

PRICED TIMED PETRI NETS *

PAROSH AZIZ ABDULLA ^a AND RICHARD MAYR ^b

^a Uppsala University, Department of Information Technology, Box 337, SE-751 05 Uppsala, Sweden

^b University of Edinburgh, School of Informatics, 10 Crichton Street, Edinburgh EH89AB, UK

ABSTRACT. We consider priced timed Petri nets, i.e., unbounded Petri nets where each token carries a real-valued clock. Transition arcs are labeled with time intervals, which specify constraints on the ages of tokens. Furthermore, our cost model assigns token storage costs per time unit to places, and firing costs to transitions. This general model strictly subsumes both priced timed automata and unbounded priced Petri nets.

We study the cost of computations that reach a given control-state. In general, a computation with minimal cost may not exist, due to strict inequalities in the time constraints. However, we show that the infimum of the costs to reach a given control-state is computable in the case where all place and transition costs are non-negative.

On the other hand, if negative costs are allowed, then the question whether a given control-state is reachable with zero overall cost becomes undecidable. In fact, this negative result holds even in the simpler case of discrete time (i.e., integer-valued clocks).

1. INTRODUCTION

Petri nets [Pet62, Pet77] are a widely used model for the study and analysis of concurrent systems. Many different formalisms have been proposed which extend Petri nets with clocks and real-time constraints, leading to various definitions of *Timed Petri nets (TPNs)*. A complete discussion of all these formalisms is beyond the scope of this paper and the interested reader is referred to the surveys in [Srb08, Bow96, BCH⁺05].

An important distinction is whether the time model is discrete or continuous. In discrete-time nets, time is interpreted as being incremented in discrete steps and thus the ages of tokens are in a countable domain, commonly the natural numbers. Such discrete-time nets have been studied in, e.g., [RGdFE99, dFERA00]. In continuous-time nets, time

2012 ACM CCS: [Theory of computation]: Models of computation; Formal languages and automata theory; [Software and its engineering]: Software organization and properties—Software system structures—Software system models—Petri nets; Software organization and properties—Software functional properties—Formal methods.

Key words and phrases: Formal verification; Petri nets; Timed Automata.

* Extended abstracts of parts of this work have appeared in the proceedings of FOSSACS 2009 [AM09] and LICS 2011 [AM11].

^b Supported by Royal Society grant JP080268.

is interpreted as continuous, and the ages of tokens are real numbers. Some problems for continuous-time nets have been studied in [AN01, AN02, ADMN04, AMM07].

In parallel, there have been several works on extending the model of timed automata [AD94] with *prices (weights)* (see e.g., [ATP01, LBB⁺01, BBR07, BFH⁺01, BFLM11, JT08, BFL⁺08]). Weighted timed automata are suitable models for embedded systems, where one has to take into consideration the fact that the behavior of the system may be constrained by the consumption of different types of resources. Concretely, weighted timed automata extend classical timed automata with a cost function *Cost* that maps every location and every transition to a nonnegative integer (or rational) number. For a transition, *Cost* gives the cost of performing the transition. For a location, *Cost* gives the cost per time unit for staying in the location. In this manner, we can define, for each computation of the system, the accumulated cost of staying in locations and performing transitions along the computation.

Here we consider a very expressive model that subsumes all models mentioned above. *Priced Timed Petri Nets (PTPN)* are a generalization of classic Petri nets [Pet62] with real-valued (i.e., continuous-time) clocks, real-time constraints, and prices for computations.

Each token is equipped with a real-valued clock, representing the age of the token. The firing conditions of a transition include the usual ones for Petri nets. Additionally, each arc between a place and a transition is labeled with a time-interval whose bounds are natural numbers (or possibly ∞ as upper bound). These intervals can be open, closed or half open. Like in timed automata, this is used to encode strict or non-strict inequalities that describe constraints on the real-valued clocks. When firing a transition, tokens which are removed/added from/to places must have ages lying in the intervals of the corresponding transition arcs. Furthermore, we add special *read-arcs* to our model. These affect the enabledness of transitions, but, unlike normal arcs, they do not remove the token from the input place. Read arcs preserve the exact age of the input token, unlike the scenario where a token is first removed and then replaced. Read arcs are necessary in order to make PTPN subsume the classic priced timed automata of [BBBR07].

We assign a cost to computations via a cost function *Cost* that maps transitions and places of the Petri net to natural numbers. For a transition t , $Cost(t)$ gives the cost of performing the transition, while for a place p , $Cost(p)$ gives the cost per time unit per token in the place. The total cost of a computation is given by the sum of all costs of fired transitions plus the storage costs for storing certain numbers of tokens on certain places for certain times during the computation. Like in priced timed automata, having integers as costs and time bounds is not a restriction, because the case of rational numbers can be reduced to the integer case. In most of the paper we consider non-negative costs for transitions and places. In the last section we show that allowing negative costs makes even very basic questions undecidable.

Apart from the cost model, our PTPN are very close to the Timed-Arc Petri Nets of [Srb08] (except for some extensions; see below) in the sense that time is recorded by clocks in the individual tokens, of which there can be unboundedly many. This model differs significantly from the *Time Petri Nets* (also described in [Srb08]) where time is measured by a bounded number of clocks in the transitions. In addition to the cost model, our PTPN also extend the models of [Srb08], [AN01, AN02, ADMN04, AMM07] and [RGdFE99, dFERA00] in two other ways. First, our PTPN model includes read-arcs, which is necessary to subsume (priced) timed automata. Secondly, in PTPN, newly generated tokens do not necessarily have age zero, but instead their age is chosen nondeterministically from specified

intervals (which can be point intervals). Our PTPN model uses a continuous-time semantics (with real-valued clocks) like the models considered in [AN01, AN02, ADMN04, AMM07] and unlike the simpler discrete-time semantics (with integer-valued clocks) considered in [RGdFE99, dFERA00]. We use a non-urgent semantics where tokens can grow older even if this disables the firing of certain transitions (sometimes for ever), like in the Timed-Arc Petri Nets of [Srb08] and unlike in the Time Petri Nets of [Srb08].

Thus, our PTPN are a continuous-time model which subsumes the continuous-time TPN of [AN01, AN02, ADMN04, AMM07], the Timed-Arc Petri Nets of [Srb08] and the priced timed automata of [ATP01, LBB⁺01, BBBR07]. It should be noted that PTPN are infinite-state in several different ways. First, the Petri net itself is unbounded. So the number of tokens (and thus the number of clocks) can grow beyond any bound, i.e., the PTPN can create and destroy arbitrarily many clocks. In that PTPN differ from the priced timed automata of [ATP01, LBB⁺01, BBBR07], which have only a finite number of control-states and only a fixed finite number of clocks. Secondly, every single clock value is a real number of which there are uncountably many.

Our contribution. We study the cost to reach a given control-state in a PTPN. In Petri net terminology, this is called a control-state reachability problem or a coverability problem. The related reachability problem (i.e., reaching a particular configuration) is undecidable for (continuous-time and discrete-time) TPN [RGdFE99], even without taking costs into account. In general, a cost-optimal computation may not exist (e.g., even in priced timed automata it can happen that there is no computation of cost 0, but there exist computations of cost $\leq \epsilon$ for every $\epsilon > 0$).

Our main contribution is to show that the *infimum* of the costs to reach a given control-state is computable, provided that all transition and place costs are non-negative.

This cost problem had been shown to be decidable for the much simpler model of *discrete-time* PTPN in [AM09]. However, discrete-time PTPN do not subsume the priced timed automata of [BBBR07]. Moreover, the techniques from [AM09] do not carry over to the continuous-time domain (e.g., arbitrarily many delays of length 2^{-n} for $n = 1, 2, \dots$ can happen in ≤ 1 time).

On the other hand, if negative costs are allowed, then even very basic questions become undecidable. In Section 10 we show that the question whether a given control-state is reachable with zero overall cost is undecidable if negative transition costs are allowed. This negative result does not need real-valued clocks; it even holds in the simpler case of discrete time (i.e., integer-valued) clocks.

Outline of Used Techniques. Since the PTPN model is very expressive, several powerful new techniques are developed in this paper to analyze them. These techniques are interesting in their own right and can be instantiated to solve other problems.

In Section 2 we define PTPN and the priced coverability problem, and describe its relationship with priced timed automata and Petri nets. Then, in Sections 3–5, we reduce the priced coverability problem for PTPN to a coverability problem in an abstracted untimed model called AC-PTPN. This abstraction is done by an argument similar to a construction in [BBBR07], where parameters indicating a feasible computation are contained in a polyhedron, which is described by a totally unimodular matrix. However, our class of matrices is more general than in [BBBR07], because PTPN allow the creation of new clocks with a nonzero value. The resulting AC-PTPN are still much more expressive than Petri nets,

because their configurations are arbitrarily long sequences of multisets (instead of a single multiset in the case of normal Petri nets). Moreover, the transitions of AC-PTPN are not monotone, because larger configurations cost more and might thus exceed the cost limit.

In order to solve coverability for AC-PTPN, we develop a very general method to solve reachability/coverability problems in infinite-state transition systems which are more general than the well-quasi-ordered/well-structured transition systems of [AČJT00, FS01]. We call this method the *abstract phase construction*, and it is described in abstract terms in Section 6. In particular, it includes a generalization of the Valk-Jantzen construction [VJ85] to arbitrary well-quasi-ordered domains.

In Section 7, we instantiate this abstract method with AC-PTPN and prove the main result. This instantiation is nontrivial and requires several auxiliary lemmas, which ultimately use the decidability of the reachability problem for Petri nets with one inhibitor arc [Rei08, Bon11]. There exist close connections between timed Petri nets, Petri nets with one inhibitor arc, and transfer nets. In Section 8 we establish a connection between a subclass of transfer nets called *simultaneous-disjoint-transfer nets* (SD-TN) and Petri nets with one inhibitor arc, and in Section 9 we show the decidability of a crucial property by reducing it to a reachability problem for SD-TN. By combining all parts, we show the main result, i.e., the computability of the infimum of the costs for PTPN.

The undecidability result for general costs of Section 10 is shown by a direct reduction from Minsky 2-counter machines, where a tradeoff between positive and negative costs is used to ensure a faithful simulation.

2. PRICED TIMED PETRI NETS

2.1. Preliminaries. We use $\mathbb{N}, \mathbb{R}_{\geq 0}, \mathbb{R}_{> 0}$ to denote the sets of natural numbers (including 0), nonnegative reals, and strictly positive reals, respectively. For a natural number k , we use \mathbb{N}^k and \mathbb{N}_{ω}^k to denote the set of vectors of size k over \mathbb{N} and $\mathbb{N} \cup \{\omega\}$, respectively (ω represents the first limit ordinal). For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{0, \dots, n\}$. For $x \in \mathbb{R}_{\geq 0}$, we use $\text{frac}(x)$ to denote the fractional part of x . We use a set Intrv of intervals. An open interval is written as $(w : z)$ where $w \in \mathbb{N}$ and $z \in \mathbb{N} \cup \{\infty\}$. Intervals can also be closed in one or both directions, e.g. $[w : z]$ is closed in both directions and $[w : z)$ is closed to the left and open to the right.

For a set A , we use A^* and A° to denote the set of finite words and finite multisets over A , respectively. We view a multiset b over A as a mapping $b : A \mapsto \mathbb{N}$. Sometimes, we write finite multisets as lists (possibly with multiple occurrences), so both $[2.4, 2.4, 2.4, 5.1, 5.1]$ and $[2.4^3, 5.1^2]$ represent a multiset b over $\mathbb{R}_{\geq 0}$ where $b(2.4) = 3$, $b(5.1) = 2$ and $b(x) = 0$ for $x \neq 2.4, 5.1$. For multisets b_1 and b_2 over A , we say that $b_1 \leq b_2$ if $b_1(a) \leq b_2(a)$ for each $a \in A$. We define $b_1 + b_2$ to be the multiset b where $b(a) = b_1(a) + b_2(a)$, and (assuming $b_1 \leq b_2$) we define $b_2 - b_1$ to be the multiset b where $b(a) = b_2(a) - b_1(a)$, for each $a \in A$. We use $a \in b$ to denote that $b(a) > 0$. We use \emptyset or $[\]$ to denote the empty multiset and ε to denote the empty word.

Let (A, \leq) be a poset. We define a partial order \leq^w on A^* as follows. Let $a_1 \dots a_n \leq^w b_1 \dots b_m$ iff there is a subsequence $b_{j_1} \dots b_{j_n}$ of $b_1 \dots b_m$ s.t. $\forall k \in \{1, \dots, n\}. a_k \leq b_{j_k}$. A subset $B \subseteq A$ is said to be *upward closed* in A if $a_1 \in B, a_2 \in A$ and $a_1 \leq a_2$ implies $a_2 \in B$. If A is known from the context, then we say simply that B is *upward closed*. For $B \subseteq A$ we define the *upward closure* $B \uparrow$ to be the set $\{a \in A \mid \exists a' \in B : a' \leq a\}$. A *downward closed* set

B and the *downward closure* $B\downarrow$ are defined in a similar manner. We use $a\uparrow$, $a\downarrow$, a instead of $\{a\}\uparrow$, $\{a\}\downarrow$, $\{a\}$, respectively.

Given a transition relation \longrightarrow , we denote its transitive closure by \longrightarrow^+ and its reflexive-transitive closure by \longrightarrow^* . Given a set of configurations C , let $Pre_{\longrightarrow}(C) = \{c' \mid \exists c \in C. c' \longrightarrow c\}$ and $Pre_{\longrightarrow}^*(C) = \{c' \mid \exists c \in C. c' \longrightarrow^* c\}$.

2.2. Priced Timed Petri Nets. A *Priced Timed Petri Net (PTPN)* is a tuple $\mathcal{N} = (Q, P, T, Cost)$ where Q is a finite set of control-states and P is a finite set of places. Though control-states can in principle be encoded into extra places, it is conceptually useful to distinguish them. From a modeling point of view, this distinguishes the control-flow from the data. In technical proofs, it is useful to distinguish the finite memory of the control from the infinite memory of the (timed) tokens in the Petri net. T is a finite set of transitions, where each transition $t \in T$ is of the form $t = (q_1, q_2, In, Read, Out)$. We have that $q_1, q_2 \in Q$ are the source control-state and target control-state, respectively, and $In, Read, Out \in (P \times Inrv)^\circ$ are finite multisets over $P \times Inrv$ which define the input-arcs, read-arcs and output-arcs of t , respectively. $Cost : P \cup T \rightarrow \mathbb{N}$ is the cost function assigning firing costs to transitions and storage costs to places. Note that it is not a restriction to use integers for time bounds and costs in PTPN. By the same standard technique as in timed automata, the problem for rational numbers can be reduced to the integer case (by multiplying all numbers with the least common multiple of the divisors). To simplify the presentation we use a one-dimensional cost. This can be generalized to multidimensional costs; see Section 11. We let cmx denote the maximum integer appearing on the arcs of a given PTPN. A *configuration* of \mathcal{N} is a tuple (q, M) where $q \in Q$ is a control-state and M is a *marking* of \mathcal{N} . A marking is a multiset over $P \times \mathbb{R}_{\geq 0}$, i.e., $M \in (P \times \mathbb{R}_{\geq 0})^\circ$. The marking M defines the numbers and ages of tokens in each place in the net. We identify a token in a marking M by the pair (p, x) representing its place and age in M . Then, $M(p, x)$ defines the number of tokens with age x in place p . Abusing notation, we define, for each place p , a multiset $M(p)$ over $\mathbb{R}_{\geq 0}$, where $M(p)(x) = M(p, x)$.

