k-Bounded Petri Net Synthesis from Modal Transition Systems*

Uli Schlachter[1] and Harro Wimmel[2]

1 **Department of Computing Science, Carl von Ossietzky Universität, Oldenburg, Germany**
uli.schlachter@informatik.uni-oldenburg.de
2 **Department of Computing Science, Carl von Ossietzky Universität, Oldenburg, Germany**
harro.wimmel@informatik.uni-oldenburg.de

Abstract

We present a goal-oriented algorithm that can synthesise k-bounded Petri nets ($k \in \mathbb{N}^+$) from hyper modal transition systems (hMTS), an extension of labelled transition systems with optional and required behaviour. The algorithm builds a potential reachability graph of a Petri net from scratch, extending it stepwise with required behaviour from the given MTS and over-approximating the result to a new valid reachability graph. Termination occurs if either the MTS yields no additional requirements or the resulting net of the second step shows a conflict with the behaviour allowed by the MTS, making it non-sythesisable.

1998 ACM Subject Classification D.2.2 Design Tools and Techniques, F.4.1 Mathematical Logic

Keywords and phrases Modal transition system, bounded Petri net, synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2017.6

1 Introduction

Petri net synthesis, or more precisely, the problem of finding an unlabelled Petri net implementing a given labelled transition system, goes back to Ehrenfeucht and Rozenberg [13] and was recently comprehensively presented by Badouel, Bernardinello and Darondeau [5]. Petri net synthesis not only yields implementations which are correct by design, but also allows to extract concurrency and distributability information from a sequential specification [6, 8, 21].

Modal transition systems (MTS) are a well-known and useful method for specifying systems [17, 1, 9, 16]. They are an extension of labelled transition systems (LTS) that can distinguish between required behaviour (must edges) and optional behaviour (may edges). Hyper MTS (called disjunctive MTS in [18]) are a further extension, where must edges are defined as hyper edges, requiring at least one out of (possibly) several actions. An LTS implements a (hyper) MTS if it allows the required behaviour and disallows any not specified (optional or required) behaviour, i.e. there may be more than one LTS implementing the same MTS.

Since modal transition systems are extensions of labelled transition systems, it has been suggested to extend Petri net synthesis to cover modal transition systems [12, 5]. The question here is if a given modal transition system can be realised by a Petri net, which means that the reachability graph of a Petri net can implement the modal transition system,

* This work is supported by the German Research Foundation (DFG) projects ARS and ASYST, reference numbers Be 1267/15-1 and Be 1267/16-1.

and if this is answered positively, to produce such a Petri net. Recent results show that this question is undecidable even when asking for pure Petri nets [14] or for bounded Petri nets [20].

We will easily see that for a given $k \in \mathbb{N}^+$ the synthesis of a k-bounded Petri net from some MTS (or some other specification) is decidable, since there is only a finite number of different reachability graphs available for k-bounded Petri nets with a fixed set of transitions. This is, to our knowledge, the first general positive decidability result for Petri net synthesis from modal specifications. Such a brute force approach is not advisable, though, due to the state space explosion with growing k and number of transitions.

Instead, we introduce a goal-oriented algorithm for synthesising k-bounded Petri nets from hyper MTS, where the value k is an input to the algorithm. Our algorithm iteratively applies two operations until a fixed point is reached. The first operation adds missing edges so that its input implements a given modal transition system. The second operation produces minimal Petri net solvable over-approximations of otherwise Petri net unsolvable labelled transition systems. The algorithm will construct all minimal Petri net realisations of a given modal transition system, where minimality is defined with respect to an LTS homomorphism preorder, similar to the simulation preorder. This preorder implies language inclusion, which means that the algorithm also calculates minimal realisations with respect to language inclusion.

The paper is organised as follows: In Section 2 we present the basic terminology regarding LTS and Petri nets, including LTS homomorphisms, the construction of k-bounded regions, and the synthesis of k-bounded Petri nets from such regions. In Section 3 we introduce (hyper) MTS and their relation to LTS. Section 4 contains our main algorithm including some examples. In Section 5 we prove termination and correctness of our algorithm. Section 6 contains some thought on the algorithmic complexity of the algorithm and how to implement it.

2 Petri Net Synthesis

▶ **Definition 1.** A *labelled transition system (LTS)* A is a structure $A = (Q, \Sigma, \to, q_0)$ where Q is a set of *states*, Σ is a set of *actions*, $q_0 \in Q$ is the *initial state* and $\to \subseteq Q \times \Sigma \times Q$ is a set of (action-)labelled *edges*. A tuple $(q, a, q') \in \to$ is also written as $q \xrightarrow{a} q'$ and $q \xrightarrow{a}$ expresses that some q' with $q \xrightarrow{a} q'$ exists. This notation is canonically extended to words $w \in \Sigma^*$ by inductively defining $q \xrightarrow{\varepsilon} q$ for all $q \in Q$ and $q \xrightarrow{wa} q' \iff \exists q'' \colon q \xrightarrow{w} q'' \wedge q'' \xrightarrow{a} q'$. An LTS is *deterministic* if it satisfies $\forall (q, a, q'), (q, a, q'') \in \to \colon q' = q''$. It is called *totally reachable* if $\forall q \in Q \colon \exists w \in \Sigma^* \colon q_0 \xrightarrow{w} q$. It is called *finite* if Q and Σ (and hence also $\to$) are finite. The language $L(A)$ of an LTS A is $L(A) := \{w \in \Sigma^* \mid q_0 \xrightarrow{w} \}$.

Given two LTS $A_i = (Q_i, \Sigma, \to_i, q_{0i})$ with $i \in \{1, 2\}$, an *LTS homomorphism from* A_1 *to* A_2 is a function $f \colon Q_1 \to Q_2$ with $f(q_{01}) = q_{02}$ such that $q \xrightarrow{a} q'$ implies $f(q) \xrightarrow{a} f(q')$ for all $q, q' \in Q_1$ and $a \in \Sigma$. If such a homomorphism f exists we write $A_1 \sqsubseteq A_2$ (via f). $\sqsubseteq$ is reflexive (with the identity homomorphism id) and transitive (as homomorphisms are closed under composition). A bijection f is an *isomorphism* if both, f and f^{-1}, are homomorphisms, in which case we abstract from state names of the isomorphic LTS A_1 and A_2 and identify them, writing $A_1 = A_2$.

▶ Remark. Unless given explicitly, the components of an LTS A will be named canonically as $A = (Q_A, \Sigma_A, \to_A, q_{0,A})$. The same notation will be used for other structures. For example, the set of places of a Petri net N, which will be defined later, is referred to as P_N. For

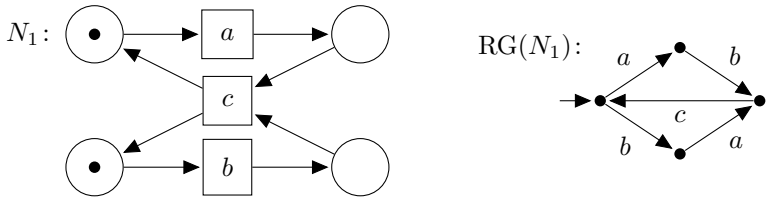

Figure 1 Example of a Petri net and its reachability graph. In the Petri net, places are drawn as circles containing tokens indicating the initial marking. Transitions are drawn as rectangles containing the name of the transition. The reachability graph has states shown as circles where the initial state is indicated by an arrow tip. Edges are drawn as arrows with the edge label next to it.

simplicity, the subscript will be left out if it is clear from the context. This last point will mainly be used for edges, e.g. for $(q, a, q') \in \rightarrow_A$ we might write $q \xrightarrow{a} q'$ instead of $q \xrightarrow{a}_A q'$.

