. We call $N(P)$ the *structural semantics of* $P$.

Consider fragment $F_1$ with $F_1 \to Q$. It yields a transition $([F_1],[Q])$. But $F_1 \mid F_2$ also leads to $Q \mid F_2$ for every fragment $F_2$. Thus, we additionally have transitions $([F_1 \mid F_2],[Q \mid F_2])$ for every reachable fragment $[F_2]$. The situation is illustrated in Figure 11.2. The additional transitions do not change the transition system and we do not compute them. Excluding them by a side condition would complicate the proof of Theorem 11.1.

$$
\begin{aligned}
S &:= fg(rf(R(P)))/_{\equiv}\\
T &:= \{([F],[Q]) \in S \times \mathscr{P}/_{\equiv} \mid F \rightarrow Q\}\\
&\quad \cup \{([F_1 \mid F_2],[Q]) \in \mathscr{P}/_{\equiv} \times \mathscr{P}/_{\equiv} \mid [F_1],[F_2] \in S \text{ and } F_1 \mid F_2 \rightarrow Q\}\\
M_0 &:= dec(rf(P)).
\end{aligned}
$$

Consider place $[G] \in S$ and two transitions $([F],[Q]),([F_1 \mid F_2],[Q]) \in T$. The weight function $W$ is defined as follows:

$$
\begin{aligned}
W([G],([F],[Q])) &:= (dec(F))([G])\\
W([G],([F_1 \mid F_2],[Q])) &:= (dec(F_1 \mid F_2))([G])\\
W(([F],[Q]),[G]) &:= (dec(rf(Q)))([G])\\
W(([F_1 \mid F_2],[Q]),[G]) &:= (dec(rf(Q)))([G]).
\end{aligned}
$$

**Table 11.2** Definition of the structural semantics $N(P) = (S,T,W,M_0)$ of process $P$.

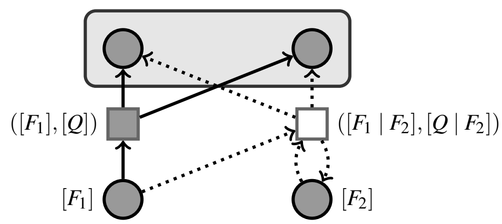



**Fig. 11.2** Illustration of the transitions $([F_1],[Q])$ and $([F_1 \mid F_2],[Q \mid F_2])$. The latter are depicted dotted and can be avoided in the construction.

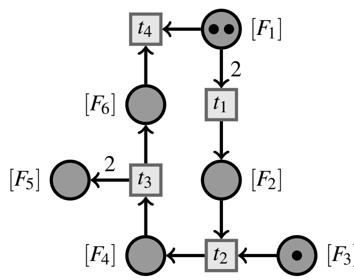
*Example 11.4 (Structural Semantics).* We illustrate the Petri net translation on an example. Consider

$$P = \Pi^2 a(x).x(y).y(z).\overline{a}\langle d\rangle + \overline{a}\langle b\rangle \mid \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle.(c(x) \mid c(x)).$$

The semantics $N(P)$ is depicted in Figure 11.3. The reachable fragments are given by the transition sequence

$$
\begin{aligned}
&\Pi^2 a(x).x(y).y(z).\overline{a}\langle d\rangle + \overline{a}\langle b\rangle \mid \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle.(c(x) \mid c(x))\\
\rightarrow\ & b(y).y(z).\overline{a}\langle d\rangle \mid \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle.(c(x) \mid c(x))\\
\rightarrow\ & \nu h.(h(z).\overline{a}\langle d\rangle \mid \overline{h}\langle b\rangle.(c(x) \mid c(x)))\\
\rightarrow\ & \overline{a}\langle d\rangle \mid c(x) \mid c(x).
\end{aligned}
$$

Since all processes are in restricted form, we can take their fragments as the set of places. The transitions are as follows. Fragment $F_1$ communicates with a structurally congruent fragment, $t_1 = ([F_1 \mid F_1],[F_2])$. Fragment $F_3$ passes the restricted name $h$ to $F_2$, which results in fragment $F_4 = \nu h.(h(z).\overline{a}\langle d\rangle \mid \overline{h}\langle b\rangle.(c(x) \mid c(x)))$. Transition $t_2 = ([F_2 \mid F_3],[F_4])$ models this communication. It demonstrates how the scope of restricted names influences the Petri net semantics. A pair of processes is represented by a single token on place $[F_4]$. Fragment $F_4$ lets its two processes communicate on the restricted channel $h$, which yields $Q = \overline{a}\langle d\rangle \mid c(x) \mid c(x) = F_6 \mid F_5 \mid F_5$. By definition, we get the transition $t_3 = ([F_4],[Q])$. The transition shows how fragments consisting of several processes break up when restricted names are forgotten.




$F_1 = a(x).x(y).y(z).\overline{a}\langle d\rangle + \overline{a}\langle b\rangle$ $\quad F_2 = b(y).y(z).\overline{a}\langle d\rangle$

$F_3 = \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle.(c(x) \mid c(x))$ $\quad F_4 = \nu h.(h(z).\overline{a}\langle d\rangle \mid \overline{h}\langle b\rangle.(c(x) \mid c(x)))$

$F_5 = c(x)$ $\quad F_6 = \overline{a}\langle d\rangle$

**Fig. 11.3** The structural semantics $N(P)$ of process $P$ in Example 11.4. The meaning of transitions is explained in the text.

The definition of the set of transitions does not take the overall process behaviour into account. The Petri net may contain transitions that are never enabled. Transition $t_4$ illustrates this fact. The fragments $F_1$ and $F_6$ communicate to $G = d(y).y(z).\overline{a}\langle d\rangle$. This results in $t_4 = ([F_1 \mid F_6],[G])$. The transition is never executed since the reaction is not possible in $P$. Since $G$ is no reachable fragment, $(dec(G))([F]) = 0$ for all places $[F]$, so transition $t_4$ has no places in its postset.

The example suggests the following rules of thumb for the structural semantics.

*Remark 11.1. Passing* restricted names *merges* fragments. If fragment $F$ passes a restricted name $\nu a$ to fragment $G$, this may result in a new fragment $\nu a.(F' \mid G')$ and we have a transition from the places $[F]$ and $[G]$ to place $[\nu a.(F' \mid G')]$. Transition $t_2$ in Example 11.4 illustrates the behaviour.

*Forgetting* restricted names *splits* fragments. If fragment $F$ forgets the restricted name $a$ when it evolves to $F'$, fragment $\nu a.(F \mid G)$ reacts to $F' \mid \nu a.G$. This results in a transition with $[\nu a.(F \mid G)]$ in its preset and $[F']$ and $[\nu a.G]$ in its postset. Transition $t_3$ in Example 11.4 serves as an example.

To ensure that our semantics is a suitable representation of $\pi$-Calculus processes, we show that we can retrieve all information about a process and its transitions from the semantics. To relate a marking in the Petri net $N(P)$ and a process, we define the function *retrieve* : $R(N(P)) \to \mathscr{P}/_{\equiv}$. It constructs a process from a marking by composing (1) the fragments that are marked in parallel (2) as often as required by the marking. This mimics the construction in the elementary equivalence.