For a marking M of the form $[(p_1, x_1), \dots, (p_n, x_n)]$ and $x \in \mathbb{R}_{> 0}$, we use M^{+x} to denote the marking $[(p_1, x_1 + x), \dots, (p_n, x_n + x)]$.

2.3. Computations. We define two transition relations on the set of configurations: timed transition and discrete transition. A *timed transition* increases the age of each token by the same real number. Formally, for $x \in \mathbb{R}_{> 0}, q \in Q$, we have $(q, M_1) \xrightarrow{x}_{Time} (q, M_2)$ if $M_2 = M_1^{+x}$. We use $(q, M_1) \longrightarrow_{Time} (q, M_2)$ to denote that $(q, M_1) \xrightarrow{x}_{Time} (q, M_2)$ for some $x \in \mathbb{R}_{> 0}$.

We define the set of *discrete transitions* \longrightarrow_{Disc} as $\bigcup_{t \in T} \longrightarrow_t$, where \longrightarrow_t represents the effect of *firing* the discrete transition t . To define \longrightarrow_t formally, we need the auxiliary predicate *match* that relates markings with the inputs/reads/outputs of transitions. Let $M \in (P \times \mathbb{R}_{\geq 0})^\circ$ and $\alpha \in (P \times Inrv)^\circ$. Then $match(M, \alpha)$ holds iff there exists a bijection $f : M \mapsto \alpha$ s.t. for every $(p, x) \in M$ we have $f((p, x)) = (p', \mathcal{I})$ with $p' = p$ and $x \in \mathcal{I}$. Let $t = (q_1, q_2, In, Read, Out) \in T$. Then we have a discrete transition $(q_1, M_1) \longrightarrow_t (q_2, M_2)$ iff there exist $I, O, R, M_1^{rest} \in (P \times \mathbb{R}_{\geq 0})^\circ$ s.t. the following conditions are satisfied:

- $M_1 = I + R + M_1^{rest}$
- $match(I, In)$, $match(R, Read)$ and $match(O, Out)$.

- $M_2 = O + R + M_1^{rest}$

We say that t is *enabled* in (q_1, M_1) if the first two conditions are satisfied. A transition t may be fired iff for each input-arc and each read-arc, there is a token with the right age in the corresponding input place. These tokens in I matched to the input arcs will be removed when the transition is fired, while the tokens in R matched to the read-arcs are kept. The newly produced tokens in O have ages which are chosen nondeterministically from the relevant intervals on the output arcs of the transitions. This semantics is lazy, i.e., enabled transitions do not need to fire and can be disabled again.

We write $\longrightarrow = \longrightarrow_{Time} \cup \longrightarrow_{Disc}$ to denote all transitions. For sets C, C' of configurations, we write $C \xrightarrow{*} C'$ to denote that $c \xrightarrow{*} c'$ for some $c \in C$ and $c' \in C'$. A *computation* π (from c to c') is a sequence of transitions $c_0 \longrightarrow c_1 \longrightarrow \dots \longrightarrow c_n$ such that $c_0 = c$ and $c_n = c'$. We write $c \xrightarrow{\pi} c'$ to denote that π is a computation from c to c' . Similarly, we write $C \xrightarrow{\pi} C'$ to denote that $\exists c_1 \in C, c_n \in C'. c_1 \xrightarrow{\pi} c_n$.

2.4. Costs. The cost of a computation consisting of one discrete transition $t \in T$ is defined as $Cost((q_1, M_1) \longrightarrow_t (q_2, M_2)) := Cost(t)$. The cost of a computation consisting of one timed transition is defined by $Cost((q, M) \xrightarrow{x} (q, M^{+x})) := x * \sum_{p \in P} |M(p)| * Cost(p)$. The cost of a computation is the sum of all transition costs in it, i.e.,

$$Cost((q_1, M_1) \longrightarrow (q_2, M_2) \longrightarrow \dots \longrightarrow (q_n, M_n)) := \sum_{1 \leq i < n} Cost((q_i, M_i) \longrightarrow (q_{i+1}, M_{i+1}))$$

We write $C \xrightarrow{v} C'$ to denote that there is a computation π such that $C \xrightarrow{\pi} C'$ and $Cost(\pi) \leq v$. We define $OptCost(C, C')$ to be the infimum of the set $\{v \mid C \xrightarrow{v} C'\}$, i.e., the infimum of the costs of all computations leading from C to C' . We use the infimum, because the minimum does not exist in general. We partition the set of places $P = P_c \cup P_f$ where $Cost(p) > 0$ for $p \in P_c$ and $Cost(p) = 0$ for $p \in P_f$. The places in P_c are called cost-places and the places in P_f are called free-places.

2.5. Relation of PTPN to Other Models. PTPN subsume the priced timed automata of [ATP01, LBB⁺01, BBBR07] via the following simple encoding. For every one of the finitely many clocks of the automaton we have one place in the PTPN with exactly one token on it whose age encodes the clock value. We assign cost zero to these places. For every control-state s of the automaton we have one place p_s in the PTPN. Place p_s contains exactly one token iff the automaton is in state s , and it is empty otherwise. An automaton transition from state s to state s' is encoded by a PTPN transition consuming the token from p_s and creating a token on $p_{s'}$. The transition guards referring to clocks are encoded as read-arcs to the places which encode clocks, labeled with the required time intervals. Note that open and half-open time intervals are needed to encode the strict inequalities used in timed automata. Clock resets are encoded by consuming the timed token (by an input-arc) and replacing it (by an output-arc) with a new token on the same place with age 0. The cost of staying in state s is encoded by assigning a cost to place p_s , and the cost of performing a transition is encoded as the cost of the corresponding PTPN transition. Also PTPN subsume fully general unbounded (i.e., infinite-state) Petri nets (by setting all time intervals to $[0 : \infty)$ and thus ignoring the clock values).

Note that (just like for timed automata) the problems for continuous-time PTPN cannot be reduced to (or approximated by) the discrete-time case. Replacing strict inequalities

with non-strict ones might make the final control-state reachable, when it originally was unreachable (i.e., it would cause qualitative differences, and not only quantitative ones).

2.6. The Priced Coverability Problem. We will consider two variants of the cost problem, the *Cost-Threshold* problem and the *Cost-Optimality* problem. They are both characterized by an *initial* control state q_{init} and a *final* control state q_{fin} .

Let $c_{init} = (q_{init}, [])$ be the initial configuration and $C_{fin} = \{(q_{fin}, M) \mid M \in (P \times \mathbb{R}_{\geq 0})^{\odot}\}$ the set of final configurations defined by the control-state q_{fin} . I.e., we start from a configuration where the control state is q_{init} and where all the places are empty, and then consider the cost of computations that takes us to q_{fin} . (If c_{init} contained tokens with a non-integer age then the optimal cost might not be an integer.)

In the *Cost-Threshold* problem we ask the question whether $OptCost(c_{init}, C_{fin}) \leq v$ for a given threshold $v \in \mathbb{N}$.

In the *Cost-Optimality* problem, we want to compute $OptCost(c_{init}, C_{fin})$.

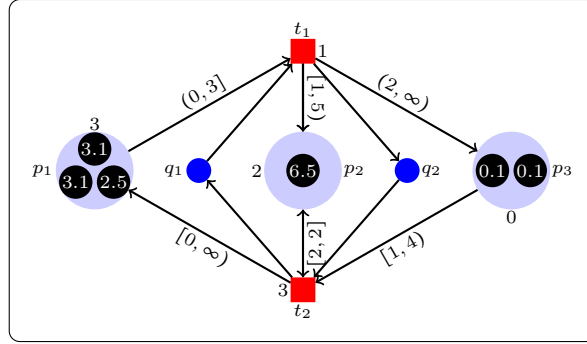


Figure 1: A simple example of a PTPN.

2.7. A Running Example. Figure 1 shows a simple PTPN. We will use this PTPN to give examples of some of the concepts that we introduce in this paper.

Places and Transitions. The PTPN has two control states (q_1 and q_2) depicted as dark-colored circles, three places (p_1 , p_2 , p_3) depicted as light-colored circles, and two transitions (t_1 and t_2) depicted as rectangles. Source/target control states, input/output places are indicated by arrows to the relevant transition. Read places are indicated by double headed arrows. The source and target control states of t_1 are q_1 , resp. q_2 . The input, read resp. output arcs of t_1 are given by the multisets $[(p_1, (0, 3))]$, $[\]$ resp. $[(p_2, [1, 5]), (p_3, (2, \infty))]$. In a similar manner, t_2 is defined by the tuple $(q_2, q_1, [(p_3, [1, 4]), (p_2, [2, 2]), (p_1, [0, \infty))])$. The prices of t_1, t_2, p_1, p_2, p_3 are 1, 3, 3, 2, 0 respectively. The value of cm_{ax} is 5.

Markings. Figure 1 shows a marking $[(p_1, 3.1)^2, (p_1, 2.5), (p_2, 6.5), (p_3, 0.1)^2]$.

Computations and Prices. An example of a computation π is:

$$\begin{aligned}
& (q_1, [(p_1, 3.1)^2, (p_1, 2.5), (p_2, 6.5), (p_3, 0.1)^2]) \\
& \xrightarrow{t_1} (q_2, [(p_1, 3.1)^2, (p_2, 6.5), (p_2, 1.3), (p_3, 0.1)^2, (p_3, 2.2)]) \\
& \xrightarrow[0.7]{Time} (q_2, [(p_1, 3.8)^2, (p_2, 7.2), (p_2, 2.0), (p_3, 0.8)^2, (p_3, 2.9)]) \\
& \xrightarrow{t_2} (q_1, [(p_1, 3.8)^2, (p_1, 9.2), (p_2, 7.2), (p_2, 2.0), (p_3, 0.8)^2]) \\
& \xrightarrow[1.3]{Time} (q_1, [(p_1, 5.1)^2, (p_1, 10.5), (p_2, 8.5), (p_2, 3.3), (p_3, 2.1)^2])
\end{aligned}$$

The cost $Cost(\pi)$ is given by

$$\begin{aligned}
& 1 + 2 * 3 * 0.7 + 2 * 2 * 0.7 + 3 * 0 * 0.7 + \\
& 3 + 3 * 3 * 1.3 + 2 * 2 * 1.3 + 1 * 0 * 1.3 = 27.9
\end{aligned}$$

The transition t_2 is not enabled from any of the following configurations:

- The marking $(q_1, [(p_1, 3.8), (p_2, 2.0), (p_3, 2.9)])$ since it does not have the correct control state.
- The marking $(q_2, [(p_1, 3.1)^2, (p_2, 2.0), (p_3, 0.1)^2])$ since it is missing input tokens with the correct ages in p_3 .
- The marking $(q_2, [(p_1, 3.1)^2, (p_2, 1.0), (p_3, 1.1)^2])$ since it is missing read tokens with the correct ages in p_2 .

3. COMPUTATIONS IN δ -FORM

We solve the Cost-Threshold and Cost-Optimality problems for PTPN via a reduction that goes through a series of abstraction steps. First we show that, in order to solve the cost problems, it is sufficient to consider computations of a certain form where the ages of all the tokens are arbitrarily close to an integer. This technique is similar to the corner-point abstraction used in the analysis of priced timed automata [BFH⁺01, BFLM11, BBBR07] where sets of possible clock valuations are described by polyhedra, and where the infimum cost is achieved in the corner-points of these polyhedra. For PTPN, extra difficulties arise from the unbounded growth of configurations by newly created clocks in new tokens and the potential disappearance of old clocks in consumed tokens. Moreover, newly created clocks in PTPN do not necessarily have value zero. This leads to a more complex structure of the matrices that describe the polyhedra of clock valuations than in the case of priced timed automata; see Def. 3.2. However, we show in Lemma 3.4 that these more general matrices are still totally unimodular, which makes the corner-point abstraction possible.

We decompose PTPN markings M into submarkings such that in every submarking the fractional parts (but not necessarily the integer parts) of the token ages are the same. We then arrange these submarkings in a sequence $M_{-m}, \dots, M_{-1}, M_0, M_1, \dots, M_n$ such that M_{-m}, \dots, M_{-1} contain tokens with fractional parts $\geq 1/2$ in increasing order, M_0 contains the tokens with fractional part zero, and M_1, \dots, M_n contain tokens with fractional parts $< 1/2$ in increasing order.

For example, the marking $[(p_1, 3.1)^2, (p_1, 2.5), (p_2, 6.5), (p_3, 0.1)^2]$ of Figure 1 is decomposed as $M_{-1} = [(p_1, 2.5), (p_2, 6.5)]$, $M_0 = []$ and $M_1 = [(p_1, 3.1)^2, (p_3, 0.1)^2]$.

Formally, the decomposition of a PTPN marking M into its fractional parts

$$M_{-m}, \dots, M_{-1}, M_0, M_1, \dots, M_n$$

is uniquely defined by the following properties:

- $M = M_{-m} + \dots + M_{-1} + M_0 + M_1 + \dots + M_n$.
- If $(p, x) \in M_i$ and $i < 0$ then $\text{frac}(x) \geq 1/2$. If $(p, x) \in M_0$ then $\text{frac}(x) = 0$. If $(p, x) \in M_i$ and $i > 0$ then $\text{frac}(x) < 1/2$.
- Let $(p_i, x_i) \in M_i$ and $(p_j, x_j) \in M_j$. Then $\text{frac}(x_i) = \text{frac}(x_j)$ iff $i = j$, and if $-m \leq i < j < 0$ or $0 \leq i < j \leq n$ then $\text{frac}(x_i) < \text{frac}(x_j)$.
- $M_i \neq \emptyset$ if $i \neq 0$ (M_0 can be empty, but the other M_i must be non-empty in order to get a unique representation.)

We say that a timed transition $(q, M) \xrightarrow{x} (q, M')$ is *detailed* iff at most one fractional part of any token in M changes its status about reaching or exceeding the next higher integer value. Formally, let ϵ be the fractional part of the token ages in M_{-1} , or $\epsilon = 1/2$ if M_{-1} does not exist. Then $(q, M) \xrightarrow{x} (q, M')$ is *detailed* iff either $0 < x < 1 - \epsilon$ (i.e., no tokens reach the next integer), or $M_0 = \emptyset$ and $x = \epsilon$ (no tokens had integer age, but those in M_{-1} reach integer age). Every computation of a PTPN can be transformed into an equivalent one (w.r.t. reachability and cost) where all timed transitions are detailed, by replacing some long timed transitions with several detailed shorter ones where necessary. Thus we may assume w.l.o.g. that timed transitions are detailed. This property is needed to obtain a one-to-one correspondence between PTPN steps and the steps of A-PTPN, defined in the next section.

For $\delta \in (0 : 1/5]$ the marking $[(p_1, x_1), \dots, (p_n, x_n)]$ is in δ -form if, for all $i : 1 \leq i \leq n$, it is the case that either (i) $\text{frac}(x_i) < \delta$ (low fractional part), or (ii) $\text{frac}(x_i) > 1 - \delta$ (high fractional part). I.e., the age of each token is close to (within $< \delta$) an integer. We choose $\delta \leq 1/5$ to ensure that the cases (i) and (ii) do not overlap, and that they still do not overlap for a new $\delta' \leq 2/5$ after a delay of $\leq 1/5$ time units.