▶ **Lemma 2.** *Let A and B be LTS so that $A \sqsubseteq B$ via f and $A \sqsubseteq B$ via f'. If A is totally reachable and B is deterministic, then $f = f'$.*

Proof. We prove by induction on the length of words $w \in L(A)$ that if $q_{0,A} \xrightarrow{w} q$, then $f(q) = f'(q)$. Since A is totally reachable, we reach all states of A in this way, showing that $f(q) = f'(q)$ for all $q \in Q_A$. The induction basis shows the conclusion for the initial state by applying the definition of $\sqsubseteq$: $f(q_{0,A}) = q_{0,B} = f'(q_{0,A})$.

For the induction step, assume that $q_{0,A} \xrightarrow{w} q$ with $f(q) = f'(q)$ and consider any $q \xrightarrow{a} q'$. Since $A \sqsubseteq B$ via f and f', we have $f(q) \xrightarrow{a} f(q')$ and $f'(q) \xrightarrow{a} f'(q')$. However, because B is deterministic, there cannot be two different states that are reached from $f(q) = f'(q)$ via label a. We conclude $f(q') = f'(q')$. ◀

▶ **Lemma 3.** *Let A and B be totally reachable and deterministic LTS so that $A \sqsubseteq B$ via f_A and $B \sqsubseteq A$ via f_B. Then $A = B$.*

Proof. Consider $A \sqsubseteq A$ (via id) and $A \sqsubseteq B \sqsubseteq A$ via $f_B \circ f_A$. Since Lemma 2 is applicable, we conclude $f_B \circ f_A = \mathrm{id}$, making f_A injective and f_B surjective. With an analogous argument for $f_A \circ f_B = \mathrm{id}$ we see that f_A and f_B are both bijective homomorphisms with $f_A^{-1} = f_B$, i.e. they are isomorphisms. Abstracting from state names for the isomorphic A and B, we may write $A = B$. ◀

▶ **Definition 4.** A *Petri net* is a tuple $N = (P, T, F, M_0)$ where P and T are finite and disjoint sets of *places* and *transitions*, respectively, and $F\colon ((P \times T) \cup (T \times P)) \to \mathbb{N}$ is a *flow relation* specifying *arc weights*. M_0 is the *initial marking* where a *marking* is a mapping $P \to \mathbb{N}$. A transition $t \in T$ is *enabled* in a marking M iff $\forall p \in P\colon F(p,t) \leq M(p)$. An enabled transition can *fire* leading to the marking M' defined by $M'(p) = M(p) - F(p,t) + F(t,p)$. We write $M[t\rangle M'$ and this syntax is inductively extended to label sequences $\sigma \in T^*$. The set of *reachable markings* of a Petri net N is $\mathcal{E}(N) = \{M \mid \exists \sigma \in T^*\colon M_0[\sigma\rangle M\}$. A Petri net is called *k-bounded by $k \in \mathbb{N}$* if $\forall M \in \mathcal{E}(N),\ p \in P\colon M(p) \leq k$. There is a *self-loop* around $p \in P$ and $t \in T$ if $F(p,t) > 0 \wedge F(t,p) > 0$. A Petri net is called *pure* if it does not contain any self-loops. The *reachability graph of N* is the LTS $\mathrm{RG}(N) = (\mathcal{E}(N), T, \rightarrow, M_0)$ with $\rightarrow = \{(M, t, M') \in \mathcal{E}(N) \times T \times \mathcal{E}(N) \mid M[t\rangle M'\}$. An LTS A is called *Petri net solvable* if a Petri net N exists with $A = \mathrm{RG}(N)$. In this situation, N *solves* A.

An example of a Petri net and its reachability graph is shown in Figure 1. The following lemma can easily be verified:

▶ **Lemma 5.** *For a Petri net N, $\mathrm{RG}(N)$ is totally reachable and deterministic. If N is k-bounded (for some $k \in \mathbb{N}^+$), $\mathrm{RG}(N)$ is finite.*

The next lemma shows that there are only finitely many LTS solvable via k-bounded Petri nets. As an immediate consequence, if we can decide whether the reachability graph of a specific k-bounded Petri net "implements" some specification, we can also decide whether any k-bounded Petri net fulfils this, i.e. brute force decisions are possible.

▶ **Lemma 6.** *For any given $k \in \mathbb{N}^+$ and finite set of transitions T, the number of structurally different reachability graphs of k-bounded Petri nets with transitions from T is finite.*

Proof. A weight $F(p,t) > k$ or $F(t,p) > k$ in a k-bounded Petri net means that the transition t can never fire (or the bound would be violated). With weights in $\{0,\dots,k\}$, a place can only be connected in finitely many ways to all the transitions. Exact duplicates of places do not have an effect on the structure of the reachability graph. ◀

A place p of a Petri net has a number of tokens $M(p)$ in each reachable marking M. The next definition formalises this concept on the reachability graph as a so-called region [7, 10, 11].

▶ **Definition 7.** Let $k \in \mathbb{N}^+$. A *k-bounded region of a finite*[1] *LTS A* is a function $r\colon Q_A \to \{0,1,\dots,k\}$ satisfying $\forall a \in \Sigma_A\colon \forall q \xrightarrow{a} q', q'' \xrightarrow{a} q'''\colon r(q') - r(q) = r(q''') - r(q'')$, which means that edges with the same label have the same *gradient*, where the *gradient of $a \in \Sigma_A$* is $\Delta_r(a) = r(q') - r(q)$ for any edge $q \xrightarrow{a} q'$. The value $\mu_r(a) = \min\{r(q) \mid q \xrightarrow{a}\}$ is the minimum value $r(q)$ of a state q with an outgoing edge with label a (generally assuming that every $a \in \Sigma_A$ occurs as a label of some edge).

A region corresponds to a possible place p of a Petri net with initial marking $M_0(p) = r(q_0)$ and flow relation $F(p,t) = \mu_r(t)$, $F(t,p) = \Delta_r(t) + \mu_r(t)$. This correspondence allows to define the Petri net $N(R)$ generated by a set R of regions of an LTS A via $N(R) = (R, \Sigma_A, F, M_0)$ with M_0 and F as above.

This definition adds the maximum possible number of self-loops to the generated place p, because this restricts the behaviour of the Petri net the most. Thus, places with fewer self-loops cannot be generated in this setting, but also will not be needed.

▶ **Lemma 8.** *Let A and B be finite LTS so that $A \sqsubseteq B$ via f. If r is a region of B, then $r' = r \circ f$ is a region of A.*

Proof. Let $q \xrightarrow{a} q'$ and $\widetilde{q} \xrightarrow{a} \widetilde{q}'$ be two edges of A. Then, $f(q) \xrightarrow{a} f(q')$ and $f(\widetilde{q}) \xrightarrow{a} f(\widetilde{q}')$ are edges in B and by the uniqueness of the gradient we get $r'(q') - r'(q) = r(f(q')) - r(f(q)) = r(f(\widetilde{q}')) - r(f(\widetilde{q})) = r'(\widetilde{q}') - r'(\widetilde{q})$. ◀

While r and r' have the same gradient $\Delta_r = \Delta_{r'}$, the values for μ_r and $\mu_{r'}$ may differ, i.e. the places of a Petri net constructed from these regions may have different self-loops.

By definition, there are only a finite number of k-bounded regions. We can use all of them and define a Petri net:

▶ **Definition 9.** For a finite LTS A and a number $k \in \mathbb{N}^+$, let $\mathrm{SN}_k(A)$ be the Petri net generated from all k-bounded regions of A.

[1] We exclude infinite LTS since they lead to unbounded Petri nets, for which the synthesis from modal transition systems is known to be undecidable [14].