**Definition 11.7.** Given a process $P \in \mathscr{P}$, the function *retrieve* : $R(N(P)) \to \mathscr{P}/_{\equiv}$ associates with every marking reachable in the structural semantics, $M \in R(N(P))$, a process class $[Q] \in \mathscr{P}/_{\equiv}$ as follows:

$$retrieve\,(M) := [\Pi_{[H]\in supp(M)}\Pi^{M([H])}H].$$

The support of $M$ has to be finite to ensure *retrieve* $(M)$ is a process. This holds since every transition has a finite postset and the initial marking is finite.

The transition systems of $P$ and $N(P)$ are isomorphic. Furthermore, the states in both transition systems correspond using the retrieve function. This relationship is illustrated in Figure 11.4.

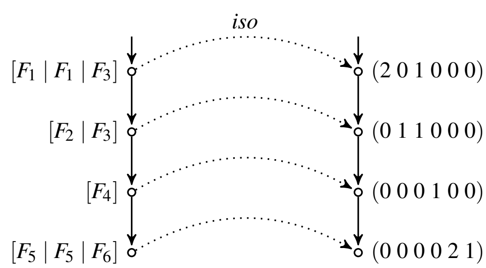



**Fig. 11.4** Illustration of the transition system isomorphism in Theorem 11.1 on process $P$ from Example 11.4. The transition system $T(P)$ is depicted to the left, $T(N(P))$ is depicted to the right. The isomorphism $iso : R(P)/_{\equiv} \to R(N(P))$ is represented by dotted arrows.

**Theorem 11.1.** *The transition systems of $P \in \mathscr{P}$ and its structural semantics $N(P)$ are isomorphic. The isomorphism iso : $R(P)/_{\equiv} \to R(N(P))$ maps $[Q]$ to $dec(rf(Q))$. A process is reconstructed from a marking by retrieve $(iso([Q])) = [Q]$.*

To prove the theorem one shows that *retrieve* is the inverse of *iso* and that *iso* is an isomorphism between the transition systems, i.e., *iso* maps the initial process to the initial marking, *iso* is bijective, and *iso* is a strong graph homomorphism. A strong graph homomorphism requires that $[P_1] \to [P_2]$ in the transition system of $P$ if and only if $iso([P_1]) \to iso([P_2])$ in the transition system of $N(P)$.

The definition of the structural semantics is declarative as it refers to the set of all reachable fragments and adds transitions where appropriate. In the following section, we comment on the implementation.

# Chapter 12
# Structural Stationarity

**Abstract** Finiteness of $N(P)$

We proposed a Petri net semantics of $\pi$-Calculus that highlights the connection structure of processes. Since the $\pi$-Calculus is Turing complete but finite Petri nets are not, such a semantics has to yield infinite nets for some processes. The goal of this section is to understand the sources of infinity for the structural semantics. Our interest in finiteness is based on the observation that all automated verification methods for Petri nets rely on this constraint. Ultimately this research will lead us to the borderline of decidability for dynamic networks.

For simplicity, call a process $P \in \mathscr{P}$ *structurally stationary* if its Petri net $N(P)$ is finite. We obtain two alternative characterizations of structural stationarity that refer to the parallel composition and to the restriction operator, respectively. The first characterization eases the proof of structural stationarity. The second one reveals that infinity of the semantics has two sources: unbounded breadth and unbounded depth. Unbounded breadth is caused by unbounded distribution of restricted names. Unbounded depth is caused by unboundedly long chains of processes connected by restricted names. In particular, unbounded name and unbounded process creation do not necessarily imply infinite automata-theoretic representations.

## 12.1 Structural Stationarity and Finiteness

Intuitively, a process is structurally stationary if there is a finite number of fragments in the system. Technically, there is a finite set of fragments so that the restricted form of all reachable processes is a parallel composition of those fragments.

**Definition 12.1.** Process $P \in \mathscr{P}$ is *structurally stationary* if

$$\exists \{F_1, \ldots, F_n\} \subseteq \mathscr{P}_{fg} : \forall Q \in R(P) : \forall F \in fg\,(rf(Q)) : \exists i \in [1,n] : F \equiv F_i.$$

The *set of all structurally stationary processes is* $\mathscr{P}_{fg<\infty}$.

Lemma 12.1 states the equivalence between finiteness of the structural semantics and structural stationarity mentioned in the introduction.

**Lemma 12.1 (Finiteness).** *$N(P)$ is finite if and only if $P \in \mathscr{P}_{fg<\infty}$.*

*Proof.* Finiteness of $N(P) = (S,T,W,M_0)$ is equivalent to finiteness of the set of places $S = fg\,(rf(R(P)))/_{\equiv}$. Finiteness of $fg\,(rf(R(P)))/_{\equiv}$ is equivalent to structural stationarity. ☐

To prove structural stationarity is not easy. The difficult part is to come up with a suitable set of fragments $\{F_1,\ldots,F_n\}$. The characterization in Section 12.3 reduces this task to finding a bound on the number of sequential processes in every reachable fragment. To establish completeness of this characterization, i.e., to show structural stationarity from boundedness, we in fact have to construct a finite set of fragments. The benefit is that we do this construction once when proving Theorem 12.1. When the characterization has been established, we simply apply it whenever we show structural stationarity. The construction relies on the notion of derivatives.

## 12.2 Derivatives

The derivatives of a process $P$ can be understood as a *finite* skeleton for all reachable processes. More formally, we show that all reachable processes are created from derivatives via parallel composition, restriction, and substitution. The corresponding Proposition 12.1 is crucial in the proof of Theorem 12.1.

The derivatives are constructed by recursively removing all prefixes from $P$ as if they were consumed in communications. If a process identifier $K$ is called, directly in $P$ or indirectly in one of its defining equations, we also add the derivatives of the process defining $K$.

**Definition 12.2.** We rely on the auxiliary function $der : \mathscr{P} \to \mathbb{P}(\mathscr{P})$ defined by

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

The set of *derivatives* of $P \in \mathscr{P}$, denoted by *derivatives*$(P)$, is the smallest set so that (1) $der(P) \subseteq derivatives(P)$ and (2) if $K\lfloor\tilde{a}\rfloor \in derivatives(P)$ and $K(\tilde{x}) := Q$ then also $der(Q) \subseteq derivatives(P)$.

There are two differences between the derivatives and the processes obtained by transitions. Names $y$ that are replaced by received names when an action $b(y)$ is consumed remain unchanged in the derivatives. Parameters $\tilde{x}$ that are instantiated to $\tilde{a}$ in a call $K\lfloor\tilde{a}\rfloor$ are not replaced in the derivatives. Both shortcomings are corrected by substitutions applied to the free names in the derivatives. Proposition 12.1 shows that this yields all reachable processes.

**Proposition 12.1.** *Every process $Q \in R(P)$ and every fragment $F \in fg(rf(Q))$ is structurally congruent to a process $\nu\tilde{a}.\big(\Pi_{i\in I}\, Q_i\sigma_i\big)$ where $Q_i \in derivatives(P)$ and $\sigma_i : fn(Q_i) \to fn(P) \cup \tilde{a}$.*

The following example provides some intuition to this technical statement.