The occurrence of a discrete transition t is said to be in δ -form if its output O is in δ -form, i.e., the ages of the newly generated tokens are close to an integer. This is not a property of the transition t as such, but a property of its occurrence, because it depends on the particular choice of O (which is not fixed but possibly nondeterministic within certain constraints; see Subsection 2.3).

Let $\mathcal{N} = (Q, P, T, \text{Cost})$ be a PTPN and $c_{\text{init}} = (q_{\text{init}}, [])$ and $C_{\text{fin}} = \{(q_{\text{fin}}, M) \mid M \in (P \times \mathbb{R}_{\geq 0})^{\circ}\}$ as in the last section.

For $0 < \delta \leq 1/5$, the computation π is in δ -form iff

- (1) Every occurrence of a discrete transition $c_i \xrightarrow{t} c_{i+1}$ is in δ -form, and
- (2) For every timed transition $c_i \xrightarrow{x} c_{i+1}$ we have either $x \in (0 : \delta)$ or $x \in (1 - \delta : 1)$.

We show that, in order to find the infimum of the possible costs, it suffices to consider computations in δ -form, for arbitrarily small values of $\delta > 0$.

Lemma 3.1. *Let $c_{\text{init}} \xrightarrow{\pi} C_{\text{fin}}$, where π is $c_{\text{init}} = c_0 \longrightarrow \dots \longrightarrow c_{\text{length}} \in C_{\text{fin}}$. Then for every $\delta > 0$ there exists a computation π' in δ -form where $c_{\text{init}} \xrightarrow{\pi'} C_{\text{fin}}$, where π' is $c_{\text{init}} = c'_0 \longrightarrow \dots \longrightarrow c'_{\text{length}} \in C_{\text{fin}}$ s.t. $\text{Cost}(\pi') \leq \text{Cost}(\pi)$, π and π' have the same length and $\forall i : 0 \leq i \leq \text{length}. |c_i| = |c'_i|$. Furthermore, if π is detailed then π' is detailed.*

Proof. Outline of the proof. We construct π' by fixing the structure of the computation π and varying the finitely many real numbers describing the delays of timed transitions and the ages of newly created tokens. The tuples of numbers corresponding to a possible computation are contained in a polyhedron, which is described by inequations via a totally unimodular matrix, and whose vertices thus have integer coordinates. Since the cost function is linear in these numbers, the infimum of the costs can be approximated arbitrarily closely by computations π' whose numbers are arbitrarily close to integers, i.e., computations π' in δ -form for arbitrarily small $\delta > 0$.

Detailed proof. The computation π with $c_{init} \xrightarrow{\pi} C_{fin}$ consists of a sequence of discrete transitions and timed transitions. Let n be the number of timed transitions in π and $x_i > 0$ (for $1 \leq i \leq n$) be the delay of the i -th timed transition in π . Let m be the number of newly created tokens in π . We fix some arbitrary order on these tokens (it does not need to agree with the order of token creation) and call them t_1, \dots, t_m . Let y_i be the age of token t_i when it is created in π . (Recall that the age of new tokens is not always zero, but chosen nondeterministically out of given intervals.)

We now consider the set of all computations π' that have the same structure, i.e., the same transitions, as π , but with modified values of y_1, \dots, y_m and x_1, \dots, x_n . Such computations π' have the same length as π and the sizes of the visited configurations match. Also if π is detailed then π' is detailed.

It remains to show that one such computation π' is in δ -form and $Cost(\pi') \leq Cost(\pi)$.

The set of tuples $(y_1, \dots, y_m, x_1, \dots, x_n)$ for which such a computation π' is feasible is described by a set of inequations that depend on the transition guards. (The initial configuration, and the set of final configurations do not introduce any constraints on $(y_1, \dots, y_m, x_1, \dots, x_n)$, because they are closed under changes to token ages.) The inequations are derived from the following conditions.

- The time always advances, i.e., $x_i > 0$.
- When the token t_j is created by an output arc with interval $[a : b]$ we have $a \leq y_j \leq b$, and similarly with strict inequalities if the interval is (half) open. Note that the bounds a and b are integers (except where $b = \infty$ in which case there is no upper bound constraint).
- Consider a token t_j that is an input of some discrete transition t via an input arc or a read arc labeled with interval $[a : b]$. Note that the bounds a and b are integers (or ∞). Let $x_k, x_{k+1}, \dots, x_{k+l}$ be the delays of the timed transitions that happened between the creation of token t_j and the transition t . Then we must have $a \leq y_j + x_k + x_{k+1} + \dots + x_{k+l} \leq b$. (Similarly with strict inequalities if the interval is (half) open.)

These inequations describe a polyhedron PH which contains all feasible tuples of values $(y_1, \dots, y_m, x_1, \dots, x_n)$. By the precondition of this lemma, there exists a computation $c_{init} \xrightarrow{\pi} C_{fin}$ and thus the polyhedron PH is nonempty. Therefore we obtain the closure of the polyhedron \overline{PH} by replacing all strict inequalities $<, >$ with normal inequalities \leq, \geq . Thus \overline{PH} contains PH , but every point in \overline{PH} is arbitrarily close to a point in PH . Now we show that the vertices of the polyhedron \overline{PH} have integer coordinates.

Let $v = (y_1, \dots, y_m, x_1, \dots, x_n)$ be a column vector of the free variables. Then the polyhedron \overline{PH} can be described by the inequation $M \cdot v \leq c$, where c is a column vector of integers and M is an integer matrix. Now we analyze the shape of the matrix M . Each inequation corresponds to a row in M . If the inequality is \leq then the elements are in $\{0, 1\}$, and if the inequality is \geq then the elements are in $\{0, -1\}$. Each of the inequations above refers to at most one variable y_j , and possibly one continuous block of several variables

$x_k, x_{k+1}, \dots, x_{k+l}$. Moreover, for each y_j , this block (if it is nonempty) starts with the same variable x_k . This is because the $x_k, x_{k+1}, \dots, x_{k+l}$ describe the delays of the timed transitions between the creation of token t_j and the moment where t_j is used. x_k is always the first delay after the creation of t_j , and no delays can be left out. Note that the token t_j can be used more than once, because transitions with read arcs do not consume the token. We present the inequalities in blocks, where the first block contains all which refer to y_1 , the second block contains all which refer to y_2 , etc. The last block contains those inequations that do not refer to any y_j , but only to variables x_i . Inside each block we sort the inequalities w.r.t. increasing length of the $x_k, x_{k+1}, \dots, x_{k+l}$ block, i.e., from smaller values of l to larger ones. (For y_j we have the same k .) Thus the matrix M has the following form:

$$\left(\begin{array}{cccc|cccccc}
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
 \dots & & & & & & & & & \\
 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & -1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
 \dots & & & & & & & & &
 \end{array} \right)$$

Formally, the shape of these matrices is defined as follows.

Definition 3.2. We call a $(z \times m + n)$ -matrix a *PTPN constraint matrix*, if every row has one of the following two forms. Let $j \in \{1, \dots, m\}$ and $k(j) \in \{1, \dots, n\}$ be a number that depends only on j , and let $\alpha \in \{-1, 1\}$. First form: $0^{j-1} \alpha 0^{m-j} 0^{k(j)-1} \alpha^* 0^*$. Second form: $0^* \alpha^* 0^*$. Matrices that contain only rows of the second form all called *3-block matrices* in [BBBR07].

Definition 3.3. [NW88] An integer matrix is called *totally unimodular* iff the determinant of all its square submatrices is equal to 0, 1 or -1 .

Lemma 3.4. *All PTPN constraint matrices are totally unimodular.*

Proof. First, every square submatrix of a PTPN constraint matrix has the same form and is also a PTPN constraint matrix. Thus it suffices to show the property for square PTPN constraint matrices. We show this by induction on the size. The base case of size 1×1 is trivial, because the single value must be in $\{-1, 0, 1\}$. For the induction step consider a square $k \times k$ PTPN constraint matrix M , with some n, m s.t. $n + m = k$. If M does not contain any row of the first form then M is a 3-block matrix and thus totally unimodular by [BBBR07] (Lemma 2). Otherwise, M contains a row i of the first form where $M(i, j) \in \{-1, 1\}$ for some $1 \leq j \leq m$. Without restriction let i be such a row in M where the number of nonzero entries is minimal. Consider all rows i' in M where $M(i', j) \neq 0$. Except for $M(i', j)$, they just contain (at most) one block of elements 1 (or -1) that starts at position $m + k(j)$. By adding/subtracting row i to all these other rows i' where $M(i', j) \neq 0$ we obtain a new matrix M' where $M'(i, j)$ is the only nonzero entry in column j in M' and $\det(M') = \det(M)$. Moreover, M' is also a PTPN constraint matrix, because of the minimality of the nonzero block length in row i and because all these blocks start at $m + k(j)$. I.e., in M' these modified rows i' have the form $0^* 1^* 0^*$ or $0^* (-1)^* 0^*$. We obtain M'' from M' by deleting column j and row i , and M'' is a $(k-1) \times (k-1)$ PTPN constraint matrix (because $j \leq m$). By induction hypothesis, M'' is totally unimodular and $\det(M'') \in \{-1, 0, 1\}$. By

the cofactor method, $\det(M') = (-1)^{i+j} * M'(i, j) * \det(M'') \in \{-1, 0, 1\}$. Thus $\det(M) = \det(M') \in \{-1, 0, 1\}$ and M is totally unimodular. \square

Theorem 3.5. [NW88]. *Consider the polyhedron $\{v \in \mathbb{R}^k \mid M \cdot v \leq c\}$ with M a totally unimodular $(p \times k)$ matrix and $c \in \mathbb{Z}^p$. Then the coordinates of its vertices are integers.*

Since our polyhedron \overline{PH} is described by a PTPN constraint matrix, which is totally unimodular by Lemma 3.4, it follows from Theorem 3.5 that the vertices of \overline{PH} have integer coordinates.

Since the *Cost* function is linear in x_1, \dots, x_n (and does not depend on y_1, \dots, y_m), the infimum of the costs on PH is obtained at a vertex of \overline{PH} , which has integer coordinates by Theorem 3.5. Therefore, one can get arbitrarily close to the infimum cost with values $y_1, \dots, y_m, x_1, \dots, x_n$ which are arbitrarily close to some integers. Thus, for every computation $c_{init} \xrightarrow{\pi} C_{fin}$ there exists a modified computation π' with values $y_1, \dots, y_m, x_1, \dots, x_n$ arbitrarily close to integers (i.e., π' in δ -form for arbitrarily small $\delta > 0$) such that $c_{init} \xrightarrow{\pi'} C_{fin}$ and $Cost(\pi') \leq Cost(\pi)$. (Note that the final configuration reached by π' possibly differs from the final configuration of π in the ages of some tokens. However, this does not matter, because the set of configurations C_{fin} is closed under such changes.) \square

The following corollary follows directly from Lemma 3.1 and shows that in order to find the infimum of the cost it suffices to consider only computations in δ -form for arbitrarily small $\delta > 0$.

Corollary 3.6. *For every $\delta > 0$ we have*

$$OptCost(c_{init}, C_{fin}) = \inf\{Cost(\pi) \mid c_{init} \xrightarrow{\pi} C_{fin}, \pi \text{ in } \delta\text{-form}\}$$

4. ABSTRACT PTPN

We now reduce the Cost-Optimality problem to a simpler case without explicit clocks by defining a new class of systems called *abstract PTPN* (for short A-PTPN), whose computations represent PTPN computations in δ -form, for infinitesimally small values of $\delta > 0$. For each PTPN $\mathcal{N} = (Q, P, T, Cost)$, we define a corresponding A-PTPN \mathcal{N}' , sometimes denoted by $aptpn(\mathcal{N})$. The A-PTPN \mathcal{N}' is syntactically of the same form $(Q, P, T, Cost)$ as \mathcal{N} . However, \mathcal{N}' induces a different transition system, because its configurations and operational semantics are different. Below we define the set of markings of the A-PTPN, and then describe the transition relation. We will also explain the relation to the markings and the transition relation induced by the original PTPN.

Markings and Configurations. Fix a $\delta : 0 < \delta \leq 2/5$. A marking M of \mathcal{N} in δ -form is encoded by a marking $aptpn(M)$ of \mathcal{N}' which is described by a triple (w^{high}, b_0, w^{low}) where $w^{high}, w^{low} \in ((P \times [cmax + 1])^\odot)^*$ and $b_0 \in (P \times [cmax + 1])^\odot$. The ages of the tokens in $aptpn(M)$ are integers and therefore only carry the integral parts of the tokens in the original PTPN. However, the marking $aptpn(M)$ carries additional information about the fractional parts of the tokens as follows. The tokens in w^{high} represent tokens in M that have *high* fractional parts (their values are at most δ below the next integer); the tokens in w^{low} represent tokens in M that have *low* fractional parts (their values at most δ above the previous integer); while tokens in b_0 represent tokens in M that have *zero* fractional parts (their values are equal to an integer). Furthermore, the ordering among the

fractional parts of tokens in w^{high} (resp. w^{low}) is represented by the positions of the multisets to which they belong in w^{high} (resp. w^{low}). Let $M = M_{-m}, \dots, M_{-1}, M_0, M_1, \dots, M_n$ be the decomposition of M into fractional parts, as defined in Section 3. Then we define $aptpn(M) := (w^{high}, b_0, w^{low})$ with $w^{high} = b_{-m} \dots b_{-1}$, and $w^{low} = b_1 \dots b_n$, where $b_i((p, [x])) = M_i((p, x))$ if $x \leq cmax$. (This is well defined, because M_i contains only tokens with one particular fractional part.) Furthermore, $b_0((p, cmax + 1)) = \sum_{y > cmax} M((p, y))$, i.e., all tokens whose age is $> cmax$ are abstracted as tokens of age $cmax + 1$, because the PTPN cannot distinguish between token ages $> cmax$. Note that w^{high} and w^{low} represent tokens with fractional parts in increasing order. An A-PTPN configuration is a control-state plus a marking. If we apply $aptpn$ to a set of configurations (i.e., $aptpn(C_{fin})$), we implicitly restrict this set to the subset of configurations in $2/5$ -form.

Transition Relation. The transitions on the A-PTPN are defined as follows. For every discrete transition $t = (q_1, q_2, In, Read, Out) \in T$ we have $(q_1, b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \xrightarrow{t} (q_2, c_{-m'} \dots c_{-1}, c_0, c_1 \dots c_{n'})$ if the following conditions are satisfied: For every $i : -m \leq i \leq n$ there exist $b_i^I, b_i^R, b_i^{rest}, \hat{O}, b_0^O \in (P \times [cmax + 1])^\circ$ s.t. for every $0 < \epsilon < 1$ we have

- $b_i = b_i^I + b_i^R + b_i^{rest}$ for $-m \leq i \leq n$
- $match((\sum_{i \neq 0} b_i^I)^{+\epsilon} + b_0^I, In)$
- $match((\sum_{i \neq 0} b_i^R)^{+\epsilon} + b_0^R, Read)$
- $match(\hat{O}^{+\epsilon} + b_0^O, Out)$
- There is a strictly monotone injection $f : \{-m, \dots, n\} \mapsto \{-m', \dots, n'\}$ where $f(0) = 0$ s.t. $c_{f(i)} \geq b_i - b_i^I$ and $c_0 = b_0 - b_0^I + b_0^O$ and $\sum_{i \neq 0} c_i = (\sum_{i \neq 0} b_i - b_i^I) + \hat{O}$.