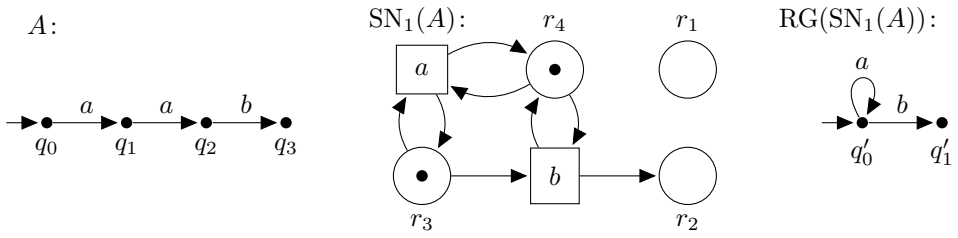

Figure 2 Example for $\mathrm{SN}_k(A)$ and the resulting reachability graph (on the right).

An example for this construction with $k = 1$ is shown in Figure 2. The LTS A has only four regions r_1 to r_4. We identify a region r with the vector $(r(q_0), r(q_1), r(q_2), r(q_3))$:

$$r_1 = (0,0,0,0) \qquad r_2 = (0,0,0,1) \qquad r_3 = (1,1,1,0) \qquad r_4 = (1,1,1,1)$$

We also identify the gradient Δ_r and minimum values μ_r of r with the vector $(\Delta_r(a), \Delta_r(b))$, respectively $(\mu_r(a), \mu_r(b))$:

$$\begin{array}{llll} \Delta_{r_1} = (0,0) & \Delta_{r_2} = (0,1) & \Delta_{r_3} = (0,-1) & \Delta_{r_4} = (0,0) \\ \mu_{r_1} = (0,0) & \mu_{r_2} = (0,0) & \mu_{r_3} = (1,1) & \mu_{r_4} = (1,1) \end{array}$$

The event a always has a gradient of $\Delta_r(a) = 0$, since otherwise at least one of the values $r(q_0)$, $r(q_1)$, or $r(q_2)$ would need to be outside of $\{0,1\}$. For the event b, we can have $\Delta_r(b) \in \{-1,0,1\}$. Altogether, A has four regions which correspond to the places of the Petri net shown in the middle of Figure 2. As can be seen in the reachability graph, the event a forms a loop around the initial state.

▶ **Lemma 10.** *For any $k \in \mathbb{N}^+$ and finite LTS A, the Petri net $\mathrm{SN}_k(A)$ is k-bounded.*

Proof. Assume a reachable marking M where some place p contains more than k tokens, i.e. $M(p) > k$. We use the well-known idea of complement places: With r being the region corresponding to p, define a new region r' via $r'(q) = k - r(q)$. Clearly, r' is also a region of A and corresponds, by definition, to some place p' of $\mathrm{SN}_k(A)$, which is built from all regions of A. Also, for every event $a \in \Sigma_A$ we have $\Delta_r(a) + \Delta_{r'}(a) = 0$ and for every state q we have $r(q) + r'(q) = k$. This also holds for the corresponding places: for any reachable marking M', $M'(p) + M'(p') = k$. Since $M(p) > k$ we find $M(p') < 0$. This contradicts M being a (reachable) marking. ◀

Next, we want to show that $\mathrm{SN}_k(A)$ is the smallest Petri net over-approximation of A. This means that any other Petri net which is an over-approximation of A will also be an over-approximation of the reachability graph of $\mathrm{SN}_k(A)$.

▶ **Theorem 11.** *Given a number $k \in \mathbb{N}^+$ and a finite LTS A, we have that $A \sqsubseteq \mathrm{RG}(\mathrm{SN}_k(A))$ and for all k-bounded Petri nets N with $A \sqsubseteq \mathrm{RG}(N)$, also $\mathrm{RG}(\mathrm{SN}_k(A)) \sqsubseteq \mathrm{RG}(N)$.*

Proof. It is clear that $A \sqsubseteq \mathrm{RG}(\mathrm{SN}_k(A))$, because $\mathrm{SN}_k(A)$ is defined in such a way that it does not prevent edges present in A ($q \xrightarrow{t} \Rightarrow r(q) \geq \mu_r(t) = F(p,t)$).

Then, for any N with $A \sqsubseteq \mathrm{RG}(N)$ via f, each place of N corresponds to a region r on $\mathrm{RG}(N)$. This region can be translated into a region $r \circ f$ of A via Lemma 8, which by

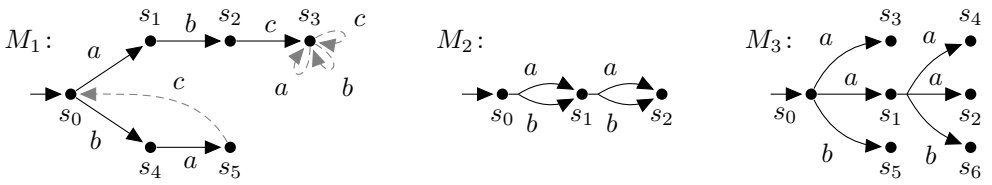

Figure 3 Three different hMTS. Initial states are marked with an arrow tip. Must edges are drawn as solid lines and also indicate may edges as required by the definition. For example, $(s_0, \{(a, s_1), (b, s_1)\}) \in \longrightarrow$ for M_2. Non-must may edges are shown with dashed lines. M_1 is a deterministic MTS, M_2 a deterministic hMTS, M_3 a non-deterministic hMTS.

definition corresponds to a place of $\mathrm{SN}_k(A)$, yielding a mapping of places[2] $\alpha\colon P_N \to P_{\mathrm{SN}_k(A)}$. We can now define a function β that maps markings of $\mathrm{SN}_k(A)$ to markings of N via $\beta(M)(p) = M(\alpha(p))$, i.e. a place p of N is assigned the same number of tokens as its corresponding place $\alpha(p)$ of $\mathrm{SN}_k(A)$ has in M. We can easily verify $M[t\rangle M' \Rightarrow \beta(M)[t\rangle\beta(M')$, i.e. we have $\mathrm{RG}(\mathrm{SN}_k(A)) \sqsubseteq \mathrm{RG}(N)$ via β. ◀

3 Modal Transition Systems

Hyper modal transition systems [18] represent classes of LTS. They allow for multiple initial states and have two different classes of edges. May edges allow an action while must edges require an action. Compared to classical modal transition systems [17], must edges are hyper edges that require at least one out of several possible actions, while in a standard modal transition system no such logical combinations are expressible. A hyper modal transition system is similar to an acceptance specification [19], but its must edges can be considered to be in conjunctive normal form while an acceptance specification uses the disjunctive normal form.

▶ **Definition 12.** A *hyper modal transition system (hMTS)* M is a structure $M = (S, \Sigma, \longrightarrow, \dashrightarrow, S_0)$ where S is a finite set of *states*, Σ is a set of *actions*, $\emptyset \neq S_0 \subseteq S$ is the set of *initial states*, $\dashrightarrow \subseteq S \times \Sigma \times S$ is the set of *may edges* and $\longrightarrow \subseteq S \times (2^{\Sigma \times S} \setminus \emptyset)$ is the set of *must edges* satisfying $\forall (s, D) \in \longrightarrow\colon \forall (a, s') \in D\colon (s, a, s') \in \dashrightarrow$. The syntax $s \overset{w}{\dashrightarrow} s'$ is defined just like $q \overset{w}{\longrightarrow} q'$ was defined for LTS. An hMTS is called *deterministic* iff $S_0 = \{s_0\}$ is a singleton and the LTS $(S, \Sigma, \dashrightarrow, s_0)$ is deterministic. A *modal transition system (MTS)* is an hMTS with $|S_0| = 1$ and $\forall (s, D) \in \longrightarrow\colon |D| = 1$.