*Example 12.1.* Consider $P = \nu b.\overline{a}\langle b\rangle.b(x) \mid a(y).K\lfloor a,y\rfloor$ with $K(a,y) := \overline{y}\langle a\rangle$. The only transition sequence is

$$\nu b.\overline{a}\langle b\rangle.b(x) \mid a(y).K\lfloor a,y\rfloor \to \nu b.(b(x) \mid K\lfloor a,b\rfloor) \to \nu b.(b(x) \mid \overline{b}\langle a\rangle) \to \mathbf{0}.$$

We compute the set of derivatives:

$$derivatives(P) = \{\overline{a}\langle b\rangle.b(x), b(x), a(y).K\lfloor a,y\rfloor, K\lfloor a,y\rfloor, \overline{y}\langle a\rangle\}.$$

The following congruences show that every reachable fragment can be constructed from the derivatives as stated in Proposition 12.1:

$$\begin{aligned}
\nu b.\overline{a}\langle b\rangle.b(x) &\equiv \nu b.((\overline{a}\langle b\rangle.b(x))\{a,b/a,b\}) \\
a(y).K\lfloor a,y\rfloor &\equiv (a(y).K\lfloor a,y\rfloor)\{a/a\} \\
\nu b.(b(x) \mid K\lfloor a,b\rfloor) &\equiv \nu b.(b(x)\{b/b\} \mid K\lfloor a,y\rfloor\{a,b/a,y\}) \\
\nu b.(b(x) \mid \overline{b}\langle a\rangle) &\equiv \nu b.(b(x)\{b/b\} \mid \overline{y}\langle a\rangle\{b,a/y,a\}).
\end{aligned}$$

In the proof of Theorem 12.1, finiteness of the set of derivatives is important.

**Lemma 12.2.** *The set $derivatives(P)$ is finite for all $P \in \mathscr{P}$.*

## 12.3 First Characterization of Structural Stationarity

We characterize structural stationarity as mentioned above: structural stationarity is equivalent to boundedness of all reachable fragments in the number of sequential processes. The *number of sequential processes in* $P \in \mathscr{P}$ is $\|P\|_S \in \mathbb{N}$ defined by $\|\mathbf{0}\|_S := 0$ and (with $M \neq \mathbf{0}$):

$$\begin{aligned}
\|M\|_S &:= 1 & \|P \mid Q\|_S &:= \|P\|_S + \|Q\|_S \\
\|K\lfloor\tilde{a}\rfloor\|_S &:= 1 & \|\nu a.P\|_S &:= \|P\|_S.
\end{aligned}$$

The function is invariant under structural congruence: $P \equiv Q$ implies $\|P\|_S = \|Q\|_S$. Bounding this number means we actually restrict the use of parallel composition. In Section 12.4, we establish a second characterization of structural stationarity, which restricts the use of operator $\nu$ instead. While the present characterization provides a handle to proving structural stationarity, the second characterization explains which processes fail to be structurally stationary.

**Definition 12.3.** A process $P \in \mathscr{P}$ is *bounded in the sequential processes* if there is a bound on the number of sequential processes in all reachable fragments:

$$\exists k_S \in \mathbb{N} : \forall Q \in R(P) : \forall F \in fg\,(rf(Q)) : \|F\|_S \leq k_S.$$

The *set of all processes that are bounded in the sequential processes* is $\mathscr{P}_{S<\infty}$.

**Theorem 12.1.** $\mathscr{P}_{fg<\infty} = \mathscr{P}_{S<\infty}$.

*Proof.* $\Rightarrow$ If $P \in \mathscr{P}_{fg<\infty}$ then all reachable processes are made up of finitely many fragments $F_1, \ldots, F_n$. Thus, the number of sequential processes in all reachable fragments is bounded by $max\{\|F_i\|_S \mid 1 \leq i \leq n\}$.

$\Leftarrow$ Let $P \in \mathscr{P}_{S<\infty}$ where $k_S \in \mathbb{N}$ is a bound on the number of sequential processes in fragments. We construct a finite set of fragments $FG$ that includes up to structural congruence every reachable fragment. The set $FG$ is defined as a union

$$FG := \bigcup_{i=1}^{k_S} FG_i.$$

The idea is that $FG_i$ only contains fragments with $i \in \mathbb{N}$ sequential processes. For the construction of suitable such fragments, we rely on Proposition 12.1. It provides processes $\nu\tilde{a}.(\Pi_{j=1}^{i} Q_j \sigma_j')$ that are sufficient to represent every reachable fragment. To ensure finiteness, we rename $\tilde{a}$ to distinguished names $\tilde{u}_i$ that we define below. We add the restricted form of $\nu\tilde{u}_i.(\Pi_{j=1}^{i} Q_j \sigma_j)$ to $FG_i$ provided it is a fragment:

$$FG_i := \big\{\, rf(\nu\tilde{u}_i.(\Pi_{j=1}^{i} Q_j \sigma_j)) \mid Q_j \in derivatives(P),\ \sigma_j : fn(Q_j) \to fn(P) \cup \tilde{u}_i, \\ \text{and } rf(\nu\tilde{u}_i.(\Pi_{j=1}^{i} Q_j \sigma_j)) \text{ is a fragment } \big\}.$$

We first show that $FG_i$ is finite for every $i \in \mathbb{N}$. The $Q_j$ are derivatives of $P$. This set is finite by Lemma 12.2. The same finiteness means that the maximum $maxFN := max\{|fn(Q)| \mid Q \in derivatives(P)\}$ exists. A parallel composition of $i$ derivatives thus restricts at most $i \cdot maxFN$ names. Hence, the names $\tilde{u}_i := u_1, \ldots, u_{i \cdot maxFN}$ are sufficient to reflect all restrictions. There are finitely many substitutions $\sigma_j : fn(Q_j) \to fn(P) \cup \tilde{u}_i$ between the finite sets $fn(Q_j)$ and $fn(P) \cup \tilde{u}_i$. This concludes the proof of finiteness for $FG_i$. Finiteness of $FG$ follows immediately.

It remains to show that up to structural congruence every reachable fragment $F$ is included in $FG$. With Proposition 12.1, $F$ is structurally congruent with a fragment $rf(\nu\tilde{u}_{|I|}.(\Pi_{i\in I} Q_i \sigma_i))$ in $FG_{|I|}$. The inclusion $FG_{|I|} \subseteq FG$ then follows from

$$|I| = \|\nu\tilde{u}_{|I|}.(\Pi_{i\in I} Q_i \sigma_i)\|_S = \|F\|_S \leq k_S.$$

The second equality is due to the invariance of $\|-\|_S$ under structural congruence. The inequality is the boundedness assumption. $\square$

We explain the construction of $FG$ on an example.