The intuition is that the A-PTPN tokens in b_i for $i \neq 0$ represent PTPN tokens with a little larger, and strictly positive, fractional part. Thus their age is incremented by $\epsilon > 0$ before it is matched to the input, read and output arcs. The fractional parts of the tokens that are not involved in the transition stay the same. However, since all the time intervals in the PTPN have integer bounds, the fractional parts of newly created tokens are totally arbitrary. Thus they can be inserted at any position in the sequence, between any positions in the sequence, or before/after the sequence of existing fractional parts. This is specified by the last condition on the sequence $c_{-m'} \dots c_{-1}, c_0, c_1 \dots c_{n'}$.

The following lemma shows the connection between a PTPN and its corresponding A-PTPN for discrete transition steps.

Lemma 4.1. *Let (q, M) be a PTPN configuration in δ -form for some $\delta \leq 1/5$. There is an occurrence of a discrete transition in δ -form $(q, M) \xrightarrow{t} (q', M')$ if and only if $aptpn((q, M)) \xrightarrow{t} atpnp((q', M'))$.*

Proof. Let $M = M_{-m} + \dots + M_{-1} + M_0 + M_1 + \dots + M_n$ be the unique decomposition of M into increasing fractional parts, and $aptpn(M) := (b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)$, as defined in Section 4. Let $t = (q, q', In, Read, Out)$.

Now we prove the first implication. Provided that $(q, M) \xrightarrow{t} (q', M')$ there exist $I, O, R, M^{rest} \in (P \times \mathbb{R}_{\geq 0})^\circ$ s.t. the following conditions are satisfied:

- $M = I + R + M^{rest}$
- $match(I, In)$, $match(R, Read)$ and $match(O, Out)$.
- $M' = O + R + M^{rest}$.

Thus each M_i can be decomposed into parts $M_i = M_i^I + M_i^R + M_i^{rest}$, where $I = \sum_i M_i^I$, $R = \sum_i M_i^R$, $M^{rest} = \sum_i M_i^{rest}$. Let $b_i^I = atpnp(M_i^I)$, $b_i^R = atpnp(M_i^R)$, $b_i^{rest} = atpnp(M_i^{rest})$.

Then $b_i = b_i^I + b_i^R + b_i^{rest}$. Since the time intervals on transitions have integer bounds, we obtain $match((\sum_{i \neq 0} b_i^I)^{+\epsilon} + b_0^I, In)$ and $match((\sum_{i \neq 0} b_i^R)^{+\epsilon} + b_0^R, Read)$.

Similarly as M , the marking O can be uniquely decomposed into parts with increasing fractional part of the ages of tokens, i.e., $O = O_{-j} + \dots + O_{-1} + O_0 + O_1 + \dots + O_k$. Let $\hat{O} = aptpn(O - O_0)$ and $b_0^O = aptpn(O_0)$. Thus we get $match(\hat{O}^{+\epsilon} + b_0^O, Out)$.

Since $M' = O + R + M^{rest}$, the sequence of the remaining parts of the M_i is merged with the sequence $O_{-j} + \dots + O_{-1} + O_0 + O_1 + \dots + O_k$. Thus M' can be uniquely decomposed into parts with increasing fractional part of the ages of tokens, i.e., $M' = M'_{-m'} + \dots + M'_{-1} + M'_0 + M'_1 + \dots + M'_{n'}$. Let $c_i = aptpn(M'_i)$. Thus there is a strictly monotone injection $f : \{-m, \dots, n\} \mapsto \{-m', \dots, n'\}$ where $f(0) = 0$ s.t. $c_{f(i)} \geq b_i - b_i^I$ and $c_0 = b_0 - b_0^I + b_0^O$ and $\sum_{i \neq 0} c_i = (\sum_{i \neq 0} b_i - b_i^I) + \hat{O}$.

Thus $aptpn((q, M)) = (q, b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \xrightarrow{t} (q', c_{-m'} \dots c_{-1}, c_0, c_1 \dots c_{n'}) = aptpn((q', M'))$.

Now we show the other direction. If $aptpn((q, M)) \xrightarrow{t} aptpn((q', M'))$ then we have $aptpn((q', M')) = (q', c_{-m'} \dots c_{-1}, c_0, c_1 \dots c_{n'})$ s.t.

- $b_i = b_i^I + b_i^R + b_i^{rest}$ for $-m \leq i \leq n$
- $match((\sum_{i \neq 0} b_i^I)^{+\epsilon} + b_0^I, In)$
- $match((\sum_{i \neq 0} b_i^R)^{+\epsilon} + b_0^R, Read)$
- $match(\hat{O}^{+\epsilon} + b_0^O, Out)$
- There is a strictly monotone injection $f : \{-m, \dots, n\} \mapsto \{-m', \dots, n'\}$ where $f(0) = 0$ s.t. $c_{f(i)} \geq b_i - b_i^I$ and $c_0 = b_0 - b_0^I + b_0^O$ and $\sum_{i \neq 0} c_i = (\sum_{i \neq 0} b_i - b_i^I) + \hat{O}$.

As before, each M_i can be decomposed into parts $M_i = M_i^I + M_i^R + M_i^{rest}$, where $b_i^I = aptpn(M_i^I)$, $b_i^R = aptpn(M_i^R)$, and $b_i^{rest} = aptpn(M_i^{rest})$. Let $I = \sum_i M_i^I$, $R = \sum_i M_i^R$, and $M^{rest} = \sum_i M_i^{rest}$. So we have $M = I + R + M^{rest}$. Furthermore, since the interval bounds are integers, we have $match(I, In)$, $match(R, Read)$ and $match(O, Out)$. Finally, due to the conditions on \hat{O} and b_0^O , there exists a marking O s.t. $\hat{O} + b_0^O = aptpn(O)$, $M' = O + R + M^{rest}$ and $aptpn((q', M')) = (q', c_{-m'} \dots c_{-1}, c_0, c_1 \dots c_{n'})$. Moreover, this O can be chosen to be in δ -form, for the following reasons. The tokens in O whose fractional part is the same as a fractional part in M are trivially in δ -form, because M is in δ -form. The tokens in O whose fractional part is between two fractional parts in M is also trivially in δ -form, because M is in δ -form. Now consider the tokens in O whose fractional part is larger than any fractional part in $M_1 + \dots + M_n$. Let δ_1 be the maximal fractional part in $M_1 + \dots + M_n$. We have $\delta_1 < \delta$, because M is in δ -form. Since $\delta_1 < \delta$, the interval $(\delta_1 : \delta)$ is nonempty and contains uncountably many different values. Therefore there is still space for infinitely many different fractional parts in O in the interval $(\delta_1 : \delta)$. Finally consider the tokens in O whose fractional part is smaller than any fractional part in $M_{-m} + \dots + M_{-1}$. Let δ_2 be the minimal fractional part in $M_{-m} + \dots + M_{-1}$. We have $\delta_2 > 1 - \delta$, because M is in δ -form. Therefore there is still space for infinitely many different fractional parts in O in the nonempty interval $(1 - \delta : \delta_2)$.

Thus, since O is in δ -form, the transition $(q, M) \xrightarrow{t} (q', M')$ is in δ -form, as required. \square

Now we show how to encode timed transitions into A-PTPN. We define A-PTPN transitions that encode the effect of PTPN detailed timed transitions \xrightarrow{x} for $x \in (0 : \delta)$ or $x \in (1 - \delta : 1)$ for sufficiently small $\delta > 0$. We call these *abstract timed transitions*. For any multiset $b \in (P \times [cmax + 1])^\circ$ let $b^+ \in (P \times [cmax + 1])^\circ$ be defined by $b^+(p, x + 1) = b(p, x)$

for $x \leq cmax$ and $b^+((p, cmax + 1)) = b((p, cmax + 1)) + b((p, cmax))$, i.e., the age $cmax + 1$ represents all ages $> cmax$. There are 4 different types of abstract timed transitions. (In the following all b_i are nonempty.)

Type 1: $(q_1, b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \longrightarrow (q_1, b_{-m} \dots b_{-1}, \emptyset, b_0 b_1 \dots b_n)$. This simulates a very small delay $\delta > 0$ where the tokens of integer age in b_0 now have a positive fractional part, but no tokens reach an integer age.

Type 2: $(q_1, b_{-m} \dots b_{-1}, \emptyset, b_1 \dots b_n) \longrightarrow (q_1, b_{-m} \dots b_{-2}, b_{-1}^+, b_1 \dots b_n)$. This simulates a very small delay $\delta > 0$ in the case where there were no tokens of integer age and the tokens in b_{-1} just reach the next higher integer age.

Type 3: $(q_1, b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \longrightarrow (q_1, b_{-m}^+ \dots b_{-2}^+ b_{-1}^+ b_0 \dots b_k, \emptyset, b_{k+1}^+ \dots b_n^+)$ for a $k \in \{0, \dots, n\}$. This simulates a delay in $(1 - \delta : 1)$ where the tokens in $b_0 \dots b_k$ do not quite reach the next higher integer and no token gets an integer age.

Type 4: $(q_1, b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \longrightarrow (q_1, b_{-m}^+ \dots b_{-2}^+ b_{-1}^+ b_0 \dots b_k, b_{k+1}^+, b_{k+2}^+ \dots b_n^+)$ for some $k \in \{0, \dots, n-1\}$. This simulates a delay in $(1 - \delta : 1)$ where the tokens in b_0, \dots, b_k do not quite reach the next higher integer and the tokens on b_{k+1} just reach the next higher integer age.

The cost model for A-PTPN is defined as follows. For every transition $t \in T$ we have $Cost((q_1, M_1) \xrightarrow{t} (q_2, M_2)) := Cost(t)$, just like in PTPN. For abstract timed transitions of types 1 and 2 we define the cost as zero. For abstract timed transitions $(q, M_1) \longrightarrow (q, M_2)$ of types 3 and 4, we define $Cost((q, M_1) \longrightarrow (q, M_2)) := \sum_{p \in P} |M_1(p)| * Cost(p)$ (i.e., as if the elapsed time had length 1). The intuition is that, as δ converges to zero, the cost of the PTPN timed transitions of length in $(0 : \delta)$ (types 1 and 2) or in $(1 - \delta : 1)$ (types 3 and 4) converges to the cost of the corresponding abstract timed transitions in the A-PTPN. This will be shown formally in Lemma 4.4.

The following two lemmas show the connection between (detailed) timed transitions in PTPN and abstract timed transitions in the corresponding A-PTPN.

Lemma 4.2. *Let (q, M) be a PTPN configuration in δ -form for some $\delta \leq 1/5$ and $x \in (0 : \delta)$. There is a PTPN detailed timed transition $(q, M) \xrightarrow{x} (q, M^{+x})$ if and only if there is an A-PTPN abstract timed transition of type 1 or 2 s.t. $aptpn((q, M)) \longrightarrow aptpn((q, M^{+x}))$.*

Proof. Let $M = M_{-m} + \dots + M_{-1} + M_0 + M_1 + \dots + M_n$ be the unique decomposition of M into increasing fractional parts (as defined in Section 3), and $aptpn(M) := (b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)$, as defined as above. Let ϵ be the fractional part of the ages of the tokens in M_{-1} . Since (q, M) is in δ -form, we have $0 < 1 - \epsilon < \delta$. Now there are two cases.

In the first case we have $x < 1 - \epsilon$. Then the tokens in M_{-1}^{+x} will have fractional part $\epsilon + x \in (1 - \delta : 1)$, and the tokens in M_0^{+x} will have fractional part $x \in (0 : \delta)$. Therefore $aptpn((q, M)) = (q, (b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)) \longrightarrow (q, (b_{-m} \dots b_{-1}, \emptyset, b_0 b_1 \dots b_n)) = aptpn((q, M^{+x}))$, by an A-PTPN abstract timed transition of type 1, if and only if $(q, M) \xrightarrow{x} (q, M^{+x})$.

In the second case we must have $x = 1 - \epsilon$ and $M_0 = \emptyset$, because $(q, M) \xrightarrow{x} (q, M^{+x})$ is a detailed timed transition. In this case exactly the tokens in M_{-1} reach the next higher integer age, i.e., the tokens in M_{-1}^{+x} have integer age and the integer is one higher than the integer part of the age of the tokens in M_0 . Thus $aptpn((q, M)) = (q, (b_{-m} \dots b_{-1}, \emptyset, b_1 \dots b_n)) \longrightarrow (q, (b_{-m} \dots b_{-2}, b_{-1}^+, b_1 \dots b_n)) = aptpn((q, M^{+x}))$, by an A-PTPN abstract timed transition of type 2, if and only if $(q, M) \xrightarrow{x} (q, M^{+x})$. \square

Lemma 4.3. *Let (q, M) be a PTPN configuration in δ -form for some $\delta \leq 1/5$ and $x \in (1 - \delta : 1)$. There is a PTPN timed transition $(q, M) \xrightarrow{x} (q, M^{+x})$ if and only if there is an A-PTPN transition of either type 3 or 4 s.t. $\text{aptpn}((q, M)) \longrightarrow \text{aptpn}((q, M^{+x}))$.*

Proof. Let $M = M_{-m} + \dots + M_{-1} + M_0 + M_1 + \dots + M_n$ be the unique decomposition of M into increasing fractional parts (as defined in Section 3), and $\text{aptpn}(M) := (b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)$, as defined above. Let ϵ_k be the fractional part of the ages of the tokens in M_k for $0 \leq k \leq n$. Since (q, M) is in δ -form, we have $0 < \epsilon_k < \delta$. Now there are two cases.

In the first case we have $x \in (1 - \epsilon_{k+1} : 1 - \epsilon_k) \subseteq (1 - \delta : 1)$ for some $0 \leq k \leq n$. (If $k = n$ we have $x \in (1 - \delta : 1 - \epsilon_n)$, and if $k = 0$ we have $x \in (1 - \epsilon_1 : 1)$.) Then, in the step from M_{k+1} to M_{k+1}^{+x} , the token ages in M_{k+1} reach and slightly exceed the next higher integer age, while the token ages in M_k^{+x} still stay slightly below the next higher integer. Therefore $\text{aptpn}((q, M)) = (q, (b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)) \longrightarrow (q, (b_{-m}^+ \dots b_{-1}^+ b_0 \dots b_k, \emptyset, b_{k+1}^+ \dots b_n^+)) = \text{aptpn}((q, M^{+x}))$, by an A-PTPN abstract timed transition of type 3, if and only if $(q, M) \xrightarrow{x} (q, M^{+x})$.

The only other case is where $x = 1 - \epsilon_{k+1}$ for some $k \in \{0, \dots, n-1\}$. Here exactly the tokens in M_{k+1} reach the next higher integer age. Therefore $\text{aptpn}((q, M)) = (q, (b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)) \longrightarrow (q, (b_{-m}^+ \dots b_{-1}^+ b_0 \dots b_k, b_{k+1}^+, b_{k+1}^+ \dots b_n^+)) = \text{aptpn}((q, M^{+x}))$, by an A-PTPN abstract timed transition of type 4, if and only if $(q, M) \xrightarrow{x} (q, M^{+x})$. \square

The following Lemma 4.4, which follows from Lemmas 4.1, 4.2, and 4.3, shows the connection between the computation costs in a PTPN and the corresponding A-PTPN.

Lemma 4.4.