As an example, three hMTS are shown in Figure 3. In M_1, the lower path forms a *bac*-loop that can be terminated after an a, the upper path requires *abc*, after which anything is allowed. M_2 contains two hyper edges requiring at least either an a or a b, but not necessarily both. In an LTS meeting the specification of M_3, the three must edges at s_0 demand that both a and b must be available actions, while the hyper edge at s_1 only requires that one of a and b is possible.

The following definition introduces how an LTS implements an hMTS. This implementation relation is similar to a bisimulation, but may and must edges are not treated the same.

[2] A mapped place $\alpha(p)$ has the same initial marking and transitions of $\mathrm{SN}_k(A)$ have the same effect $\Delta_r = \Delta_{r \circ f}$ on it as on p, but self-loops might be added (if $\mu_{r \circ f}(t) > \mu_r(t)$ due to edges not existing in $\mathrm{RG}(\mathrm{SN}_k(A))$ but in $\mathrm{RG}(N)$).

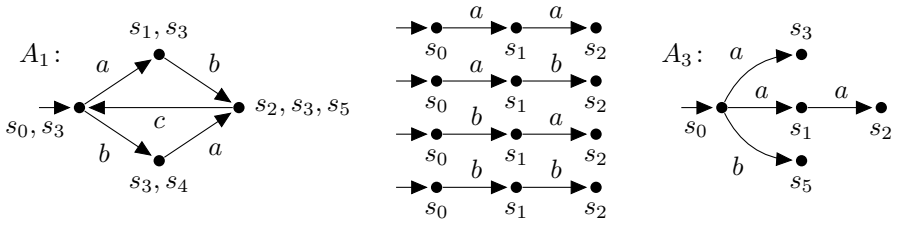

Figure 4 Implementations of the hMTS from Figure 3: A_1 implements M_1, A_3 implements M_3, and the four LTS in the middle are minimal implementations of M_2. A_1 is solved by N_1 from Figure 1. States are named according to the implementation relation R from Definition 13.

A must edge specifies a disjunction of edges, at least one of which must be present, while a may edge defines that some edge is allowed.

▶ **Definition 13.** An LTS A is an *implementation via* $R \subseteq Q_A \times S_M$ of an hMTS M if there is an $s_0 \in S_{0,M}$ with $(q_{0,A}, s_0) \in R$ and for all $(q, s) \in R$:

- $\forall q \xrightarrow{a} q' \colon \exists s \overset{a}{\dashrightarrow} s' \colon (q', s') \in R$,
- $\forall (s, D) \in \longrightarrow \colon \exists (a, s') \in D, q \xrightarrow{a} q' \colon (q', s') \in R$.

If A is an implementation of M via some relation R, we call R an *implementation relation* and write $A \models M$ (via R). If RG(N) is an implementation of M for some Petri net N, we also say that N *realises* M.

Examples for this definition are given in Figure 4. The LTS A_1 is an implementation of M_1 from Figure 3. An implementation relation R can be built as follows. Initially, $(\{s_0, s_3\}, s_0) \in R$. The lower path of M_1 adds first $(\{s_3, s_4\}, s_4)$ and then $(\{s_2, s_3, s_5\}, s_5)$ to R. The must edges of the upper path require $(\{s_1, s_3\}, s_1)$, $(\{s_2, s_3, s_5\}, s_2)$, and $(\{s_0, s_3\}, s_3)$. In the state s_3, all edges are allowed and loop back to s_3, i.e. all states reachable from $\{s_0, s_3\}$ must also be related to s_3: $(\{s_1, s_3\}, s_3), (\{s_3, s_4\}, s_3), (\{s_2, s_3, s_5\}, s_3) \in R$. Thus, the state names of the LTS A_1 show exactly the implementation relation for $A_1 \models M_1$. If we remove in M_1 the may edge labelled with c at s_5, A_1 would not be an implementation any more: $\{s_2, s_3, s_5\}$ must admit a c (forced by the upper path in M_1) and forbid a c (using the lower path) at the same time. We could split up the state $\{s_2, s_3, s_5\}$ into $\{s_2, s_3\}$ and $\{s_5\}$, but then no Petri net could solve the LTS, since the paths ab and ba from the initial state must lead to the same marking[3].

The four LTS in the middle of Figure 4 are all valid implementations of M_2, and they are all minimal according to the LTS homomorphism $\sqsubseteq$, which is a partial order for deterministic and totally reachable LTS according to Lemma 3. There are larger implementations admitting a and b at either s_0 or s_1 (or both).

The LTS A_3 is a minimal implementation of M_3. The states s_1, s_3, and s_5 are forced by the implementation relation; the hyper edge at s_1 allows a choice: at least one of the targets of the hyper edge must exist. Here, we chose s_2. Note that the states s_1 and s_3 cannot be identified in spite of the non-deterministic a-edges with the same source s_0. The implementation relation would contain a pair $(\{s_1, s_3\}, s_3)$ which would in consequence – due to the edge $\{s_1, s_3\} \xrightarrow{a} s_2$ – require s_3 to have at least a may edge in M_3. Since reachability graphs of Petri nets are deterministic, no LTS implementing M_3 can be solved by a Petri

[3] This is due to the well-known Petri net state equation $M[w\rangle M' \Rightarrow M' = M + C \cdot \Psi(w)$ where $C \in \mathbb{Z}^{P \times N}$ with $C(p,t) = F(t,p) - F(p,t)$. This equation only considers the number of times $\Psi(w)(t)$ that event $t \in T$ occurs in the word $w \in T^*$, and not the order of appearances.

net. However, if we omit the edge $(s_0, \{(a, s_3)\}) \in \longrightarrow$ in M_3, Petri net realisations become possible. There are (up to isomorphism) nine different deterministic LTS implementing the modified M_3, five of which can be solved by Petri nets. They differ in whether a, b, or both are possible in s_1, and in the number of states without outgoing edges.

4 Goal-oriented Synthesis

In this section, we will introduce the algorithm that calculates Petri net realisations for hyper modal transition systems. The algorithm works by iteratively adding unimplemented must edges to an LTS and doing minimal Petri net over-approximations. For identifying unimplemented must edges, another kind of relation is needed, which we will define next.

▶ **Definition 14.** Let A be an LTS and M be an hMTS. A relation $R \subseteq Q_A \times S_M$ is called an *expansion relation* if there is a state $s_0 \in S_{0,M}$ with $(q_{0,A}, s_0) \in R$ and for all $(q, s) \in R$ and all $q \xrightarrow{a} q'$, there is an s' with $s \xrightarrow{a} s'$ and $(q', s') \in R$.

Let $R_{\mathrm{E}}(A, M)$ be the set of all expansion relations of A and M. Given LTS A and B, an expansion relation $R \in R_{\mathrm{E}}(A, M)$, and an LTS homomorphism f witnessing $A \sqsubseteq B$, the relation $f(R)$ is defined as $f(R) = \{(f(q), s) \in Q_B \times S_M \mid (q, s) \in R\}$.

Expansion relations are a weaker variant of implementation relations, i.e. any implementation relation is an expansion relation with an additional condition for the must edges.

Our algorithm for realising an hMTS is Algorithm 1. The idea behind the algorithm is to iteratively enlarge a current LTS towards becoming an implementation. The first operation for this is to add edges to the LTS so that currently unimplemented must edges become implemented (Expand). The second operation is the minimal Petri net over-approximation (PNApprox). The algorithm begins in the procedure RealiseMTS, which gets as input the bound $k \in \mathbb{N}^+$ for the Petri nets and the hMTS M to be realised. The first LTS to be considered is the minimal LTS which only has an initial state and no edges. This state is related to each possible initial state of M and the procedure Recurse is called for each such combination.