*Example 12.2 (FG).* Reconsider $P = \nu b.\overline{a}\langle b\rangle.b(x) \mid a(y).K\lfloor a,y\rfloor$ from Example 12.1 with $K(a,y) := \overline{y}\langle a\rangle$ . The number of sequential processes in all reachable fragments is bounded by $k_S = 2$. The set $FG$ is therefore defined as $FG = FG_1 \cup FG_2$. The maximal number of free names in derivatives is $maxFN = 2$. Thus, $FG_1$ and $FG_2$ contain fragments

$$rf(\nu u_1, u_2.(Q\sigma)) \quad \text{and} \quad rf(\nu u_1, \dots, u_4.(Q_1\sigma_1 \mid Q_2\sigma_2)),$$

where $Q \in derivatives(P)$ with $\sigma : fn(Q) \to \{u_1, u_2, a\}$ and $Q_j \in derivatives(P)$ with $\sigma_j : fn(Q_j) \to \{u_1, \dots, u_4, a\}$, for $j = 1,2$. As an example, consider process $Q = \overline{a}\langle b\rangle.b(x) \in derivatives(P)$. Applying the substitutions $\sigma : \{a,b\} \to \{u_1, u_2, a\}$ yields amongst others

$$\nu u_1.((\overline{a}\langle b\rangle.b(x))\{a, u_1/a, b\}) = \nu u_1.\overline{a}\langle u_1\rangle.u_1(x) \in FG_1.$$

The process is structurally congruent with the reachable fragment $\nu b.\overline{a}\langle b\rangle.b(x)$.

Several known subclasses of $\pi$-Calculus are immediately shown to be structurally stationary with Theorem 12.1. Furthermore, the proof of Theorem 12.2 underlines its importance.

## 12.4 Second Characterization of Structural Stationarity

The characterization of structural stationarity we develop in this section refers to the restriction operator. We observe that a bounded number of restricted names does not imply structural stationarity. In fact, a process with only one restricted name may not be structurally stationary. Consider $\nu a.K\lfloor a\rfloor$ with $K(x) := \overline{x}\langle x\rangle \mid K\lfloor x\rfloor$. It generates processes sending on the restricted channel $a$. The transition sequence

$$\nu a.K\lfloor a\rfloor \to \nu a.(\overline{a}\langle a\rangle \mid K\lfloor a\rfloor) \to \nu a.(\overline{a}\langle a\rangle \mid \overline{a}\langle a\rangle \mid K\lfloor a\rfloor) \to \dots$$

forms infinitely many fragments that are pairwise not structurally congruent. In the graph interpretation in Figure 12.1 there is no bound on the number of vertices connected with the hyperedge of name $a$, i.e., the degree of this edge is not bounded.

The degree of a hyperedge is the number of processes that share the name. To imitate this value at process level, we define $\|F\|_{\mid}$ the *maximal number of fragments under a restriction*. For example $\|\nu a.K\lfloor a\rfloor\|_{\mid} = 1$ and $\|\nu a.(\overline{a}\langle a\rangle \mid K\lfloor a\rfloor)\|_{\mid} = 2$. To reflect the maximum of the edge degrees, we search for the widest representation $F_B$ of a fragment $F$. Widest means that $\|F_B\|_{\mid}$ is maximal in the congruence class.

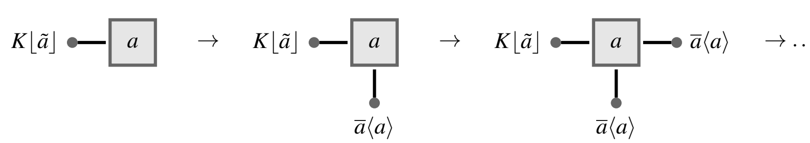



**Fig. 12.1** Transition sequence illustrating unbounded breadth

**Definition 12.4.** The *maximal number of fragments under a restriction* is defined inductively by $\|M\|_{\,|} := 1$, $\|K\lfloor\tilde{a}\rfloor\|_{\,|} := 1$, and

$$\|\nu a.(F_1 \mid \ldots \mid F_n)\|_{\,|} := max\{n, \|F_1\|_{\,|}, \ldots, \|F_n\|_{\,|}\}.$$

The *breadth* of fragment $F$ is $\|F\|_B := max\{\|G\|_{\,|} \mid G \equiv F\}$. Process $P \in \mathscr{P}$ is *bounded in breadth*, denoted by $P \in \mathscr{P}_{B<\infty}$, if the breadth of all reachable fragments is bounded:

$$\exists k_B \in \mathbb{N} : \forall Q \in R(P) : \forall F \in fg\,(rf(Q)) : \|F\|_B \leq k_B.$$

By definition, the breadth is invariant under structural congruence: $F \equiv G$ implies $\|F\|_B = \|G\|_B$. As it refers to all fragments in the congruence class, the notion is hard to grasp. We provide an example that illustrates the definition.

*Example 12.3 (Breadth).* Consider $\nu a.L\lfloor a\rfloor$ with $L(x) := \nu b.(\overline{x}\langle b\rangle \mid \overline{x}\langle b\rangle \mid L\lfloor x\rfloor)$. The only transition sequence is

$$\begin{aligned}\nu a.L\lfloor a\rfloor &\to \nu a.(\nu a_1.(\overline{a}\langle a_1\rangle \mid \overline{a}\langle a_1\rangle) \mid L\lfloor a\rfloor)\\ &\to \nu a.(\nu a_1.(\overline{a}\langle a_1\rangle \mid \overline{a}\langle a_1\rangle) \mid \nu a_2.(\overline{a}\langle a_2\rangle \mid \overline{a}\langle a_2\rangle) \mid L\lfloor a\rfloor) \to \ldots\end{aligned}$$

After $n \in \mathbb{N}$ transitions we have the following fragment $F_D \equiv F_B$:

$$\begin{aligned}F_D &= \nu a.(\Pi_{i=1}^{n} \nu a_i.(\overline{a}\langle a_i\rangle \mid \overline{a}\langle a_i\rangle) \mid L\lfloor a\rfloor)\\ F_B &= \nu a_1.(\ldots(\nu a_n.(\nu a.(\Pi_{i=1}^{n}(\overline{a}\langle a_i\rangle \mid \overline{a}\langle a_i\rangle) \mid L\lfloor a\rfloor)))\ldots).\end{aligned}$$

We have $\|F_D\|_{\,|} = n+1$ and $\|F_B\|_{\,|} = 2n+1$. In $F_B$ the number of fragments under a restriction is maximal in the congruence class of $F_D \equiv F_B$. So after $n$ transitions we have $\|F_D\|_B = \|F_B\|_B = \|F_B\|_{\,|} = 2n+1$. There is no bound on the breadth of the reachable fragments, $\nu a.L\lfloor a\rfloor \notin \mathscr{P}_{B<\infty}$.