(1) *Let c_0 be a PTPN configuration where all tokens have integer ages. For every PTPN computation $\pi = c_0 \longrightarrow \dots \longrightarrow c_n$ in detailed form and δ -form s.t. $n * \delta \leq 1/5$ there exists a corresponding A-PTPN computation $\pi' = \text{aptpn}(c_0) \longrightarrow \dots \longrightarrow \text{aptpn}(c_n)$ s.t.*

$$|\text{Cost}(\pi) - \text{Cost}(\pi')| \leq n * \delta * (\max_{0 \leq i \leq n} |c_i|) * (\max_{p \in P} \text{Cost}(p))$$

(2) *Let c'_0 be an A-PTPN configuration $(\epsilon, b_0, \epsilon)$. For every A-PTPN computation $\pi' = c'_0 \longrightarrow \dots \longrightarrow c'_n$ and every $0 < \delta \leq 1/5$ there exists a PTPN computation $\pi = c_0 \longrightarrow \dots \longrightarrow c_n$ in detailed form and δ -form s.t. $c'_i = \text{aptpn}(c_i)$ for $0 \leq i \leq n$ and*

$$|\text{Cost}(\pi) - \text{Cost}(\pi')| \leq n * \delta * (\max_{0 \leq i \leq n} |c'_i|) * (\max_{p \in P} \text{Cost}(p))$$

Proof. For the first part let $\pi = c_0 \longrightarrow \dots \longrightarrow c_n$ be a PTPN computation in detailed form and δ -form s.t. $n * \delta \leq 1/5$. So every timed transition \xrightarrow{x} has either $x \in (0 : \delta)$ or $x \in (1 - \delta : 1)$. Furthermore, the fractional part of the age of every token in any configuration c_i is $< i * \delta$ away from the nearest integer, because c_0 only contains tokens with integer ages. Since $i \leq n$ these ages are $< n * \delta \leq 1/5$ away from the nearest integer. Moreover, π is detailed and thus Lemmas 4.1, 4.2 and 4.3 apply. Thus there exists a corresponding A-PTPN computation $\pi' = \text{aptpn}(c_0) \longrightarrow \dots \longrightarrow \text{aptpn}(c_n)$. By definition of the cost of A-PTPN transitions, for every discrete transition $c_i \longrightarrow c_{i+1}$ we have $\text{Cost}(c_i \longrightarrow c_{i+1}) = \text{Cost}(\text{aptpn}(c_i) \longrightarrow \text{aptpn}(c_{i+1}))$. Moreover, for every timed transition $c_i \xrightarrow{x} c_{i+1}$ we have $|\text{Cost}(c_i \xrightarrow{x} c_{i+1}) - \text{Cost}(\text{aptpn}(c_i) \longrightarrow \text{aptpn}(c_{i+1}))| \leq \delta * |c_i| * (\max_{p \in P} \text{Cost}(p))$, because either $x \in (0 : \delta)$ or $x \in (1 - \delta : 1)$. Therefore $|\text{Cost}(\pi) - \text{Cost}(\pi')| \leq n * \delta * (\max_{0 \leq i \leq n} |c_i|) * (\max_{p \in P} \text{Cost}(p))$ as required.

For the second part let c_0 be a PTPN configuration s.t. $(\epsilon, b_0, \epsilon) = c'_0 = \text{aptpn}(c_0)$, i.e., all tokens in c_0 have integer ages. We now use Lemmas 4.1, 4.2 and 4.3 to construct the PTPN computation π . Let $\delta_i := \delta * 2^{i-n}$ for $0 \leq i \leq n$. The construction ensures the following invariants. (1) $c'_i = \text{aptpn}(c_i)$, and (2) c_i is in δ_i -form. Condition (1) follows directly from Lemmas 4.1, 4.2 and 4.3. For the base case $i = 0$, condition (2) holds trivially, because all tokens in c_0 have integer ages. Now we consider the step from i to $i + 1$. Since c_i is in δ_i -form, we obtain from Lemmas 4.1, 4.2 and 4.3 that if the i -th transition in this sequence is a timed transition \xrightarrow{x} then either $x \in (0 : \delta_i)$ or $x \in (1 - \delta_i : 1)$. Therefore, since c_i is in δ_i -form, c_{i+1} is in $(2 * \delta_i)$ -form and thus in δ_{i+1} -form.

Now we consider the cost of the PTPN computation π . By definition of the cost of A-PTPN transitions, for every discrete transition $c_i \rightarrow c_{i+1}$ we have $\text{Cost}(c_i \rightarrow c_{i+1}) = \text{Cost}(\text{aptpn}(c_i) \rightarrow \text{aptpn}(c_{i+1}))$. Moreover, for every timed transition $c_i \xrightarrow{x} c_{i+1}$ we have $|\text{Cost}(c_i \xrightarrow{x} c_{i+1}) - \text{Cost}(\text{aptpn}(c_i) \rightarrow \text{aptpn}(c_{i+1}))| \leq \delta_i * |c'_i| * (\max_{p \in P} \text{Cost}(p))$, because either $x \in (0 : \delta_i)$ or $x \in (1 - \delta_i : 1)$. Therefore $|\text{Cost}(\pi) - \text{Cost}(\pi')| \leq n * \delta * (\max_{0 \leq i \leq n} |c'_i|) * (\max_{p \in P} \text{Cost}(p))$ as required. \square

The following theorem summarizes the results of this section. To compute the optimal cost of a PTPN, it suffices to consider the corresponding A-PTPN.

Theorem 4.5. *The infimum of the costs in a PTPN coincides with the infimum of the costs in the corresponding A-PTPN.*

$$\inf\{\text{Cost}(\pi) \mid c_{init} \xrightarrow{\pi} C_{fin}\} = \inf\{\text{Cost}(\pi') \mid \text{aptpn}(c_{init}) \xrightarrow{\pi'} \text{aptpn}(C_{fin})\}$$

Proof. We show that neither of the two costs for PTPN and A-PTPN can be larger than the other.

Let $I := \inf\{\text{Cost}(\pi) \mid c_{init} \xrightarrow{\pi} C_{fin}\}$ and $I' := \inf\{\text{Cost}(\pi') \mid \text{aptpn}(c_{init}) \xrightarrow{\pi'} \text{aptpn}(C_{fin})\}$.

First we show that $I' \not\leq I$. By definition of I , for every $\lambda > 0$ there is a computation $c_{init} \xrightarrow{\pi_\lambda} C_{fin}$, s.t. $\text{Cost}(\pi_\lambda) - I \leq \lambda$. Without restriction we can assume that π_λ is also in detailed form. Let $n_\lambda := |\pi_\lambda|$ be the length of π_λ and $\pi_\lambda = c_0 \rightarrow \dots \rightarrow c_{n_\lambda}$. Let $\delta_\lambda := \min\{1/(5n_\lambda), \lambda/(n_\lambda * (\max_{0 \leq i \leq n_\lambda} |c_i|) * (\max_{p \in P} \text{Cost}(p)))\}$.

By Lemma 3.1 there exists a computation $c_{init} \xrightarrow{\pi''_\lambda} C_{fin}$ in detailed form and δ_λ -form where $|\pi''_\lambda| = |\pi_\lambda|$ and $\pi''_\lambda = c''_0 \rightarrow \dots \rightarrow c''_{n_\lambda}$ s.t. $|c''_i| = |c_i|$ and $\text{Cost}(\pi''_\lambda) \leq \text{Cost}(\pi_\lambda)$. It follows that $\text{Cost}(\pi''_\lambda) - I \leq \lambda$.

By part (1) of Lemma 4.4, there exists a corresponding A-PTPN computation $\pi'_\lambda = \text{aptpn}(c''_0) \rightarrow \dots \rightarrow \text{aptpn}(c''_{n_\lambda})$ s.t. $|\text{Cost}(\pi''_\lambda) - \text{Cost}(\pi'_\lambda)| \leq n_\lambda * \delta_\lambda * (\max_{0 \leq i \leq n_\lambda} |c''_i|) * (\max_{p \in P} \text{Cost}(p)) \leq \lambda$. Thus we obtain $\text{Cost}(\pi'_\lambda) - I \leq 2\lambda$. Since this holds for every $\lambda > 0$ we get $I' \not\leq I$.

Now we show that $I \not\leq I'$. By definition of I' , for every $\lambda > 0$ there is a A-PTPN computation $c_{init} \xrightarrow{\pi'_\lambda} C_{fin}$, s.t. $\text{Cost}(\pi'_\lambda) - I' \leq \lambda$. Let $n_\lambda := |\pi'_\lambda|$ be the length of π'_λ and $\pi'_\lambda = c'_0 \rightarrow \dots \rightarrow c'_{n_\lambda}$. Let $\delta_\lambda := \min\{1/(5n_\lambda), \lambda/(n_\lambda * (\max_{0 \leq i \leq n_\lambda} |c'_i|) * (\max_{p \in P} \text{Cost}(p)))\}$.

By Lemma 4.4 (2), there exists a corresponding PTPN computation $\pi_\lambda = c_0 \rightarrow \dots \rightarrow c_{n_\lambda}$ in detailed form and δ_λ -form s.t. $c'_i = \text{aptpn}(c_i)$ and $|\text{Cost}(\pi_\lambda) - \text{Cost}(\pi'_\lambda)| \leq n_\lambda * \delta_\lambda * (\max_{0 \leq i \leq n_\lambda} |c'_i|) * (\max_{p \in P} \text{Cost}(p)) \leq \lambda$. Thus we obtain $\text{Cost}(\pi_\lambda) - I' \leq 2\lambda$. Since this holds for every $\lambda > 0$ we get $I \not\leq I'$.

By combining $I' \not\leq I$ with $I \not\leq I'$ we obtain $I = I'$ as required. \square

4.1. **The Running Example (cont.)** We consider the running example from Subsection 2.7 again.

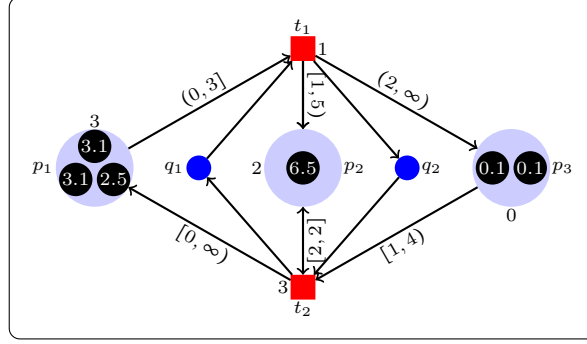


Figure 2: A simple example of a PTPN.

Abstract Markings. Fix $\delta = 0.2$. Then the configuration

$$c = [(p_1, 2.1), (p_1, 1.0), (p_1, 2.85), (p_1, 3.9), \\ (p_2, 1.1), (p_2, 9.1), (p_2, 1.0), (p_2, 9.85), \\ (p_3, 8.1), (p_3, 0.85), (p_3, 2.9), (p_3, 4.9), (p_3, 9.0)]$$

is in δ -form. We have

$$c_1 = \text{aptpn}(c) = \left(q_1, \left(\left[\begin{array}{c} (p_1, 2) \\ , \\ (p_2, 6) \\ , \\ (p_3, 0) \end{array} \right] \left[\begin{array}{c} (p_1, 3) \\ , \\ (p_3, 2) \\ , \\ (p_3, 4) \end{array} \right] \left[\begin{array}{c} (p_1, 1) \\ , \\ (p_2, 1) \\ , \\ (p_3, 6) \end{array} \right] \left[\begin{array}{c} (p_1, 2) \\ , \\ (p_2, 1) \\ , \\ (p_2, 6) \\ , \\ (p_3, 6) \end{array} \right] \right) \right)$$

Note that token ages $> cmax$ are abstracted as $cmax + 1$. Since here $cmax = 5$, all token ages > 5 are abstracted as 6.

Below we describe four examples of abstract computation steps (not the same as in Subsection 2.7).

(i) A type 1 transition from c_1 leads to

$$c_2 = \left(q_1, \left(\left[\begin{array}{c} (p_1, 2) \\ , \\ (p_2, 6) \\ , \\ (p_3, 0) \end{array} \right] \left[\begin{array}{c} (p_1, 3) \\ , \\ (p_3, 2) \\ , \\ (p_3, 4) \end{array} \right] \left[\begin{array}{c} (p_1, 1) \\ , \\ (p_2, 1) \\ , \\ (p_3, 6) \end{array} \right] \left[\begin{array}{c} (p_1, 2) \\ , \\ (p_2, 1) \\ , \\ (p_2, 6) \\ , \\ (p_3, 6) \end{array} \right] \right) \right)$$

(ii) A type 2 transition from c_2 leads to

$$c_3 = \left(q_1, \left(\left[\begin{array}{c} (p_1, 2) \\ , \\ (p_2, 6) \\ , \\ (p_3, 0) \end{array} \right] \left[\begin{array}{c} (p_1, 4) \\ , \\ (p_3, 3) \\ , \\ (p_3, 5) \end{array} \right], \left[\begin{array}{c} (p_1, 1) \\ , \\ (p_2, 1) \\ , \\ (p_3, 6) \end{array} \right] \left[\begin{array}{c} (p_1, 2) \\ , \\ (p_2, 1) \\ , \\ (p_2, 6) \\ , \\ (p_3, 6) \end{array} \right] \right) \right)$$

(iii) A type 3 transition from c_3 leads to

$$c_4 = \left(q_1, \left(\left[\begin{array}{c} (p_1, 3) \\ , \\ (p_2, 6) \\ , \\ (p_3, 1) \end{array} \right] \left[\begin{array}{c} (p_1, 4) \\ , \\ (p_3, 3) \\ , \\ (p_3, 5) \end{array} \right] \left[\begin{array}{c} (p_1, 1) \\ , \\ (p_2, 1) \\ , \\ (p_3, 6) \end{array} \right], \emptyset, \left[\begin{array}{c} (p_1, 3) \\ , \\ (p_2, 2) \\ , \\ (p_2, 6) \\ , \\ (p_3, 6) \end{array} \right] \right) \right)$$

(iv) A type 4 transition from c_3 leads to

$$c_5 = \left(q_1, \left(\left[\begin{array}{c} (p_1, 3) \\ , \\ (p_2, 6) \\ , \\ (p_3, 1) \end{array} \right] \left[\begin{array}{c} (p_1, 4) \\ , \\ (p_3, 3) \\ , \\ (p_3, 5) \end{array} \right] \left[\begin{array}{c} (p_1, 2) \\ , \\ (p_2, 2) \\ , \\ (p_3, 6) \end{array} \right] \left[\begin{array}{c} (p_1, 3) \\ , \\ (p_2, 2) \\ , \\ (p_2, 6) \\ , \\ (p_3, 6) \end{array} \right] \right) \right)$$

Below, we give three concrete timed transitions that correspond to the abstract steps (i)-(iii) described above.

$$[(p_1, 2.1), (p_1, 1.0), (p_1, 2.85), (p_1, 3.9), \\ (p_2, 1.1), (p_2, 9.1), (p_2, 1.0), (p_2, 9.85), \\ (p_3, 8.1), (p_3, 0.85), (p_3, 2.9), (p_3, 4.9), (p_3, 9.0)]$$

$$\xrightarrow{0.01} \text{Time}$$

$$[(p_1, 2.11), (p_1, 1.01), (p_1, 2.86), (p_1, 3.91), \\ (p_2, 1.11), (p_2, 9.11), (p_2, 1.01), (p_2, 9.86), \\ (p_3, 8.11), (p_3, 0.86), (p_3, 2.91), (p_3, 4.91), (p_3, 9.01)]$$

$$\xrightarrow{0.09} \text{Time}$$

$$[(p_1, 2.2), (p_1, 1.1), (p_1, 2.95), (p_1, 4.0), \\ (p_2, 1.2), (p_2, 9.2), (p_2, 1.1), (p_2, 9.95), \\ (p_3, 8.2), (p_3, 0.95), (p_3, 3.0), (p_3, 5.0), (p_3, 9.1)]$$

$$\xrightarrow{0.85} \text{Time}$$

$$[(p_1, 3.05), (p_1, 1.95), (p_1, 3.8), (p_1, 4.85), \\ (p_2, 2.05), (p_2, 10.05), (p_2, 1.95), (p_2, 10.8), \\ (p_3, 9.05), (p_3, 1.8), (p_3, 3.85), (p_3, 5.85), (p_3, 9.95)]$$

A concrete timed transitions that correspond to the abstract step (iv) is the following

$$[(p_1, 2.2), (p_1, 1.1), (p_1, 2.95), (p_1, 4.0), \\ (p_2, 1.2), (p_2, 9.2), (p_2, 1.1), (p_2, 9.95), \\ (p_3, 8.2), (p_3, 0.95), (p_3, 3.0), (p_3, 5.0), (p_3, 9.1)]$$

$$\xrightarrow{0.9} \text{Time}$$

$$[(p_1, 3.1), (p_1, 2.0), (p_1, 3.85), (p_1, 4.9), \\ (p_2, 2.1), (p_2, 10.1), (p_2, 2.0), (p_2, 10.85), \\ (p_3, 9.1), (p_3, 1.85), (p_3, 3.9), (p_3, 5.9), (p_3, 10.0)]$$

5. ABSTRACTING COSTS IN A-PTPN

Given an A-PTPN, the cost-threshold problem is whether there exists a computation $\text{aptpn}(c_{init}) \xrightarrow{\pi} \text{aptpn}(C_{fin})$ s.t. $\text{Cost}(\pi) \leq v$ for a given threshold v .