Recurse identifies unimplemented must edges by negating the corresponding formula in the definition of an implementation relation. If no must edge is missing, a solution is found and returned. Otherwise, the procedure Expand is invoked which will add new states and edges to implement the missing must edges. This procedure returns a set, because, for example, for $m = \{(q, \{(a, s_1), (b, s_2)\})\}$, two LTS are created: In one of them, an edge $q \xrightarrow{a} q_{new}$ is created, while the other one gets the new edge $q \xrightarrow{b} q_{new}$. In the first case, (q_{new}, s_1) is added to the expansion relation R_A while in the second case (q_{new}, s_2) is added. This means the relation R_A is enlarged to keep note on which state of the hMTS the new state should implement. Expand is invoked recursively to obtain all combinations of selections for the missing must edges. In MTS must edges allow no choice, therefore Expand will return one unique result; in hMTS a high number of choices in must edges could lead to an exponential growth, though.

For each result of Expand, the procedure PNApprox is called, yielding a minimal Petri net over-approximation of its argument. As was seen in Figure 2, this can greatly influence the shape of the LTS under consideration. The relation R_A has to be updated to this new LTS. To do so, PNApprox computes a homomorphism witnessing $A \sqsubseteq B$ in line 23. Such a homomorphism exists by Theorem 11. It is unique by Lemma 2, because A is totally reachable by construction and B is deterministic by Lemma 5. This homomorphism f is then used to produce the relation $f(R_A)$ which relates B to M. The set E of all expansion relations

Algorithm 1 Algorithm for finding Petri net realisations for an hMTS.

procedure REALISEMTS(k, M) ▷ $k \in \mathbb{N}^+$ and M is an hMTS
  Let A be the LTS consisting of just an initial state q_0
  return $\bigcup_{s_0 \in S_{0,M}}$ RECURSE($k, A, \{(q_0, s_0)\}, M$)
end procedure
procedure RECURSE(k, A, R_A, M) ▷ R_A is an expansion relation
  $m = \bigcup_{(q,s) \in R_A} m_{q,s}$ where ▷ m collects missing must edges
  $m_{q,s} = \{(q, D) \in Q_A \times (2^{\Sigma \times S_M} \setminus \emptyset) \mid (s, D) \in \longrightarrow_M, \forall (a, s') \in D:$
    $\neg \exists q' \in Q_A : (q \xrightarrow{a} q' \wedge (q', s') \in R_A)\}$
  if $m = \emptyset$ **then return** $\{(A, R_A)\}$ **end if**
  return $\bigcup_{(B,R_B) \in \text{EXPAND}(A,R_A,m)} \bigcup_{(C,R_C) \in \text{PNAPPROX}(k,B,R_B,M)}$ RECURSE(k, C, R_C, M)
end procedure
procedure EXPAND(A, R_A, m)
  if $m = \emptyset$ **then return** $\{(A, R_A)\}$ **end if**
  Select some $(q, D) \in m$, add a new state q_{new} to A and set result $\leftarrow \emptyset$
  for $(a, s') \in D$ **do** ▷ Implement missing must edge
    Let B be a copy of A with an additional edge $q \xrightarrow{a} q_{new}$
    result $\leftarrow$ result $\cup$ EXPAND($B, R_A \cup \{(q_{new}, s')\}, m \setminus \{(q, D)\}$)
  end for
  return result
end procedure
procedure PNAPPROX(k, A, R_A, M)
  $B = \text{RG}(\text{SN}_k(A))$ ▷ Minimal over-approximation
  $f =$ homomorphism witnessing $A \sqsubseteq B$
  $E = \{R_B \in R_E(B, M) \mid f(R_A) \subseteq R_B\}$ ▷ All expansion relations containing $f(R_A)$
  return $\{(B, R_B) \mid R_B \in E \wedge \neg \exists R' \in E : R' \subsetneq R_B\}$ ▷ Select minimal relations
end procedure

containing $f(R_A)$ is computed and the minimal relations are returned. It is possible that this produces several different minimal relations, or none at all. In the latter case, PNAPPROX returns the empty set and this branch of the algorithm fails to find any realisation of the hMTS M.

4.1 Examples

As our first example, we will use M_1 from Figure 3 and realise it with a 1-bounded Petri net.

The call REALISEMTS$(1, M_1)$ begins by creating an LTS A consisting of just an initial state q_0 which gets related to the initial state s_0 of M_1. In RECURSE$(1, A, \{(q_0, s_0)\}, M_1)$, the set $m = \{(q_0, \{(a, s_1)\}), (q_0, \{(b, s_4)\})\}$ is calculated. Therefore, q_0 needs an outgoing edge with label a and another edge with label b. The invocation EXPAND$(A, \{(q_0, s_0)\}, m)$ produces just the pair (A_1, R_1) where A_1 is shown in Figure 5. The same figure also visualises R_1 by labelling a state q in the LTS with s if $(q, s) \in R_1$. The following call to PNAPPROX does not modify this LTS and RECURSE calls itself recursively with (A_1, R_1).

The next call to EXPAND produces the LTS $\widetilde{A_1}$ from Figure 5. This LTS cannot be solved by a Petri net, because if both sequences ab and ba are enabled in a marking, they must lead to the same state. Thus, PNAPPROX produces the LTS A_2. The relation that is indicated by its labelling is $f(\widetilde{R_1})$, i.e. no additional entries need to be added to produce an expansion relation.

This expansion procedure continues for some iterations until the LTS $\widetilde{A_5}$ from Figure 5

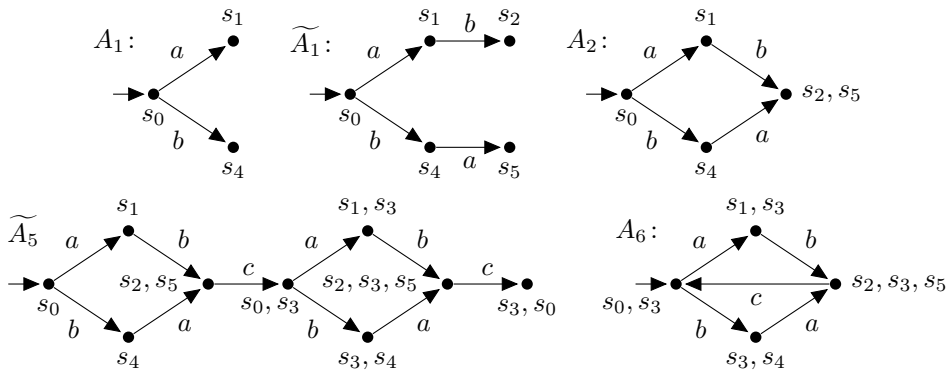

Figure 5 Intermediate LTS from the Petri net synthesis of the deterministic MTS M_1 from Figure 3. Some intermediate results are shown. To visualise a relation R, the states of an LTS are labelled with the states of M_1 that they have to implement.

is created by Expand. This LTS cannot be solved by a 1-bounded Petri net. As we have already seen in Figure 2, when some firing sequence $ww \in \Sigma^*$ is possible in a 1-bounded Petri net, then it has twice the effect of the sequence w on the marking, i.e. it cannot have an effect at all. Thus, it forms a loop in the reachability graph. The same thing happens now with the sequence abc. The minimal over-approximation produces the LTS A_6. The relation returned by PNApprox is just $R_6 = f(\widetilde{R_5})$. At this point all must edges are implemented and we have found a realisation.