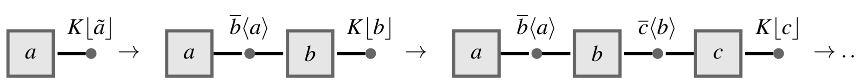



**Fig. 12.2** Transition sequence illustrating unbounded depth

Bounding the breadth of fragments is not sufficient to show structural stationarity. Consider $\nu a.K\lfloor a\rfloor$ with $K(x) := \nu b.(\overline{b}\langle x\rangle \mid K\lfloor b\rfloor)$. The process generates infinitely many fragments that are pairwise not structurally congruent but have breadth two:

$$\nu a.K\lfloor a\rfloor \to \nu a.(\nu b.(\overline{b}\langle a\rangle \mid K\lfloor b\rfloor)) \to \nu a.(\nu b.(\overline{b}\langle a\rangle \mid \nu c.(\overline{c}\langle b\rangle \mid K\lfloor c\rfloor))) \to \ldots$$

In the graphs in Figure 12.2, the length of the simple paths is not bounded. Recall that a path is simple if it does not repeat hyperedges. At process level, we mimic this length by the nesting of restrictions $\|F\|_\nu$. In the example, $\|\nu a.K\lfloor a\rfloor\|_\nu = 1$ and

$\|\nu a.(\nu b.(\overline{b}\langle a\rangle \mid K\lfloor b\rfloor))\|_\nu = 2$. To ensure the restrictions contribute to a simple path, we consider the flattest representation $F_D$ of $F$ where $\|F_D\|_\nu$ is minimal.

**Definition 12.5.** The *nesting of restrictions* $\|F\|_\nu$ is defined by $\|M\|_\nu := 0$ where $M \neq \mathbf{0}$, $\|K\lfloor\tilde{a}\rfloor\|_\nu := 0$, and

$$\|\nu a.(F_1 \mid \ldots \mid F_n)\|_\nu := 1 + max\{\|F_1\|_\nu, \ldots, \|F_n\|_\nu\}.$$

With this auxiliary function, the *depth* of $F$ is $\|F\|_D := min\{\|G\|_\nu \mid G \equiv F\}$. Process $P \in \mathscr{P}$ is *bounded in depth*, $P \in \mathscr{P}_{D<\infty}$, if

$$\exists k_D \in \mathbb{N} : \forall Q \in R(P) : \forall F \in fg\,(rf(Q)) : \|F\|_D \leq k_D.$$

Also the depth of fragments is invariant under structural congruence, $\|F\|_D = \|G\|_D$ for fragments $F \equiv G$. We continue with process $\nu a.L\lfloor a\rfloor$ from Example 12.3.

*Example 12.4 (Depth).* After $n \in \mathbb{N}$ transitions we have $F_D \equiv F_B$:

$$F_D = \nu a.(\Pi_{i=1}^n \nu a_i.(\overline{a}\langle a_i\rangle \mid \overline{a}\langle a_i\rangle) \mid L\lfloor a\rfloor)$$
$$F_B = \nu a_1.(\ldots(\nu a_n.(\nu a.(\Pi_{i=1}^n(\overline{a}\langle a_i\rangle \mid \overline{a}\langle a_i\rangle) \mid L\lfloor a\rfloor)))\ldots).$$

We have $\|F_D\|_\nu = 2$ and $\|F_B\|_\nu = n+1$.The nesting in $F_D$ is minimal in the class. Thus, $\|F_B\|_D = \|F_D\|_D = \|F_D\|_\nu = 2$. So the depth of all fragments reachable from $\nu a.L\lfloor a\rfloor$ is bounded by two, $\nu a.L\lfloor a\rfloor \in \mathscr{P}_{D<\infty}$.

There are at most $\|F\|_{\mid}$ fragments under a restriction. The nesting of restrictions is at most $\|F\|_\nu$. Thus, the number of sequential processes in $F$ is bounded as follows.

**Lemma 12.3.** $\|F\|_S \leq \|F\|_{\mid}^{\|F\|_\nu}$.

*Proof.* We proceed by an induction on the structure of fragments. In the base case, we have $F = M \neq \mathbf{0}$ and $F = K\lfloor\tilde{a}\rfloor$. The desired inequality holds with

$$\|F\|_S = 1 = 1^0 = \|F\|_{\mid}^{\|F\|_\nu}.$$

For the induction step, assume $\|F_i\|_S \leq \|F_i\|_{\mid}^{\|F_i\|_\nu}$ for all $F_i$ with $1 \leq i \leq n$. We then have for $F = \nu a.(F_1 \mid \ldots \mid F_n)$:

$$\begin{aligned}
&\|F\|_S\\
\{\text{ Def. } \|F\|_S \} &= \Sigma_{i=1}^n \|F_i\|_S\\
\{\text{ Hypothesis } \} &\leq \Sigma_{i=1}^n \|F_i\|_{\mid}^{\|F_i\|_\nu}\\
\{\text{ Def. } max \} &\leq \Sigma_{i=1}^n max\{\|F_i\|_{\mid} \mid 1 \leq i \leq n\}^{max\{\|F_i\|_\nu \mid 1 \leq i \leq n\}}.
\end{aligned}$$

Abbreviate $max_{\mid} := max\{\|F_i\|_{\mid} \mid 1 \leq i \leq n\}$ and $max_\nu := max\{\|F_i\|_\nu \mid 1 \leq i \leq n\}$. With this, the above term equals

$$
\begin{aligned}
& n \cdot max_{|}^{max_{\nu}} \\
\{ \text{ Def. } max \} \leq {} & max\{n, max_{|}\} \cdot max\{n, max_{|}\}^{max_{\nu}} \\
= {} & max\{n, max_{|}\}^{1+max_{\nu}} \\
\{ \text{ Def. } \|F\|_{\nu} \text{ and } \|F\|_{|} \} = {} & \|F\|_{|}^{\|F\|_{\nu}}.
\end{aligned}
$$

$\square$

Together, boundedness in breadth and in depth yield structural stationarity — the main result in this section. While the previous proof of structural stationarity from boundedness in the sequential processes was direct and cumbersome, Theorem 12.1 now yields an elegant proof of Theorem 12.2: a process is structurally stationary if and only if it is bounded in breadth and bounded in depth.

**Theorem 12.2.** $\mathscr{P}_{fg<\infty} = \mathscr{P}_{B<\infty} \cap \mathscr{P}_{D<\infty}$.

*Proof.* $\Rightarrow$ If the process is structurally stationary, there is a finite set of fragments $\{F_1, \ldots, F_n\}$ so that every reachable fragment is structurally congruent with an $F_i$. Then the maxima $max\{\|F_i\|_D \mid 1 \leq i \leq n\}$ and $max\{\|F_i\|_B \mid 1 \leq i \leq n\}$ exist and bound the depth and the breadth of all reachable fragments.

$\Leftarrow$ If we assume boundedness in breadth and depth there are $k_B$ and $k_D$ so that for all $Q \in R(P)$ and all $F \in fg\,(rf(Q))$ we have $\|F\|_B \leq k_B$ and $\|F\|_D \leq k_D$. We show that $k_B^{k_D}$ is a bound on the number of sequential processes. Consider $Q \in R(P)$ and $F \in fg\,(rf(Q))$. We determine the flattest representation $F_D \equiv F$ that satisfies $\|F_D\|_{\nu} = min\{\|G\|_{\nu} \mid G \equiv F\} = \|F\|_D$. We now have

$$
\begin{aligned}
& \|F\|_S \\
\{ \|-\|_S \text{ invariant under } \equiv \} = {} & \|F_D\|_S \\
\{ \text{ Lemma 12.3 } \} \leq {} & \|F_D\|_{|}^{\|F_D\|_{\nu}} \\
\{ \|F_D\|_{|} \leq max\{\|G\|_{|} \mid G \equiv F\} = \|F\|_B \} \leq {} & \|F\|_B^{\|F_D\|_{\nu}} \\
\{ \text{ Observation } \|F_D\|_{\nu} = \|F\|_D \} = {} & \|F\|_B^{\|F\|_D} \\
\{ k_B \text{ and } k_D \text{ bounds on breadth and depth } \} \leq {} & k_B^{k_D}.
\end{aligned}
$$

This proves $P$ is bounded in the number of sequential processes. With Theorem 12.1, $P$ is structurally stationary. $\square$

Theorem 12.2 helps disproving structural stationarity. A process is not structurally stationary if and only if it is not bounded in breadth or not bounded in depth. Thus, there are two sources of infinity for the structural semantics.