We now reduce this question to a question about simple coverability in a new model called AC-PTPN. The idea is to encode the cost of the computation into a part of the control-state. For every A-PTPN and cost threshold $v \in \mathbb{N}$ there is a corresponding AC-PTPN that is defined as follows.

For every A-PTPN configuration $(q, b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)$ there are AC-PTPN configurations $((q, y), b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)$ for all integers $y : 0 \leq y \leq v$, where y represents the remaining allowed cost of the computation. We define a finite set of functions

ac_y for $0 \leq y \leq v$ that map A-PTPN configurations to AC-PTPN configurations s.t. $ac_y((q, b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)) = ((q, y), b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)$.

For every discrete transition $t = (q_1, q_2, In, Read, Out) \in T$ with

$$(q_1, b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \longrightarrow_t (q_2, c_{-m'} \dots c_{-1}, c_0, c_1 \dots c_{n'})$$

in the A-PTPN, we have instead

$$((q_1, y), b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \longrightarrow_t ((q_2, y - Cost(t), c_{-m'} \dots c_{-1}, c_0, c_1 \dots c_{n'})$$

in the AC-PTPN for all nonnegative integers y s.t. $v \geq y \geq Cost(t)$. I.e., we deduct the cost of the transition from the remaining allowed cost of the computation.

For every A-PTPN abstract timed transition of the types 1 and 2 $(q_1, \dots) \longrightarrow (q_1, \dots)$ we have corresponding AC-PTPN abstract timed transitions of the same types 1 and 2 where $((q_1, y), \dots) \longrightarrow ((q_1, y), \dots)$ for all $0 \leq y \leq v$. I.e., infinitesimally small delays do not cost anything, and thus no cost is deducted from the remaining allowed cost.

For every A-PTPN abstract timed transition of type 3 (for some $k \in \{0, \dots, n\}$)

$$(q_1, b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \longrightarrow (q_1, b_{-m}^+ \dots b_{-2}^+ b_{-1}^+ b_0 \dots b_k, \emptyset, b_{k+1}^+ \dots b_n^+)$$

we have corresponding AC-PTPN abstract timed transitions of type 3 with

$$((q_1, y), b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \longrightarrow ((q_1, y - z), b_{-m}^+ \dots b_{-2}^+ b_{-1}^+ b_0 \dots b_k, \emptyset, b_{k+1}^+ \dots b_n^+)$$

for all nonnegative integers y s.t. $v \geq y \geq z$ where $z = \sum_{i=-m}^n \sum_{p \in P} |b_i(p)| * Cost(p)$. I.e., we deduct the cost of the timed step (whose length is infinitesimally close to 1) from the remaining allowed cost of the computation.

Similarly, for every A-PTPN abstract timed transition of type 4 (for some $k \in \{0, \dots, n-1\}$) with

$$(q_1, b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \longrightarrow (q_1, b_{-m}^+ \dots b_{-2}^+ b_{-1}^+ b_0 \dots b_k, b_{k+1}^+, b_{k+2}^+ \dots b_n^+)$$

we have corresponding AC-PTPN abstract timed transitions of type 4 with

$$((q_1, y), b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n) \longrightarrow ((q_1, y - z), b_{-m}^+ \dots b_{-2}^+ b_{-1}^+ b_0 \dots b_k, b_{k+1}^+, b_{k+2}^+ \dots b_n^+)$$

for all nonnegative integers y s.t. $v \geq y \geq z$ where $z = \sum_{i=-m}^n \sum_{p \in P} |b_i(p)| * Cost(p)$. I.e., we deduct the cost of the timed step (whose length is infinitesimally close to 1) from the remaining allowed cost of the computation.

The following lemma summarizes the connection between A-PTPN and AC-PTPN.

Lemma 5.1. *There exists an A-PTPN computation $aptpn(c_{init}) \xrightarrow{\pi} aptpn(C_{fin})$ with $Cost(\pi) \leq v$ iff there exists a corresponding AC-PTPN computation $ac_v(aptpn(c_{init})) \xrightarrow{\pi'} \bigcup_{0 \leq y \leq v} ac_y(aptpn(C_{fin}))$*

Proof. Directly from the definition of AC-PTPN. □

Moreover, the connection between A-PTPN and AC-PTPN is constructive, i.e., the cost-threshold problem for A-PTPN can be reduced to the coverability problem for AC-PTPN described in Lemma 5.1.

Note that, unlike A-PTPN, AC-PTPN are not monotone. This is because steps of types 3 and 4 with more tokens on cost-places cost more, and thus cost-constraints might block such transitions from larger configurations. Thus one cannot directly reduce the AC-PTPN coverability problem to an A-PTPN coverability problem.

6. THE ABSTRACT COVERABILITY PROBLEM

We describe a general construction for solving reachability/coverability problems for infinite-state systems under some abstract conditions. In particular, we do *not* assume that the behavior of these systems is fully monotone w.r.t. some well-quasi-order (unlike in many related works [AČJT00, FS01]).

In subsequent sections we will show how this construction can be applied to solve AC-PTPN coverability, and thus the A-PTPN and PTPN cost-threshold problems.

6.1. The Generalized Valk-Jantzen Construction.

Theorem 6.1. (*Valk & Jantzen [VJ85]*) *Given an upward-closed set $V \subseteq \mathbb{N}^k$, the finite set V_{min} of minimal elements of V is effectively computable iff for any vector $\vec{u} \in \mathbb{N}_\omega^k$ the predicate $\vec{u} \downarrow \cap V \neq \emptyset$ is decidable.*

We now show a generalization of this result.

Theorem 6.2. *Let (Ω, \leq) be a set with a decidable well-quasi-order (wqo) \leq , and let $V \subseteq \Omega$ be upward-closed and recursively enumerable. Then the finite set V_{min} of minimal elements of V is effectively constructible if and only if for every finite subset $X \subseteq \Omega$ it is decidable if $V \cap \overline{X \uparrow} \neq \emptyset$ (i.e., if $\exists v \in V. v \notin X \uparrow$).*

Proof. V_{min} is finite, since \leq is a wqo. For the only-if part, since $X \uparrow$ is upward-closed, it suffices to check for each of the finitely many elements of V_{min} if it is not in $X \uparrow$. This is possible, because X is finite and \leq is decidable.

For the if-part, we start with $X = \emptyset$ and keep adding elements to X until $X \uparrow = V$. In every step we do the (by assumption decidable) check if $\exists v \in V. v \notin X \uparrow$. If no, we stop. If yes, we enumerate V and check for every element v if $v \notin X \uparrow$ (this is possible since X is finite and \leq is decidable). Eventually, we will find such a v , add it to the set X , and do the next step. Consider the sequence of elements v_1, v_2, \dots which are added to X in this way. By our construction $v_j \not\leq v_i$ for $j > i$. Thus the sequence is finite, because \leq is a wqo. Therefore the algorithm terminates and the final set X satisfies $\nexists v \in V. v \notin X \uparrow$, i.e., $V \subseteq X \uparrow$. Furthermore, by our construction $X \subseteq V$ and thus $X \uparrow \subseteq V \uparrow = V$. Thus $X \uparrow = V$. Finally, we remove all non-minimal elements from X (this is possible since X is finite and \leq is decidable) and obtain V_{min} . \square

Corollary 6.3. *Let Σ be a finite alphabet and $V \subseteq \Sigma^*$ a recursively enumerable set that is upward-closed w.r.t. the substring ordering \leq . The following three properties are equivalent.*

- (1) *The finite set V_{min} of minimal elements of V is effectively constructible.*
- (2) *For every finite subset $X \subseteq \Sigma^*$ it is decidable if $\exists v \in V. v \notin X \uparrow$.*
- (3) *For every regular language $R \subseteq \Sigma^*$ it is decidable if $R \cap V = \emptyset$.*

Proof. By Higman's Lemma [Hig52], the substring order \leq is a wqo on Σ^* and thus V_{min} is finite. Therefore the equivalence of (1) and (2) follows from Theorem 6.2. Property (1) implies that V is an effectively constructible regular language, which implies property (3). Property (2) is equivalent to checking whether $V \cap \overline{X \uparrow} \neq \emptyset$ and $\overline{X \uparrow}$ is effectively regular because X is finite. Therefore, (3) implies (2) and thus (1). \square

Another interpretation of Corollary 6.3 is the following. An upward-closed set of strings V is always regular, but this regular language is effectively constructible if and only if regular queries to V are decidable.

Note that Theorem 6.2 (and even Corollary 6.3, via an encoding of vectors into strings) imply Theorem 6.1.

6.2. The Abstract Phase Construction. We define some sufficient abstract conditions on infinite-state transition systems under which a general reachability/coverability problem is decidable. Intuitively, we have two different types of transition relations. The first relation is monotone (w.r.t. a given quasi-order) on the whole state space, while the second relation is only defined/enabled on an upward-closed subspace. The quasi-order is not a well quasi-order on the entire space, but only on the subspace. In particular, this is not a well-quasi-ordered transition system in the sense of [AČJT00, FS01], but more general.

We call the following algorithm the *abstract phase construction*, because we divide sequences of transitions into phases, separated by occurrences of transitions of the second kind.

Definition 6.4. We say that a structure $(S, C, \leq, \rightarrow, \rightarrow_A, \rightarrow_B, \text{init}, F)$ satisfies the *abstract phase construction requirements* iff the following conditions hold.

- 1.: S is a (possibly infinite) set of states, $C \subseteq S$ is a finite subset, $\text{init} \in S$ is the initial state and $F \subseteq S$ is a (possibly infinite) set of final states.
- 2.: \leq is a decidable quasi-order on S . Moreover, \leq is a well-quasi-order on the subset $C \uparrow$ (where $C \uparrow = \{s \in S \mid \exists c \in C. s \geq c\}$).
- 3.: $\rightarrow = \rightarrow_A \cup \rightarrow_B$
- 4.: $\rightarrow_A \subseteq S \times S$ is a monotone (w.r.t. \leq) transition relation on S .
- 5.a.: $\rightarrow_B \subseteq C \uparrow \times C \uparrow$ is a monotone (w.r.t. \leq) transition relation on $C \uparrow$.
- 5.b.: For every finite set $X \subseteq C \uparrow$ we have that the finitely many minimal elements of the upward-closed set $\text{Pre}_{\rightarrow_B}(X \uparrow)$ are effectively constructible.
- 6.a.: $\text{Pre}_{\rightarrow_A}^*(F)$ is upward-closed and decidable.
- 6.b.: The finitely many minimal elements of $\text{Pre}_{\rightarrow_A}^*(F) \cap C \uparrow$ are effectively constructible.
- 7.a.: For any finite set $U \subseteq C \uparrow$, the set $\text{Pre}_{\rightarrow_A}^*(U \uparrow)$ is decidable.
- 7.b.: For any finite sets $U, X \subseteq C \uparrow$, it is decidable if $\overline{X \uparrow} \cap \text{Pre}_{\rightarrow_A}^*(U \uparrow) \cap C \uparrow \neq \emptyset$. (In other words, it is decidable if $\exists z \in (\overline{X \uparrow} \cap C \uparrow). z \rightarrow_A^* U \uparrow$.)

Note that in 7.a and 7.b the set $\text{Pre}_{\rightarrow_A}^*(U \uparrow)$ is not necessarily constructible, because \leq is not a well-quasi-order on S . Note also that F is not necessarily upward-closed.

Theorem 6.5. *If $(S, C, \leq, \rightarrow, \rightarrow_A, \rightarrow_B, \text{init}, F)$ satisfies the abstract phase construction requirements of Def. 6.4, then the problem $\text{init} \rightarrow^* F$ is decidable.*

Proof. By Def. 6.4 (cond. 3), we have $\text{init} \rightarrow^* F$ iff (1) $\text{init} \rightarrow_A^* F$, or (2) $\text{init} \rightarrow_A^* (\rightarrow_B \rightarrow_A^*)^+ F$.

Condition (1) can be checked directly, by Def. 6.4 (cond. 6.a).

In order to check condition (2), we first construct a sequence of minimal finite sets $U_k \subseteq C \uparrow$ for $k = 1, 2, \dots$ such that $U_k \uparrow = \{s \in S \mid \exists j : 1 \leq j \leq k. s(\rightarrow_B \rightarrow_A^*)^j F\}$ and show that this sequence converges.

First we construct the minimal finite set $U'_1 \subseteq C \uparrow$ s.t. $U'_1 \uparrow = \text{Pre}_{\rightarrow_A}^*(F) \cap C \uparrow$. This is possible by conditions 6.a and 6.b of Def. 6.4. Then we construct the minimal finite set $U_1 \subseteq C \uparrow$ s.t. $U_1 \uparrow = \text{Pre}_{\rightarrow_B}^*(U'_1 \uparrow)$. This is possible by conditions 5.a and 5.b of Def. 6.4.

For $k = 1, 2, \dots$ we repeat the following steps.