As another example, consider RealiseMTS$(2, M_2)$ where M_2 is shown in Figure 3. This produces the four LTS that are displayed in the middle of Figure 4.

If one considers RealiseMTS$(1, M_2)$ instead, i.e. $k = 1$ instead of $k = 2$, then only the LTS corresponding to the words ab and ba are generated. This is because, for example, aa cannot be generated by one-bounded Petri nets without also allowing aaa to occur. This extra behaviour is not allowed by M_2 and so the algorithm will end up with $E = \emptyset$ in line 24 and this aa-branch of the recursion does not find any realisations.

When applying the algorithm to M_3 of Figure 3, we get an LTS with states corresponding to s_0, s_1, s_3, and s_5 in the first expansion phase. The over-approximation will then combine s_1 and s_3 to $\{s_1, s_3\}$ since both a-edges must have the same effect on the marking of the Petri net. In the second expansion phase, one of the edges to s_4, s_2, or s_6 is added to the LTS, yielding three LTS to over-approximate. While the over-approximations exist, no expansion relation can be built for them. For the pair $(\{s_1, s_3\}, s_3)$, the LTS state $\{s_1, s_3\}$ has an edge now (the newly added a or b), but s_3 in M_3 has no may edge at all. Thus, $E = \emptyset$ and PNApprox returns the empty set in all cases. The realisation of M_3 by a Petri net has failed (correctly).

5 Termination and Correctness

In this section, three results will be shown: The algorithm terminates; the algorithm is correct in the sense that it really produces realisations of the hMTS M; and the algorithm is complete in the sense that it always finds a realisation if one exists.

▶ **Lemma 15.** *For an hMTS M and a $k \in \mathbb{N}^+$, a call to* RealiseMTS(k, M) *terminates.*

Proof. If the algorithm does not terminate, it must be due to Recurse because PNApprox contains only finite constructions and Expand has a recursion depth of $|m|$. Since the unions in line 10 of the algorithm are always finite, there must be an infinite recursion of Recurse with arguments $(A_i)_{i\in\mathbb{N}}$. All arguments to Recurse are Petri net solvable by construction. Also, by Lemma 6, there are only finitely many LTS (abstracting state names) over a fixed alphabet which are k-bounded Petri net solvable. Thus, the A_i are all elements of a finite set and form an infinite chain with $A_i \sqsubseteq A_{i+1}$ for $i \in \mathbb{N}$, i.e. some LTS A (up to isomorphism) will occur infinitely often. Let A_j and $A_{j'}$ be any two instances of A with $j < j'$, so $A_j = A_{j'}$. For $j < n < j'$ then $A_j \sqsubseteq A_n \sqsubseteq A_{j'}$ and Lemma 3 can be applied since all A_i are reachability graphs and thus by Lemma 5 totally reachable and deterministic. We conclude $A_j = A_n$ (up to isomorphism), i.e. the infinite chain $(A_i)_{i\in\mathbb{N}}$ becomes stationary.

Consider now some $i \geq j$. Let $\widetilde{A_i}$ be the LTS resulting from applying Expand to A_i, possibly adding one (or more) states q_{new} to A_i. Then, $A_i \sqsubseteq \widetilde{A_i}$ via the identity homomorphism id, where the new state(s) q_{new} do not occur as an image. Now, PNApprox recreates the original LTS, i.e. $\widetilde{A_i} \sqsubseteq A_{i+1} = A_i$ via some homomorphism f. Overall, $A_i \sqsubseteq \widetilde{A_i} \sqsubseteq A_i$ via $f \circ id$, which is uniquely determined according to Lemma 2. Thus, $f \circ id$ must be the identity mapping, and f also must be the identity on all states except the new one(s), q_{new}.

If Expand would add an edge $q \xrightarrow{a} q_{new}$ to A_i, then $f(q) \xrightarrow{a} f(q_{new})$ would also be an edge in $A_i = A_{i+1}$. PNApprox maps $f(q) = q$ and $f(q_{new}) = q' \in Q_{A_i}$. The new element of the expansion relation added by Expand, $(q_{new}, \tilde{s})$ where $\tilde{s}$ is determined by M, is mapped to $(q', \tilde{s}) \in R_{A_{i+1}}$. Note that $(q', \tilde{s}) \notin R_{A_i}$, otherwise Expand would not have added an edge in the first place. Therefore, $|R_{A_i}| < |R_{A_{i+1}}|$. Since the chain $(A_i)_{i\geq j}$ is stationary, the maximal size of an expansion relation is $|Q_{A_j} \times S_M|$, i.e. at some point in the chain, Expand cannot add any further edges. Therefore, the to-be-implemented set m will be empty and Recurse terminates. ◀

The following lemma shows that RealiseMTS only produces realisations of its input:

▶ **Lemma 16.** *For an hMTS M and a $k \in \mathbb{N}^+$, every $(A, R_A) \in$ RealiseMTS(k, M) satisfies $A \models M$ via R_A and A can be solved by a k-bounded Petri net.*

Proof. For (A, R_A) to appear in the result of RealiseMTS, it must satisfy $m = \emptyset$ in line 9 of the algorithm. This means that there are no unimplemented must edges, which shows the first half of $A \models M$ via R_A. The other half, that all edges in A are allowed by may-edges, holds, because every value of R_A in the algorithm is an expansion relation: Initially, A only has an initial state and R_A relates this state to an initial state of M. When Expand adds new edges to an LTS, the relation R_A is updated accordingly. Here, (s, a, s') is a may edge, because every must edge must also be a may edge by definition of an hMTS. Finally, PNApprox explicitly only returns expansion relations.

It remains to be shown that A can be solved by a k-bounded Petri net. This is the case, because every LTS A given to Recurse is either the trivial LTS having no edges or was generated by PNApprox as a minimal Petri net over-approximation. ◀

In the remainder of the section, we will show that RealiseMTS$(k, M) = \emptyset$ can only occur if there are no realisations of M. For this we need the following preorder on LTS enriched with a relation:

▶ **Definition 17.** Let S be an arbitrary set. For two LTS A and B assume relations $R_A \subseteq Q_A \times S$ and $R_B \subseteq Q_B \times S$. We write $(A, R_A) \sqsubseteq (B, R_B)$ if there is an LTS

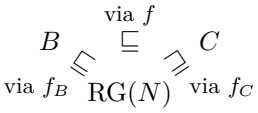

Figure 6 Illustration for part (2) of the proof of Lemma 18.

homomorphism f witnessing $A \sqsubseteq B$ so that $\forall (q,s) \in R_A : (f(q),s) \in R_B$, i.e. $f(R_A) \subseteq R_B$ (cf. Definition 14).

▶ **Lemma 18.** *Let M be an hMTS, $k \in \mathbb{N}^+$, and N be a Petri net with $\mathrm{RG}(N) \models M$ via some relation R_N. There is an $(A, R_A) \in \textsc{RealiseMTS}(k, M)$ so that $(A, R_A) \sqsubseteq (\mathrm{RG}(N), R_N)$.*

Proof. We proof inductively that there is always an (A, R_A) in the current state of the algorithm that satisfies $(A, R_A) \sqsubseteq (\mathrm{RG}(N), R_N)$.

By $\mathrm{RG}(N) \models M$ via R_N, there must be an $s_0 \in S_{0,M}$ with $(M_0, s_0) \in R_N$ (M_0 being the initial marking of N). Since RealiseMTS tries all initial states of M, the same s_0 will be picked at some point, i.e. $\textsc{Recurse}(k, A, R_A, M)$ will be called with A being the LTS having just a single state and the relation $R_A = \{(q_0, s_0)\}$. Here, $(A, R_A) \sqsubseteq (\mathrm{RG}(N), R_N)$ holds.