For processes of bounded depth but unbounded breadth, termination can be shown to be decidable by an instantiation of the WSTS framework. Processes of bounded breadth but unbounded depth are Turing complete. This follows from an encoding of counter machines that we present in the following chapter.

# Chapter 13
# Undecidability Results

**Abstract** Undecidability results for $\pi$-Calculus

There are several machine models with the ability to perform arithmetic operations on data variables, which are known to be Turing complete. For the undecidability proofs in this chapter, we use a model introduced by Minsky. Although Minsky called his formalism a *program machine* that operates on *registers*, the model is nowadays well-known under the name of *(2-)counter machines* acting on *counter variables*. We exploit Turing completeness of counter machines to show Turing completeness for processes of bounded breadth. As a consequence, we obtain undecidability of structural stationarity, boundedness in depth, and boundedness in breadth. We then change the encoding to establish undecidability of reachability for processes of depth one.

## 13.1 Counter Machines

A counter machine has two *counters* $c_1$ and $c_2$ that store arbitrarily large natural numbers and a *finite sequence of labelled instructions* $l : op$. There are two kinds of operations *op*. The first increments a counter, say $c_1$, by one and then jumps to the instruction labelled by $l'$:

$$c_1 := c_1 + 1 \text{ goto } l' \tag{13.1}$$

The second operation has the form

$$\text{if } c_1 = 0 \text{ then goto } l'; \text{ else } c_1 := c_1 - 1; \text{ goto } l''; \tag{13.2}$$

It checks counter $c_1$ for being zero and—if this is the case—jumps to the instruction labelled by $l'$. If the value of $c_1$ is positive, the counter is decremented by one and the machine jumps to $l''$.

**Definition 13.1.** A *counter machine* is a triple $CM = (c_1, c_2, instr)$ where $c_1, c_2$ are counters and $instr = l_0 : op_0; \dots, l_n : op_n; l_{n+1} : halt$ is a finite sequence of the labelled instructions defined above. The sequence ends with operation *halt* to terminate the execution.

To define the operational semantics of a counter machine *CM*, we define the notion of a configuration. A *configuration* of *CM* is a triple $cf = (v_1, v_2, l)$, where $v_i \in \mathbb{N}$ is the current value of counter $c_i$ with $i = 1, 2$ and $l \in \{l_0, \dots, l_{n+1}\}$ is the label of the operation to be executed next. A *run* of *CM* is a finite or infinite sequence of configurations

$$cf_0 \to cf_1 \to cf_2 \to \dots$$

subject to the following constraints. Initially, the counter values are zero and $l_0$ is the next instruction to be executed, $cf_0 = (0, 0, l_0)$. For every transition $cf_i \to cf_{i+1}$ with $cf_i = (v_1, v_2, l)$ the values of the counters and the instruction are changed according to the current operation *op* with $l : op$. In case *op* is an increment operation for the first counter as defined in (13.1), we have $cf_{i+1} = (v_1 + 1, v_2, l')$. This means value $v_1$ is incremented, $v_2$ is not changed, and the current label is changed to $l'$. The decrement operation on $c_1$ in (13.2) depends on whether $v_1 = 0$ holds. In this case, we jump to $l'$ without modifying the counter values, $cf_{i+1} = (v_1, v_2, l')$. If the content of $c_1$ is positive, we decrement it and jump to $l''$, which yields $cf_{i+1} = (v_1 - 1, v_2, l'')$. Action *halt* does not yield a transition.

We say *CM terminates* if all its runs are finite. A configuration $cf = (v_1, v_2, l)$ is *reachable* in *CM* if there is a run $cf_0 \to \dots \to cf_k = cf$ for some $k \in \mathbb{N}$. Since counter machines are Turing complete, termination and reachability are undecidable.

**Theorem 13.1 (Minsky 1967).** *Counter machines are Turing complete. Hence, for a counter machine CM it is undecidable whether (1) CM terminates and (2) whether a given configuration cf is reachable in CM.*

## 13.2 From Counter Machines to Bounded Breadth

The idea is to encode counters by lists. To fix the terminology, a list consists of *list elements*, namely several *list items* and one *list end*. The number of list items represents the value of the counter. Every list item and the list end has three channels to communicate on—reflecting the three operations on counters. Channel *i* is used for *increment* operations. Thus, a communication on *i* appends a list item to the list. Communications on channel *d decrement* the counter value. A message on *t* is a *test for zero*. We first explain the behaviour of a list item. To keep the definition short, we abbreviate $i, d, t$ by $\tilde{c}$. Similarly, the channels $i', d', t'$ of the following list element are abbreviated by $\tilde{c}'$. Since we are only interested in the channels, we omit parameters $x$ in send and receive actions $\bar{i}\langle x\rangle$ and $i(x)$:

$$LI(\tilde{c},\tilde{c}') := i.\overline{i'}.LI\lfloor\tilde{c},\tilde{c}'\rfloor + d.\left(\overline{d'}.LI\lfloor\tilde{c},\tilde{c}'\rfloor + \overline{t'}.LE\lfloor\tilde{c}\rfloor\right).$$

An increment operation received on channel $i$ is passed to the following list element with the send action $\overline{i'}$. As a list item stands for a positive counter value, the test for zero fails. A list item does not communicate on channel $t$. If a list item receives a decrement, it contacts the following list element. Since it is unknown whether this is a list item $LI$ or a list end $LE$, the current list item tries to communicate on both channels $\overline{d'}$ and $\overline{t'}$. If the next element is a list item, it answers the decrement call. A list end receives the $\overline{t'}$ message and, as a reaction, terminates. Now the current list item is the last element and therefore calls the defining equation $LE\lfloor\tilde{c}\rfloor$.

A list end answers a test for zero and terminates. As it represents value zero, it does not listen on the decrement channel. If the list end receives an increment, it creates new control channels $\tilde{c}' = i', d', t'$ and a new list end process $LE\lfloor\tilde{c}'\rfloor$. The former list end becomes a list item by calling the defining equation $LI\lfloor\tilde{c},\tilde{c}'\rfloor$:

$$LE(\tilde{c}) := t + i.\nu\tilde{c}'.(LI\lfloor\tilde{c},\tilde{c}'\rfloor \mid LE\lfloor\tilde{c}'\rfloor).$$

Every instruction $l : op$ of the counter machine translates into a process identifier $K_l$ whose defining process is determined by the operation $op$. For the increment operation (13.1) on counter $c_1$, we get

$$K_l(\tilde{c}_1,\tilde{c}_2) := \overline{i_1}.K_{l'}\lfloor\tilde{c}_1,\tilde{c}_2\rfloor.$$

The parameters $\tilde{c}_1 = i_1, d_1, t_1$ and $\tilde{c}_2 = i_2, d_2, t_2$ are the control channels of the lists that represent the counters $c_1$ and $c_2$, respectively.