- Given the finite set $U_k \subseteq C \uparrow$, we construct the minimal finite set $U'_{k+1} \subseteq C \uparrow$ s.t. $U'_{k+1} \uparrow = \text{Pre}_{\rightarrow_A}^*(U_k \uparrow) \cap C \uparrow$. This is possible because of Theorem 6.2, which we instantiate as follows. Let $\Omega = C \uparrow$ and $V = \text{Pre}_{\rightarrow_A}^*(U_k \uparrow) \cap C \uparrow$. Using the conditions from Def. 6.4 we have the following: By condition 2, \leq is a decidable well-quasi-order on $C \uparrow$. By condition 4, $V = \text{Pre}_{\rightarrow_A}^*(U_k \uparrow) \cap C \uparrow$ is upward-closed, since \rightarrow_A is monotone. By conditions 7.a and 2, V is decidable, and by condition 7.b the question $\overline{X} \uparrow \cap V \neq \emptyset$ is decidable. Thus, by Theorem 6.2, the finitely many minimal elements of V , i.e., the set U'_{k+1} , are effectively constructible.
- Given U'_{k+1} , we construct the minimal finite set $U''_{k+1} \subseteq C \uparrow$ s.t. $U''_{k+1} \uparrow = \text{Pre}_{\rightarrow_B}(U'_{k+1} \uparrow)$. This is possible by conditions 5.a and 5.b of Def. 6.4.

Then let U_{k+1} be the finite set of minimal elements of $U''_{k+1} \cup U_k$.

The sequence $U_1 \uparrow, U_2 \uparrow, \dots$ is a monotone-increasing sequence of upward-closed subsets of $C \uparrow$, where U_k is the finite set of minimal elements of $U_k \uparrow$. This sequence converges, because \leq is a well-quasi-order on $C \uparrow$ by condition 2 of Def. 6.4. Therefore, we get $U_n \uparrow = U_{n+1} \uparrow$ for some finite index n . Since \leq is only assumed to be a quasi-order (instead of an order) the finite sets of minimal elements U_n and U_{n+1} representing $U_n \uparrow$ and $U_{n+1} \uparrow$ are not necessarily the same. However, we can still check whether $U_n \uparrow = U_{n+1} \uparrow$, and thus detect convergence, because U_n and U_{n+1} are finite and \leq is decidable by condition 2 of Def. 6.4.

We obtain $U_n \uparrow = \{s \in S \mid s(\rightarrow_B \rightarrow_A^*)^* F\}$, because transition \rightarrow_B is only enabled in $C \uparrow$ by Def. 6.4 (cond. 5.a).

Finally, by Def. 6.4 (cond. 7.a) we can do the final check whether $\text{init} \in \text{Pre}_{\rightarrow_A}^*(U_n \uparrow)$ and thus decide condition (2). \square

In the following section we use Theorem 6.5 to solve the optimal cost problem for PTPN. However, it also has many other applications, when used with different instantiations.

Remark 1. Theorem 6.5 can be used to obtain a simple proof of decidability of the coverability problem for Petri nets with one inhibitor arc. Normal Petri net transitions are described by \rightarrow_A , while the inhibited transition is described by \rightarrow_B . (This uses the decidability of the normal Petri net reachability problem [May84] to prove conditions 7.a and 7.b).

A different instantiation could be used to show the decidability of the reachability problem for generalized classes of lossy FIFO-channel systems [AJ96, CFI96] where, e.g., an extra type of transition \rightarrow_B is only enabled when some particular channel is empty.

7. THE MAIN RESULT

In this section we state the main results on the decidability and complexity of the cost-threshold problem.

7.1. The Lower Bound. We show that the cost-threshold problem for PTPN is computationally at least as hard as two other known problems.

- (1) The reachability problem for Petri nets with one inhibitor arc.
- (2) The control-state reachability problem for timed Petri nets (TPN) without any cost model.

The first item establishes a connection between PTPN and Petri nets with one inhibitor arc. This is interesting in itself, but it only yields a relatively weak EXPSPACE lower bound (via the EXPSPACE lower bound for reachability in ordinary Petri nets [Lip76]). The second item shows that the problem is $F_{\omega\omega}$ -hard in the fast growing hierarchy, by using a recent result from [HSS12]. In particular this means that the cost-threshold problem for PTPN is non primitive recursive.

Definition 7.1. Petri nets with one inhibitor arc [Rei08, Bon11] are an extension of Petri nets. They contain a special *inhibitor arc* that prevents a certain transition from firing if a certain place is nonempty.

Formally, a Petri net with an inhibitor arc is described by a tuple $N = (Q, P, T, (p^i, t^i))$ where (p^i, t^i) describes a modified firing rule for transition t^i : it can fire only if p^i is empty.

- Q is a finite set of control-states
- P is a finite set of places
- T is a finite set of transitions. Every transition $t \in T$ has the form $t = (q_1, q_2, I, O)$ where $q_1, q_2 \in Q$ and $I, O \in P^\circ$.

Let $(q, M) \in Q \times P^\circ$ be a configuration of N .

- If $t \in T - \{t^i\}$ then $t = (q_1, q_2, I, O) \in T$ is enabled at configuration (q, M) iff $q = q_1$ and $M \geq I$.
- If $t = t^i$ then $t = (q_1, q_2, I, O) \in T$ is enabled at configuration (q, M) iff $q = q_1$ and $M \geq I$ and $M(p^i) = 0$.

Firing t yields the new configuration (q_2, M') where $M' = M - I + O$.

The reachability problem for Petri nets with one inhibitor arc is decidable [Rei08, Bon11].

Now we describe a polynomial time reduction from the reachability problem for Petri nets with one inhibitor arc to the cost-threshold problem for PTPN.

Lemma 7.2. *Let $(Q, P, T, (p^i, t^i))$ be a Petri net with one inhibitor arc with initial configuration $(q_{init}, [])$ and final configuration $(q_{fin}, [])$.*

One can construct in polynomial time a PTPN $(Q', P', T', Cost)$ with initial configuration $c_{init} = (q_{init}, [])$ and set of final configurations $C_{fin} = \{(q'_{fin}, M) \mid M \in (P \times \mathbb{R}_{\geq 0})^\circ\}$ s.t. $(q_{init}, []) \xrightarrow{} (q_{fin}, [])$ iff $OptCost(c_{init}, C_{fin}) = 0$.*

Proof. Let $(Q, P, T, (p^i, t^i))$ be a Petri net with one inhibitor arc with initial configuration $(q_{init}, [])$ and final configuration $(q_{fin}, [])$. We construct in polynomial time a PTPN $(Q', P', T', Cost)$ with initial configuration $c_{init} = (q_{init}, [])$ and set of final configurations $C_{fin} = \{(q'_{fin}, M) \mid M \in (P \times \mathbb{R}_{\geq 0})^\circ\}$ s.t. $(q_{init}, []) \xrightarrow{*} (q_{fin}, [])$ iff $\inf\{Cost(\pi) \mid c_{init} \xrightarrow{\pi} C_{fin}\} = 0$.

Let $Q' = Q \cup \{q'_{fin}, q^1_{wait}, q^2_{wait}\}$. Let $P' = P \cup \{p^1_{wait}, p^2_{wait}\}$. We define $Cost(p) = 1$ for every $p \in P$, $Cost(p) = 0$ for $p \in P' - P$, and $Cost(t') = 0$ for $t' \in T'$. In order to define the transitions, we need a function that transforms multisets of places into multisets over $P \times Inrv$ by annotating them with time intervals. Let $[p_1, \dots, p_n] \in P^\circ$ and $\mathcal{I} \in Inrv$. Then $annotate([p_1, \dots, p_n], \mathcal{I}) = [(p_1, \mathcal{I}), \dots, (p_n, \mathcal{I})] \in (P \times Inrv)^\circ$.

For every transition $t \in T - \{t^i\}$ with $t = (q_1, q_2, I, O)$ we have a transition $t' = (q_1, q_2, I', O') \in T'$ where $I' = annotate(I \cap (P - \{p^i\})^\circ, [0 : \infty]) + annotate(I \cap \{p^i\}^\circ, [0 : 0])$ and $O' = annotate(O, [0 : 0])$. I.e., the age of the input tokens from p^i must be zero

and for the other input places the age does not matter. The transitions always output tokens of age zero. Instead of $t^i = (q_1^i, q_2^i, I^i, O^i) \in T$ with the inhibitor arc (p^i, t^i) , we have the following transitions in T' : $(q_1^i, q_{wait}^1, \text{annotate}(I^i, [0 : \infty)), [(p_{wait}^1, [0 : 0])])$ and $(q_{wait}^1, q_2^i, [(p_{wait}^1, (0 : 1])], \text{annotate}(O, [0 : 0]))$. This simulates t^i in two steps while enforcing an arbitrarily small, but nonzero, delay. This is because the token on place p_{wait}^1 needs to age from age zero to an age > 0 . If p^i is empty then this yields a faithful simulation of a step of the Petri net with one inhibitor arc. Otherwise, the tokens on p^i will age to a nonzero age and can never be consumed in the future. I.e., a token with nonzero age on p^i will always stay there and indicate an unfaithful simulation.

To reach the set of final configurations C_{fin} , we add the following two transitions: $(q_{fin}, q_{wait}^2, [], [(p_{wait}^2, [0 : 0])])$ and $(q_{wait}^2, q'_{fin}, [(p_{wait}^2, [1 : 1])], [])$. This enforces a delay of exactly one time unit at the end of the computation, i.e., just before reaching C_{fin} .

If $(q_{init}, []) \xrightarrow{*} (q_{fin}, [])$ in the Petri net with one inhibitor arc, then for every $\epsilon > 0$ there is a computation $c_{init} \xrightarrow{\pi} (q_{fin}, [])$ in the PTPN which faithfully simulates it and has $Cost(\pi) < \epsilon$, because the enforced delays can be made arbitrarily small. The final step to $C_{fin} = \{(q'_{fin}, M) \mid M \in (P \times \mathbb{R}_{\geq 0})^\circ\}$ takes one time unit, but costs nothing, because there are no tokens on cost-places. Thus $OptCost(c_{init}, C_{fin}) = \inf\{Cost(\pi) \mid c_{init} \xrightarrow{\pi} C_{fin}\} = 0$.

On the other hand, if $OptCost(c_{init}, C_{fin}) = \inf\{Cost(\pi) \mid c_{init} \xrightarrow{\pi} C_{fin}\} = 0$ then the last step from q_{fin} to q'_{fin} must have taken place with no tokens on places in P . In particular, p^i must have been empty. Therefore, the PTPN did a faithful simulation of a computation $(q_{init}, []) \xrightarrow{*} (q_{fin}, [])$ in the Petri net with one inhibitor arc, i.e., the transition t^i was only taken when p^i was empty. Thus $(q_{init}, []) \xrightarrow{*} (q_{fin}, [])$. \square

Lemma 7.2 implies that even a special case of the cost-threshold problem for PTPN, namely the question $OptCost(c_{init}, C_{fin}) = 0$, is at least as hard as the reachability problem for Petri nets with one inhibitor arc. The exact complexity of this problem is not known, but it is at least as hard as reachability in standard Petri nets (without any inhibitor arc). The exact complexity of the reachability problem for standard Petri nets is not known either, but an EXPSPACE lower bound has been shown in [Lip76].

The connection between the cost-threshold problem for PTPN and the control-state reachability problem for TPN is very easy to show.

Lemma 7.3. *The control-state reachability problem for a timed Petri net (TPN) can be reduced in polynomial time to the question $OptCost(c_{init}, C_{fin}) = 0$ for a PTPN.*

Proof. We construct the PTPN by extending the TPN with a cost function that assigns cost zero to all transitions and places. We define c_{init} to be the initial configuration of the TPN and the set C_{fin} by the target control-state of the TPN. If the target control-state is reachable in the TPN then $OptCost(c_{init}, C_{fin}) = 0$ in the PTPN, otherwise $OptCost(c_{init}, C_{fin})$ is undefined. \square

It has been shown in [HSS12] (Corollary 6.) that the control-state reachability problem for timed Petri nets (TPN) is F_{ω^ω} -hard in the fast growing hierarchy. By Lemma 7.3, we obtain the following theorem.

Theorem 7.4. *The cost-threshold problem for PTPN is F_{ω^ω} -hard in the fast growing hierarchy, and thus non primitive recursive.*

7.2. The Upper Bound. Here we state the main computability result of the paper. Its proof refers to several auxiliary lemmas that will be shown in the following sections.

Theorem 7.5. *Consider a PTPN $\mathcal{N} = (Q, P, T, Cost)$ with initial configuration $c_{init} = (q_{init}, [])$ and set of final configurations $C_{fin} = \{(q_{fin}, M) \mid M \in (P \times \mathbb{R}_{\geq 0})^\circ\}$. Then $OptCost(c_{init}, C_{fin})$ is computable.*

Proof. $OptCost(c_{init}, C_{fin}) = \inf\{Cost(\pi) \mid c_{init} \xrightarrow{\pi} C_{fin}\} = \inf\{Cost(\pi') \mid aptpn(c_{init}) \xrightarrow{\pi'} aptpn(C_{fin})\}$, by Theorem 4.5. Thus it suffices to consider the computations $aptpn(c_{init}) \xrightarrow{\pi'} aptpn(C_{fin})$ of the corresponding A-PTPN. In particular, $OptCost(c_{init}, C_{fin}) \in \mathbb{N}$, provided that it exists.

To compute this value, it suffices to solve the cost-threshold problem for any given threshold $v \in \mathbb{N}$, i.e., to decide if $aptpn(c_{init}) \xrightarrow{\pi} aptpn(C_{fin})$ for some π with $Cost(\pi) \leq v$.

To show this, we first decide if $aptpn(c_{init}) \xrightarrow{\pi} aptpn(C_{fin})$ for any π (i.e., reachability). This can be reduced to the cost-threshold problem by setting all place and transition costs to zero and solving the cost-threshold problem for $v = 0$. If no, then no final state is reachable and we represent this by $\inf\{Cost(\pi) \mid c_{init} \xrightarrow{\pi} C_{fin}\} = \infty$. If yes, then we can find the optimal cost v by solving the cost-threshold problem for threshold $v = 0, 1, 2, 3, \dots$ until the answer is yes.

Now we show how to solve the cost-threshold problem. By Lemma 5.1, this question is equivalent to a reachability problem $ac_v(aptpn(c_{init})) \xrightarrow{*} \bigcup_{0 \leq y \leq v} ac_y(aptpn(C_{fin}))$ in the corresponding AC-PTPN. This reachability problem is decidable by Lemma 7.8. \square

Now we prove the remaining Lemma 7.8. For this we need some auxiliary definitions.

Definition 7.6. We define a partial order \leq^f on AC-PTPN configurations. Given two such configurations $\beta = (q_\beta, (b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n))$ and $\gamma = (q_\gamma, (c_{-m'} \dots c_{-1}, c_0, c_1 \dots c_{n'}))$ we have $\beta \leq^f \gamma$ iff $q_\beta = q_\gamma$ and there exists a strictly monotone function $f : \{-m, \dots, n\} \mapsto \{-m', \dots, n'\}$ where $f(0) = 0$ s.t.

- (1) $c_{f(i)} - b_i \in (P_f \times [cmax + 1])^\circ$, for $-m \leq i \leq n$.
- (2) $c_j \in (P_f \times [cmax + 1])^\circ$, if $\nexists i \in \{-m, \dots, n\}. f(i) = j$.

(Intuitively, γ is obtained from β by adding tokens on free-places, while the tokens on cost-places are unchanged.) In this case, if $\alpha = (q_\beta, (c_{-m'} - b_{f^{-1}(-m')}, \dots, c_{-1} - b_{f^{-1}(-1)}, c_0 - b_0, c_1 - b_{f^{-1}(1)}, \dots, c_{n'} - b_{f^{-1}(n')}))$ then we write $\alpha \oplus \beta = \gamma$. (Note that α is not uniquely defined, because it depends on the choice of the function f . However one such α always exists and only contains tokens on P_f .)