Next, assume that Recurse is called with arguments A and R_A satisfying $(A, R_A) \sqsubseteq (\mathrm{RG}(N), R_N)$ via some homomorphism f. We have to show two things: (1) One of the pairs (B, R_B) generated by $\textsc{Expand}(A, R_A, m)$ satisfies $(B, R_B) \sqsubseteq (\mathrm{RG}(N), R_N)$ and (2) the same applies to one of the pairs (C, R_C) which are generated by the following call of $\textsc{PNApprox}(k, B, R_B, M)$ in line 10. Since the algorithm always terminates by Lemma 15, eventually $m = \emptyset$ in line 9 will cause a suitable (A, R_A) to be returned. All recursive calls to Recurse pass this on and in the end the pair (A, R_A) is returned by RealiseMTS, completing the proof.

For (1) consider any unimplemented $(q, D) \in m$. By construction of m there must be a $(q, s) \in R_A$ so that $(q, D) \in m_{q,s}$ and $(s, D) \in \longrightarrow_M$. By $(A, R_A) \sqsubseteq (\mathrm{RG}(N), R_N)$ via f, we have $(f(q), s) \in R_N$. By $(s, D) \in \longrightarrow_M$, $(f(q), s) \in R_N$ and $\mathrm{RG}(N) \models M$ via R_N, there must be some $(a, s') \in D$ and $\widetilde{M}$ with $f(q) \xrightarrow{a} \widetilde{M}$ and $(\widetilde{M}, s') \in R_N$. Thus, on our way to B we can pick that LTS generated in the iteration of line 15 of the algorithm which handles the same $(a, s') \in D$. We define a new function f' that is identical to f, except that its domain is extended by q_{new} via $f'(q_{new}) = \widetilde{M}$. Having done this for all unimplemented must edges, the final function f' witnesses $(B, R_B) \sqsubseteq (\mathrm{RG}(N), R_N)$: Most of this property is inherited from f and for any new edge $q \xrightarrow{a} q_{new}$, we constructed f' so that $f'(q) \xrightarrow{a} f'(q_{new})$ is satisfied.

For (2) we assume that we have $(B, R_B) \sqsubseteq (\mathrm{RG}(N), R_N)$ via some function f_B. We have to find some $(C, R_C) \in \textsc{PNApprox}(k, B, R_B, M)$ with $(C, R_C) \sqsubseteq (\mathrm{RG}(N), R_N)$. The LTS relations are illustrated in Figure 6. From the algorithm we know $C = \mathrm{RG}(\mathrm{SN}_k(B))$. From the assumption we get $B \sqsubseteq \mathrm{RG}(N)$ via f_B. Applying Theorem 11 to B, we can deduce $C = \mathrm{RG}(\mathrm{SN}_k(B)) \sqsubseteq \mathrm{RG}(N)$ via some function f_C and $B \sqsubseteq C$ via some function f. We have $B \sqsubseteq C \sqsubseteq \mathrm{RG}(N)$ via $f_C \circ f$ by transitivity of $\sqsubseteq$. Since B is totally reachable by construction and $\mathrm{RG}(N)$ is deterministic by Lemma 5, we can apply Lemma 2 stating that the homomorphism witnessing $B \sqsubseteq \mathrm{RG}(N)$ is unique. We conclude $f_B = f_C \circ f$.

Consider the expansion relation R'_C defined by $R'_C = \{(q, s) \in Q_C \times S_M \mid (f_C(q), s) \in R_N\}$. This relation inherits the expansion property from R_N, i.e. if $(q, s) \in R'_C$ and $(q, a, q') \in \rightarrow_C$, then $(f_C(q), s) \in R_N$ and $(f_C(q), a, f_C(q')) \in \rightarrow_{\mathrm{RG}(N)}$, which means that there is a suitable $s' \in S_M$ with $s \overset{a}{\dashrightarrow} s'$ and $(f_C(q'), s') \in R_N$, so $(q', s') \in R'_C$. By definition we have $(C, R'_C) \sqsubseteq (\mathrm{RG}(N), R_N)$ via f_C. Consider any element $(f(q), s) \in f(R_B)$. By definition of $f(R_B)$, we have $(q, s) \in R_B$. By $(B, R_B) \sqsubseteq (\mathrm{RG}(N), R_N)$ via f_B, we have

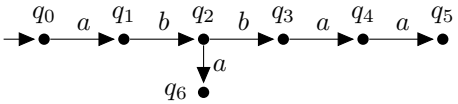

Figure 7 An LTS that can be solved exactly, but only if non-minimal regions are considered.

$(f_B(q), s) \in R_N$. By $f_B = f_C \circ f$, we have $(f_C(f(q)), s) \in R_N$. By definition of R'_C, we have $(f(q), s) \in R'_C$. Thus, $f(R_B) \subseteq R'_C$, which means that R'_C is an element of the set E inside the algorithm, i.e. there is a relation $R_C \in E$ with $f(R_B) \subseteq R_C \subseteq R'_C$ so that $(C, R_C) \in \text{PNAPPROX}(k, B, R_B, M)$. Since $(C, R'_C) \sqsubseteq (\text{RG}(N), R_N)$ via f_C and $R_C \subseteq R'_C$, we also have $(C, R_C) \sqsubseteq (\text{RG}(N), R_N)$ by the definition of $\sqsubseteq$, which needed to be shown. ◀

▶ **Corollary 19.** *For an hMTS M and a $k \in \mathbb{N}^+$, if* $\text{REALISEMTS}(k, M) = \emptyset$ *then there is no k-bounded Petri net N with* $\text{RG}(N) \models M$.

6 Some final thoughts

An aspect of the algorithm that is still open is its complexity. This depends a lot on the complexity of the Petri net over-approximation. The question of whether for a given LTS A any 1-bounded and pure Petri net N exists with $A = \text{RG}(N)$ is NP-complete [4]. The same question can be answered in polynomial time when asking for bounded Petri nets, but without picking a bound $k \in \mathbb{N}^+$ *a priori* [3]. We do not know about any complexity results for the synthesis of k-bounded Petri nets with a given k, nor for the k-bounded over-approximation that is needed in our present setting.

When looking at some algorithms for this problem, we can see that the brute force computation of all regions will certainly be exponential (checking $(k+1)^{|Q|}$ functions) in the size of the LTS, but not all regions might be necessary. Attempts to reduce this problem have been made, e.g. in [10, 11], but this procedure does not guarantee success. It only computes minimal regions, where a region r is *minimal* if there is no other region $r' \neq \mathbf{0}$ with $r' \leq r$ (pointwise). Figure 7 contains an LTS which cannot be solved when just using minimal regions. The following list contains all six minimal regions of this LTS, where a region r is identified with the vector $(r(q_0), r(q_1), \ldots, r(q_6))$ and a gradient Δ_r with the vector $(\Delta_r(a), \Delta_r(b))$:

$$r_1 = (1,0,1,2,1,0,0) \quad \Delta_{r_1} = (-1,1) \quad r_2 = (2,2,1,0,0,0,1) \quad \Delta_{r_2} = (0,-1)$$
$$r_3 = (1,2,1,0,1,2,2) \quad \Delta_{r_3} = (1,-1) \quad r_4 = (0,2,1,0,2,4,3) \quad \Delta_{r_4} = (2,-1)$$
$$r_5 = (0,1,1,1,2,3,2) \quad \Delta_{r_5} = (1,0) \quad r_6 = (0,0,1,2,2,2,1) \quad \Delta_{r_6} = (0,1)$$

It can easily be verified that for all of these regions we have $r(q_6) \geq \mu_r(b) = F(r,b)$, which means that none of these regions prevents transition b in state q_6. However, the region $r_7 = (3,2,2,2,1,0,1)$ does prevent b in q_6 ($r_7(q_6) = 1 < 2 = \mu_{r_7}(b)$). This region is not minimal since $r_1 \leq r_7$, but only with this region can the LTS from Figure 7 be solved.