The encoding of the decrement operation in (13.2) contains a subtlety. If the test for zero is successful, we delete the list end of counter $c_1$ and have to create a new one. This yields

$$K_l(\tilde{c}_1,\tilde{c}_2) := \overline{t_1}.\nu\tilde{c}_1'.\left(K_{l'}\lfloor\tilde{c}_1',\tilde{c}_2\rfloor \mid LE\lfloor\tilde{c}_1'\rfloor\right) + \overline{d_1}.K_{l''}\lfloor\tilde{c}_1,\tilde{c}_2\rfloor.$$

The instruction $l : halt$ is translated into $K_l(\tilde{c}_1,\tilde{c}_2) := \overline{halt}$. The send action will be helpful later to prove undecidability of boundedness in breadth.

To sum up, the counter machine $CM$ is translated into the process

$$P(CM) := \nu\tilde{c}_1.\nu\tilde{c}_2.(LE\lfloor\tilde{c}_1\rfloor \mid LE\lfloor\tilde{c}_2\rfloor \mid K_{l_0}\lfloor\tilde{c}_1,\tilde{c}_2\rfloor)$$

*Example 13.1.* Configuration $(2,0,l)$ of a counter machine is represented by

$$\nu\tilde{c}_1.\left[\nu\tilde{c}_1'.\left(LI\lfloor\tilde{c}_1,\tilde{c}_1'\rfloor \mid \nu\tilde{c}_1''.(LI\lfloor\tilde{c}_1',\tilde{c}_1''\rfloor \mid LE\lfloor\tilde{c}_1''\rfloor)\right) \mid \nu\tilde{c}_2.\left(LE\lfloor\tilde{c}_2\rfloor \mid K_l\lfloor\tilde{c}_1,\tilde{c}_2\rfloor\right)\right].$$

There are two list items in the list for $c_1$ to represent counter value two. Similarly, the list of counter $c_2$ consists of a single list end. The label of the current instruction can be deduced from the process identifier $K_l$.

Example 13.1 suggests a tight relationship between the configurations reachable in a counter machine *CM* and the processes reachable in its encoding $P(CM)$. We shall only need that the encoding preserves termination.

**Proposition 13.1.** *CM terminates if and only if* $P(CM)$ *terminates.*

The process representation of a counter machine is bounded in breadth by two. We exploit this observation in the following section to establish undecidability of boundedness in depth and breadth.

**Lemma 13.1.** *For every counter machine CM we have* $P(CM) \in \mathscr{P}_{B<\infty}$.

With proper synchronization mechanisms, the construction can be modified so that the steps of the counter machine coincide with step sequences of the corresponding process.

*Remark 13.1.* Processes of bounded breadth $\mathscr{P}_{B<\infty}$ are Turing complete.

## 13.3 Undecidability of Structural Stationarity

To show undecidability of structural stationarity for processes of bounded breadth, we reduce the termination problem for counter machines. This works as terminating processes are structurally stationary or, in contraposition, non-structurally stationary processes do not terminate. For structurally stationary processes we can use the structural semantics to decide termination.

**Proposition 13.2 (Undecidability of Structural Stationarity).** *For* $P \in \mathscr{P}_{B<\infty}$ *it is undecidable whether* $P \in \mathscr{P}_{fg<\infty}$ *holds.*

```
input :  CM a counter machine

begin
    compute  P(CM)
    if  ¬isStructurallyStationary(P(CM))  then
        return  CM does not terminate
    else
        compute  N(P(CM))
        return  terminates(N(P(CM)))
end
```

**Fig. 13.1** Proof of undecidability of structural stationarity. The procedure checks whether a counter machine terminates, assuming *isStructurallyStationary*$(-)$ decides structural stationarity for $\mathscr{P}_{B<\infty}$. Procedure *terminates*$(-)$ decides termination for Petri nets.

*Proof.* Assume structural stationarity is decidable for processes of bounded breadth using the procedure *isStructurallyStationary*$(-)$. Figure 13.1 gives an algorithm that then decides termination of a given counter machine *CM* as follows. We compute the process $P(CM) \in \mathscr{P}_{B<\infty}$. If the process is not structurally stationary it does not terminate. By Proposition 13.1, *CM* does not terminate.

If $P(CM)$ is a structurally stationary process, the structural semantics $N(P(CM))$ is a finite Petri net by Lemma 12.1. For finite Petri nets, termination is decidable. Moreover, the net terminates if and only if the counter machine does:

$$\begin{aligned} & CM \text{ terminates} \\ \{ \text{ Proposition 13.1 } \} \Leftrightarrow \; & P(CM) \text{ terminates} \\ \{ \text{ Theorem 11.1 } \} \Leftrightarrow \; & N(P(CM)) \text{ terminates.} \end{aligned}$$

Since termination of counter machines is undecidable, the assumption that structural stationarity is decidable for $\mathscr{P}_{B<\infty}$ has to be false. □

For a process of bounded breadth the condition of structural stationarity is equivalent to boundedness in depth according to Theorem 12.2. Since structural stationarity is undecidable, boundedness in depth is.

**Corollary 13.1 (Undecidability of Boundedness in Depth).** *Consider $P \in \mathscr{P}_{B<\infty}$. It is undecidable whether $P \in \mathscr{P}_{D<\infty}$ holds.*

To conclude the section, we reduce termination of *CM* to deciding boundedness in breadth. We again exploit the fact that our process representation $P(CM)$ of counter machines is bounded in breadth. The idea of the reduction is to compose $P(CM)$ in parallel with

$$halt.\nu a.K_{B=\infty}\lfloor a \rfloor.$$

When this process consumes $\overline{halt}$ it generates fragments of unbounded breadth. Consequently, *CM* terminates if and only if the parallel composition is not bounded in breadth.

**Lemma 13.2 (Undecidability of Boundedness in Breadth).** *For a process $P \in \mathscr{P}$ it is undecidable whether $P \in \mathscr{P}_{B<\infty}$ holds.*

*Proof.* Consider the counter machine *CM* and the process

$$P(CM) \mid halt.\nu a.K_{B=\infty}\lfloor a \rfloor$$

with $K_{B=\infty}(a) := \overline{a}\langle a \rangle \mid K_{B=\infty}\lfloor a \rfloor$. The counter machine terminates if and only if it reaches its *halt* operation. This is the case if and only if process $P(CM)$ sends $\overline{halt}$. Since $P(CM)$ is bounded in breadth, reachability of $\overline{halt}$ is equivalent to unboundedness in breadth for $P(CM) \mid halt.\nu a.K_{B=\infty}\lfloor a \rfloor$. □

## 13.4 Undecidability of Reachability in Depth 1

To establish undecidability of reachability for processes of depth one, we reduce the corresponding problem for counter machines. Since the resulting processes have to be bounded in depth, we can no longer represent counter values by lists. Instead, we use a different encoding that reflects counter values by parallel composition. For example, $c_1 = 3$ yields $\overline{a} \mid \overline{a} \mid \overline{a}$.