In the general setting of bounded Petri net synthesis without k given a priori, there are polynomial algorithms based on solving so-called *separation problems* (see e.g. [3, 5]). These algorithms seem to be quite efficient in our experiments, but so far there do not seem to be variants where k is given a priori nor for computing a minimal over-approximation. If further research finds such an algorithm, another optimisation becomes possible: When a must edge is present, the corresponding event does not need to be prevented even if this is

possible. Thus, instead of computing the minimal over-approximation, the Expand-step could be integrated with the synthesis to make it faster.

Coming back to the problem of finding a Petri net realisation of an MTS, the brute force approach would be the construction of all k-bounded Petri nets with a fixed set of transitions. This also requires the computation of all possible regions (places), but afterwards we have to walk through the power set of the places, construct the according reachability graphs, and check if they implement our MTS. As long as the Petri net over-approximation is not used too often, our algorithm will probably be faster.

Still, Expand and PNApprox are independent procedures, so it might be feasible to reduce the number of over-approximations in favour of the Expand step. We can modify our algorithm, doing as many consecutive Expand steps as possible (e.g. until a cycle in the MTS becomes fully implemented) and only then applying one over-approximation.

Of course, with general hMTS, the Expand step can itself create exponentially many LTS, but we might expect that true hyper edges (defining algorithmic points of choice) occur with low frequency in at least human-made hMTS. For MTS (without true hyper edges), every Expand step only computes a single LTS. If an hMTS is deterministic, it can easily be shown that there is at most one expansion relation and so PNApprox would also compute at most a single result. Thus, the algorithm does not branch when given a deterministic MTS.

Overall, we obtain a goal-oriented algorithm and thus a decision procedure for the realisability of hMTS by k-bounded Petri nets with hopefully much lower run times than the brute force approach by enumeration of all candidate nets. We are already close to the barrier of undecidability here, since for bounded Petri nets (without the fixed $k \in \mathbb{N}^+$), the realisability of even MTS is known to be undecidable [20].

Our approach can easily handle restricted Petri net classes such as plain Petri nets (arc weights at most 1) or pure Petri nets (see Definition 4). Instead of constructing $\mathrm{SN}_k(A)$ from all k-bounded regions, simply only regions corresponding to plain, respectively pure, places are considered.

Further research effort can be put into investigating more expressive specifications than hMTS. We would like to have a similar algorithm for finding Petri net realisations of formulas of the modal μ-calculus [15, 2]. The general approach of the algorithm would stay the same, but a suitable replacement for the Expand step is needed.

Acknowledgements. We would like to thank Eike Best and Valentin Spreckels for their help. Also, we are thankful to the anonymous reviewers for their helpful comments.

References

1 Adam Antonik, Michael Huth, Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski. 20 years of modal and mixed specifications. *Bulletin of the EATCS*, 95:94–129, 2008.

2 André Arnold and Damian Niwiński. *Rudiments of μ-calculus.* North Holland, 2001.

3 Eric Badouel, Luca Bernardinello, and Philippe Darondeau. Polynomial algorithms for the synthesis of bounded nets. In Peter D. Mosses, Mogens Nielsen, and Michael I. Schwartzbach, editors, *TAPSOFT'95*, volume 915 of *LNCS*, pages 364–378. Springer, 1995. doi:10.1007/3-540-59293-8_207.

4 Eric Badouel, Luca Bernardinello, and Philippe Darondeau. The synthesis problem for elementary net systems is NP-complete. *Theoretical Computer Science*, 186(1-2):107–134, 1997. doi:10.1016/S0304-3975(96)00219-8.

5 Eric Badouel, Luca Bernardinello, and Philippe Darondeau. *Petri Net Synthesis*. Texts in Theoretical Computer Science. Springer, 2015. doi:10.1007/978-3-662-47967-4.

6 Eric Badouel, Benoît Caillaud, and Philippe Darondeau. Distributing finite automata through Petri net synthesis. *Formal Aspects of Computing*, 13(6):447–470, 2002. doi:10.1007/s001650200022.

7 Eric Badouel and Philippe Darondeau. Theory of regions. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *LNCS*, pages 529–586. Springer, 1996. doi:10.1007/3-540-65306-6_22.

8 Eike Best and Philippe Darondeau. Petri net distributability. In Edmund M. Clarke, Irina Virbitskaite, and Andrei Voronkov, editors, *PSI 2011*, volume 7162 of *LNCS*, pages 1–18. Springer, 2011. doi:10.1007/978-3-642-29709-0_1.

9 Glenn Bruns. An industrial application of modal process logic. *Science of Computer Programming*, 29(1-2):3–22, 1997. doi:10.1016/S0167-6423(96)00027-5.

10 Josep Carmona, Jordi Cortadella, and Michael Kishinevsky. New region-based algorithms for deriving bounded Petri nets. *IEEE Transactions on Computers*, 59(3):371–384, 2010. doi:10.1109/TC.2009.131.

11 Josep Carmona, Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev. A symbolic algorithm for the synthesis of bounded Petri nets. In Kees M. van Hee and Rüdiger Valk, editors, *PETRI NETS 2008*, volume 5062 of *LNCS*, pages 92–111. Springer, 2008. doi:10.1007/978-3-540-68746-7_10.

12 Philippe Darondeau. Distributed implementations of Ramadge-Wonham supervisory control with Petri nets. In Eduardo Camacho, editor, *IEEE CDC-ECC 2005*, pages 2107–2112. IEEE, 2005. doi:10.1109/CDC.2005.1582472.

13 Andrzej Ehrenfeucht and Grzegorz Rozenberg. Partial (set) 2-structures. Part I: basic notions and the representation problem and Part II: state spaces of concurrent systems. *Acta Informatica*, 27(4):315–368, 1990. doi:10.1007/BF00264611.

14 Guillaume Feuillade. *Spécification logique de réseaux de Petri*. PhD thesis, Université de Rennes I, 2005.

15 Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

16 Jan Kretínský and Salomon Sickert. MoTraS: A tool for modal transition systems and their extensions. In Dang Van Hung and Mizuhito Ogawa, editors, *ATVA 2013*, volume 8172 of *LNCS*, pages 487–491. Springer, 2013. doi:10.1007/978-3-319-02444-8_41.

17 Kim Larsen. Modal specifications. In Joseph Sifakis, editor, *AVMFSS*, volume 407 of *LNCS*, pages 232–246. Springer, 1989. doi:10.1007/3-540-52148-8_19.

18 Kim Guldstrand Larsen and Liu Xinxin. Equation solving using modal transition systems. In *LICS 1990*, pages 108–117. IEEE Computer Society, 1990. doi:10.1109/LICS.1990.113738.

19 Jean-Baptiste Raclet. Residual for component specifications. *Electr. Notes Theor. Comput. Sci.*, 215:93–110, 2008. doi:10.1016/j.entcs.2008.06.023.

20 Uli Schlachter. Bounded Petri net synthesis from modal transition systems is undecidable. In Josée Desharnais and Radha Jagadeesan, editors, *CONCUR 2016*, volume 59 of *LIPIcs*, pages 15:1–15:14. Schloss Dagstuhl, 2016. doi:10.4230/LIPIcs.CONCUR.2016.15.

21 Rob J. van Glabbeek, Ursula Goltz, and Jens-Wolfhard Schicke-Uffmann. On characterising distributability. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:17)2013.