The problem with this representation is that parallel compositions, very similar to Petri nets, cannot faithfully model a test for zero:

$$l : \text{if } c_1 = 0 \text{ then goto } l'; \text{ else } c_1 := c_1 - 1 \text{ goto } l'';$$

To overcome this problem, we use the following trick. We implement a test for zero by a nondeterministic choice between a decrement and a test operation. If the test was done incorrectly (we branch to $l'$ although $c_1 > 0$), we reach an error process. From an error process we can never get back to a counter machine configuration. Technically, an error process leaves garbage $\nu a.(\overline{a} \mid \overline{a} \mid \overline{a})$ that cannot be removed. We turn to the construction.

We attach the processes $\overline{a}$ to a so-called *process bunch* $PB\lfloor a, i_{c_1}, d_{c_1}, t_{c_1} \rfloor$. To set up this link, we simply restrict the name $a$. For counter value $c_1 = 3$, this gives

$$\nu a.(PB\lfloor a, i_{c_1}, d_{c_1}, t_{c_1} \rfloor \mid \overline{a} \mid \overline{a} \mid \overline{a}).$$

Due to the restriction, the process bunch $PB\lfloor a, i_{c_1}, d_{c_1}, t_{c_1} \rfloor$ has exclusive access to its processes $\overline{a}$. It offers three operations to modify their numbers: $i_{c_1}$, $d_{c_1}$, and $t_{c_1}$. Communications on $i_{c_1}$ stand for *increment* and create a new process $\overline{a}$. Similarly, a message on $d_{c_1}$ *decrements* the process number by consuming a process $\overline{a}$. A *test for zero* on $t_{c_1}$ creates a new and empty process bunch for counter $c_1$. The old process bunch terminates. A term $\nu a.(\overline{a} \mid \overline{a} \mid \overline{a})$ without process bunch is the garbage that was mentioned above. The names $i_{c_1}$, $d_{c_1}$, and $t_{c_1}$ are free. Their index $c_1$ indicates that the process bunch models counter $c_1$. We abbreviate the parameter list by $\tilde{c}_x = i_{c_x}, d_{c_x}, t_{c_x}$ for $x \in \{1, 2\}$ and define

$$PB(a, \tilde{c}_x) \; := \; i_x.(PB\lfloor a, \tilde{c}_x \rfloor \mid \overline{a}) \; + \; d_x.a.PB\lfloor a, \tilde{c}_x \rfloor \; + \; t_x.\nu b.PB\lfloor b, \tilde{c}_x \rfloor.$$

The computational strength in this construction is in the process bunch deletion. This changes the linkage of an arbitrary number of processes $\overline{a}$ with a single transition.

The translation of the labelled instructions is similar to the one in Section 13.2. An increment operation $\quad l : c_1 := c_1 + 1 \text{ goto } l' \quad$ yields a process identifier

$$K_l(\tilde{c}_1, \tilde{c}_2) \; := \; \overline{i_{c_1}}.K_{l'}\lfloor \tilde{c}_1, \tilde{c}_2 \rfloor.$$

The test for zero discussed above yields a nondeterministic choice

$$K_l(\tilde{c}_1, \tilde{c}_2) \; := \; \overline{t_{c_1}}.K_{l'}\lfloor \tilde{c}_1, \tilde{c}_2 \rfloor \; + \; \overline{d_{c_1}}.K_{l''}\lfloor \tilde{c}_1, \tilde{c}_2 \rfloor.$$

A process bunch may accept a decrement although it is empty. In this case, the system deadlocks and reachability is preserved. A halt $l : halt$ translates into $K_l(\tilde{c}_1, \tilde{c}_2) := \overline{halt}$. The full translation of counter machine $CM$ is the process

$$P_1(CM) := \nu a.PB\lfloor a, \tilde{c}_1 \rfloor \mid \nu b.PB\lfloor b, \tilde{c}_2 \rfloor \mid K_{l_0}\lfloor \tilde{c}_1, \tilde{c}_2 \rfloor. \tag{13.3}$$

*Example 13.2.* Consider counter machine $CM = (c_1, c_2, instr)$ with

$$\begin{aligned}
&instr: \\
&\quad l_0 : c_1 := c_1 + 1 \text{ goto } l_1; \\
&\quad l_1 : \text{if } c_1 = 0 \text{ then goto } l_1; \text{ else } c_1 := c_1 - 1 \text{ goto } l_2; \\
&\quad l_2 : halt.
\end{aligned}$$

The machine sets $c_1$ to one, the following check for zero fails, $c_1$ is decremented, and the machine stops. The associated process $P_1(CM)$ has the form in (13.3) with the following defining equations:

$$\begin{aligned}
K_{l_0}(\tilde{c}_1, \tilde{c}_2) &:= \overline{i_{c_1}}.K_{l_1}\lfloor \tilde{c}_1, \tilde{c}_2 \rfloor \\
K_{l_1}(\tilde{c}_1, \tilde{c}_2) &:= \overline{t_{c_1}}.K_{l_1}\lfloor \tilde{c}_1, \tilde{c}_2 \rfloor + \overline{d_{c_1}}.K_{l_2}\lfloor \tilde{c}_1, \tilde{c}_2 \rfloor \\
K_{l_2}(\tilde{c}_1, \tilde{c}_2) &:= \overline{halt}.
\end{aligned}$$

The reachable states of $CM$ can be computed from the reachable processes of $P_1(CM)$. More precisely, the counter machine $CM$ reaches the state $(v_1, v_2, l)$ if and only if its encoding reaches the process

$$\nu a.(PB\lfloor a, \tilde{c}_1 \rfloor \mid \Pi^{v_1}\overline{a}) \mid \nu b.(PB\lfloor b, \tilde{c}_2 \rfloor \mid \Pi^{v_2}\overline{b}) \mid K_l\lfloor \tilde{c}_1, \tilde{c}_2 \rfloor.$$

Combined with the observation that $P_1(CM)$ is always bounded in depth by one, we arrive at the desired undecidability.

**Theorem 13.2.** *Consider two processes $P, Q \in \mathscr{P}_{D<\infty}$ where the depth is bounded by one. The problem whether $[Q] \in R(P)/_{\equiv}$ is undecidable.*

Theorem 13.2 implies undecidability of reachability for processes of bounded depth. Termination, in turn, can be shown to be decidable for $\mathscr{P}_{D<\infty}$. Since termination is undecidable for counter machines, the above encoding $P_1(CM)$ cannot preserve termination. Example 13.2 gives a counter machine $CM$ that terminates but whose process representation $P_1(CM)$ has an infinite run.

Since reachability is decidable for Petri nets, we conclude that there is no *reachability-preserving* translation into Petri nets for any class of processes that subsumes those of depth one.