The partial order \leq^c on configurations of AC-PTPN is defined analogously with P_c instead of P_f , i.e., γ is obtained from β by adding tokens on cost-places.

The partial order \leq^{fc} on configurations of AC-PTPN is defined analogously with P instead of P_f , i.e., γ is obtained from β by adding tokens on any places, and $\leq^{fc} = \leq^c \cup \leq^f$.

Lemma 7.7. *The relations of Def. 7.6 have the following properties.*

- (1) \leq^f, \leq^c and \leq^{fc} are decidable quasi-orders on the set of all AC-PTPN configurations.
- (2) For every AC-PTPN configuration c , \leq^f is a well-quasi-order on the set $\{c\}^\uparrow = \{s \mid c \leq^f s\}$ (i.e., here \uparrow denotes the upward-closure w.r.t. \leq^f).
- (3) \leq^{fc} is a well-quasi-order on the set of all AC-PTPN configurations.

Proof.

(1) For the decidability we note that if

$$\beta = (q_\beta, (b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)) \quad \text{and} \quad \gamma = (q_\gamma, (c_{-m'} \dots c_{-1}, c_0, c_1 \dots c_{n'}))$$

then only finitely many strictly monotone functions $f : \{-m, \dots, n\} \mapsto \{-m', \dots, n'\}$ exist with $f(0) = 0$, which need to be explored. Since addition/subtraction/inclusion on finite multisets are computable, the result follows.

Moreover, \leq^f , \leq^c and \leq^{fc} are quasi-orders in the set of all AC-PTPN configurations. Reflexivity holds trivially, and transitivity can easily be shown by composing the respective functions f .

(2) Now we show that \leq^f is a well-quasi-order on the set $\{c\}^\uparrow = \{s \mid c \leq^f s\}$ for every AC-PTPN configuration c . Consider an infinite sequence β_0, β_1, \dots of AC-PTPN configurations where $\beta_i \in \{c\}^\uparrow$ for every i . It follows that there exists an infinite sequence of AC-PTPN configurations $\alpha_0, \alpha_1, \dots$ s.t. α_i only contains tokens on P_f and $\beta_i = c \oplus \alpha_i$ for all i . Since $P_f \times [cmax + 1]$ is finite, multiset-inclusion is a wqo on $(P_f \times [cmax + 1])^\circ$, by Dickson's Lemma [Dic13]. Any AC-PTPN configuration α_i consists of 4 parts: A control-state (out of a finite domain), a finite sequence over $(P_f \times [cmax + 1])^\circ$, an element of $(P_f \times [cmax + 1])^\circ$, and another finite sequence over $(P_f \times [cmax + 1])^\circ$. Thus, by applying Higman's Lemma [Hig52] to each part, we obtain that there must exist indices $i < j$ s.t. $\alpha_i \leq^f \alpha_j$. Therefore $\beta_i = c \oplus \alpha_i \leq^f c \oplus \alpha_j = \beta_j$, and thus \leq^f is a wqo on $\{c\}^\uparrow$.

(3) Now we show that \leq^{fc} is a well-quasi-order on the set of all AC-PTPN configurations. Consider an infinite sequence β_0, β_1, \dots of AC-PTPN configurations. Since $P \times [cmax + 1]$ is finite, multiset-inclusion is a wqo on $(P \times [cmax + 1])^\circ$, by Dickson's Lemma [Dic13]. Any AC-PTPN configuration consists of 4 parts: A control-state (out of a finite domain), a finite sequence over $(P \times [cmax + 1])^\circ$, an element of $(P \times [cmax + 1])^\circ$, and another finite sequence over $(P \times [cmax + 1])^\circ$. Thus, by applying Higman's Lemma [Hig52] to each part, we obtain that there must exist indices $i < j$ s.t. $\beta_i \leq^{fc} \beta_j$. Thus \leq^{fc} is a wqo. \square

Now we prove the required decidability of the AC-PTPN reachability/coverability problem.

Lemma 7.8. *Given an instance of the PTPN cost problem and a threshold $v \in \mathbb{N}$, the reachability question $ac_v(\text{aptpn}(c_{init})) \xrightarrow{*} \bigcup_{0 \leq y \leq v} ac_y(\text{aptpn}(C_{fn}))$ in the corresponding AC-PTPN is decidable.*

Proof. We instantiate a structure $(S, C, \leq, \rightarrow, \rightarrow_A, \rightarrow_B, \text{init}, F)$, show that it satisfies the requirements of Def. 6.4, and then apply Theorem 6.5.

Let S be the set of all AC-PTPN configurations of the form $((q, y), b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)$ where $y \leq v$.

Let C be the set of all AC-PTPN configurations of the form $((q, y), b_{-m'} \dots b_{-1}, b_0, b_1 \dots b_{n'})$ where $y \leq v$, and $b_i \in (P_c \times [cmax + 1])^\circ$ and $\sum_{j=-m'}^{n'} |b_j| \leq v$. In other words, the configurations in C only contain tokens on cost-places and the size of these configurations is limited by v . C is finite, because P_c , $cmax$ and v are finite.

Let $\leq := \leq^f$ of Def. 7.6, i.e., in this proof \uparrow denotes the upward-closure w.r.t. \leq^f . By Lemma 7.7, \leq is decidable, \leq is a quasi-order on S , and \leq is a well-quasi-order on $\{c\}^\uparrow$ for every AC-PTPN configuration c . Therefore \leq^f is a well-quasi-order on C^\uparrow , because C is finite.

Let $init := ac_v(aptpn(c_{init}))$ and $F := \bigcup_{0 \leq y \leq v} ac_y(aptpn(C_{fin}))$. In particular, F is upward-closed w.r.t. \leq^f and w.r.t. \leq^{fc} . Thus conditions 1 and 2 of Def. 6.4 are satisfied.

Let \rightarrow_A be the transition relation induced by the discrete AC-PTPN transitions and the abstract timed AC-PTPN transitions of types 1 and 2. These are monotone w.r.t. \leq^f . Thus condition 4 of Def. 6.4 is satisfied.

Let \rightarrow_B be the transition relation induced by abstract timed AC-PTPN transitions of types 3 and 4. These are monotone w.r.t. \leq^f , but only enabled in $C \uparrow$, because otherwise the cost would be too high. (Remember that every AC-PTPN configuration stores the remaining allowed cost, which must be non-negative.) Moreover, timed AC-PTPN transitions of types 3 and 4 do not change the number or type of the tokens in a configuration, and thus $\rightarrow_B \subseteq C \uparrow \times C \uparrow$. So we have condition 5.a of Def. 6.4. Condition 5.b is satisfied, because there are only finitely many token ages $\leq cmax$ and the number and type of tokens is unchanged.

Condition 3 is satisfied, because $\rightarrow = \rightarrow_A \cup \rightarrow_B$ by the definition of AC-PTPN.

Now we show the conditions 6.a and 6.b. F is upward-closed w.r.t. \leq^{fc} and \rightarrow_A is monotone w.r.t. \leq^{fc} (not only w.r.t. \leq^f). By Lemma 7.7, \leq^{fc} is a decidable wqo on the set of AC-PTPN configurations. Therefore, $Pre_{\rightarrow_A}^*(F)$ is upward-closed w.r.t. \leq^{fc} and effectively constructible (i.e., its finitely many minimal elements w.r.t. \leq^{fc}), because the sequence $Pre_{\rightarrow_A}^{\leq^i}(F)$ for $i = 1, 2, \dots$ converges. Let K be this finite set of minimal (w.r.t. \leq^{fc}) elements of $Pre_{\rightarrow_A}^*(F)$. We obtain condition 6.a., because K is finite and \leq^{fc} is decidable. Moreover, $Pre_{\rightarrow_A}^*(F)$ is also upward-closed w.r.t. \leq^f . The set C is a finite set of AC-PTPN configurations and $C \uparrow$ is the upward-closure of C w.r.t. \leq^f . Therefore $Pre_{\rightarrow_A}^*(F) \cap C \uparrow$ is upward closed w.r.t. \leq^f . Now we show how to construct the finitely many minimal (w.r.t. \leq^f) elements of $Pre_{\rightarrow_A}^*(F) \cap C \uparrow$. For every $k \in K$ let $\alpha(k) := \{k' \mid k' \in C \uparrow, k \leq^c k'\}$, i.e., those configurations which have the right control-state for $C \uparrow$, but whose number of tokens on cost-places is bounded by v , and who are larger (w.r.t. \leq^c) than some base element in K . In particular, $\alpha(k)$ is finite and constructible, because v is finite, and \leq^c and \leq^f are decidable. Note that $\alpha(k)$ can be empty (if k has the wrong control-state or too many tokens on cost-places). Let $K' := \bigcup_{k \in K} \alpha(k)$, which is finite and constructible. We show that $Pre_{\rightarrow_A}^*(F) \cap C \uparrow = K' \uparrow$. Consider the first inclusion. If $x \in K' \uparrow$ then $\exists k' \in K', k \in K. k \leq^c k' \leq^f x, k' \in C \uparrow$. Therefore $k \leq^{fc} x$ and $x \in Pre_{\rightarrow_A}^*(F)$. Also $k' \in C \uparrow$ and $k' \leq^f x$ and thus $x \in C \uparrow$. Now we consider the other inclusion. If $x \in Pre_{\rightarrow_A}^*(F) \cap C \uparrow$ then there is a $k \in K$ s.t. $k \leq^{fc} x$. Moreover, the number of tokens on cost-places in x is bounded by v and the control-state is of the form required by $C \uparrow$, because $x \in C \uparrow$. Since, $k \leq^{fc} x$, the same holds for k and thus there is some $k' \in \alpha(k)$ s.t. $k' \leq^f x$. Therefore $x \in K' \uparrow$. To summarize, K' is the finite set of minimal (w.r.t. \leq^f) elements of $Pre_{\rightarrow_A}^*(F) \cap C \uparrow$ and thus condition 6.b holds.

Conditions 7.a and 7.b are satisfied by Lemma 9.3, (which will be proven in Section 9).

So we have that our instantiation satisfies the abstract phase construction requirements of Def. 6.4. Therefore, Theorem 6.5 yields the decidability of the reachability problem $init \rightarrow^* F$, i.e., $ac_v(aptpn(c_{init})) \xrightarrow{*} \bigcup_{0 \leq y \leq v} ac_y(aptpn(C_{fin}))$. \square

The remaining Lemma 9.3 will be shown in Section 9. Its proof uses the simultaneous-disjoint transfer nets of Section 8.

8. SIMULTANEOUS-DISJOINT-TRANSFER NETS

Definition 8.1. Simultaneous-disjoint-transfer nets (SD-TN) [AMM07] are a subclass of transfer nets [Cia94]. SD-TN subsume ordinary Petri nets. A SD-TN N is described by a tuple $(Q, P, T, Trans)$.

- Q is a finite set of control-states
- P is a finite set of places
- T is a finite set of ordinary transitions. Every transition $t \in T$ has the form $t = (q_1, q_2, I, O)$ where $q_1, q_2 \in Q$ and $I, O \in P^\circ$.
- $Trans$ describes the set of simultaneous-disjoint transfer transitions. Although these transitions can have different control-states and input/output places, they all share the *same transfer* (thus the ‘simultaneous’). The transfer is described by the relation $ST \subseteq P \times P$, which is global for the SD-TN N . Intuitively, for $(p, p') \in ST$, in a transfer every token in p is moved to p' . The transfer transitions in $Trans$ have the form (q_1, q_2, I, O, ST) where $q_1, q_2 \in Q$ are the source and target control-state, $I, O \in P^\circ$ are like in a normal Petri net transition, and $ST \subseteq P \times P$ is the same global transfer relation for all these transitions. For every transfer transition (q_1, q_2, I, O, ST) the following ‘disjointness’ restrictions must be satisfied:

- Let $(sr, tg), (sr', tg') \in ST$. Then either $(sr, tg) = (sr', tg')$ or $|\{sr, sr', tg, tg'\}| = 4$.
Furthermore, $\{sr, tg\} \cap (I \cup O) = \emptyset$.

Let $(q, M) \in Q \times P^\circ$ be a configuration of N . The firing of normal transitions $t \in T$ is defined just as for ordinary Petri nets. A transition $t = (q_1, q_2, I, O) \in T$ is enabled at configuration (q, M) iff $q = q_1$ and $M \geq I$. Firing t yields the new configuration (q_2, M') where $M' = M - I + O$.

A transfer transition $(q_1, q_2, I, O, ST) \in Trans$ is enabled at (q, M) iff $q = q_1$ and $M \geq I$. Firing it yields the new configuration (q_2, M') where

$$\begin{aligned} M'(p) &= M(p) - I(p) + O(p) && \text{if } p \in I \cup O \\ M'(p) &= 0 && \text{if } \exists p'. (p, p') \in ST \\ M'(p) &= M(p) + M(p') && \text{if } (p', p) \in ST \\ M'(p) &= M(p) && \text{otherwise} \end{aligned}$$

The restrictions above ensure that these cases are disjoint. Note that after firing a transfer transition all source places of transfers are empty, since, by the restrictions defined above, a place that is a source of a transfer can neither be the target of another transfer, nor receive any tokens from the output of this transfer transition.

Theorem 8.2. *The reachability problem for SD-TN is decidable, and has the same complexity as the reachability problem for Petri nets with one inhibitor arc.*

Proof. We show that the reachability problem for SD-TN is polynomial-time reducible to the reachability problem for Petri nets with one inhibitor arc (see Def. 7.1), and vice-versa.

For the first direction consider an SD-TN $N = (Q, P, T, Trans)$, with initial configuration (q_0, M_0) and final configuration (q_f, M_f) . We construct a Petri net with one inhibitor arc $N' = (Q', P', T', (p^i, t^i))$ with initial configuration (q'_0, M'_0) and final configuration (q'_f, M'_f) s.t. $(q_0, M_0) \xrightarrow{*} (q_f, M_f)$ in N iff $(q'_0, M'_0) \xrightarrow{*} (q'_f, M'_f)$ in N' .

Let $S := \{sr \mid (sr, tg) \in ST\}$ be the set of source-places of transfers. We add a new place p^i to P' and modify the transitions to obtain the invariant that for all reachable configurations (q, M) in N' we have $M(p^i) = \sum_{sr \in S} M(sr)$. Thus for every transition

$t = (q_1, q_2, I, O) \in T$ in N we have a transition $t' = (q_1, q_2, I', O') \in T'$ in N' where $I'(p^i) = \sum_{sr \in S} I(sr)$ and $O'(p^i) = \sum_{sr \in S} O(sr)$. For all other places p we have $I'(p) = I(p)$ and $O'(p) = O(p)$. This suffices to ensure the invariant, because no place in S is the target of a transfer.

To simulate a transfer transition $(q_1, q_2, I, O, ST) \in Trans$, we add another control-state q^i to Q' , another place $p(q_2)$ to P' and a transition $(q_1, q^i, I', O' + \{p(q_2)\})$ to T' , where I', O' are derived from I, O as above. Moreover, for every pair $(sr, tg) \in ST$ we add a transition $(q^i, q^i, \{sr, p^i\}, \{tg\})$. This allows to simulate the transfer by moving the tokens from the source to the target step-by-step. The transfer is complete when all source places are empty, i.e., when p^i is empty. Finally, we add a transition $t^i = (q^i, q_2, \{p(q_2)\}, \{\})$ and let the inhibitor arc be (p^i, t^i) . I.e., we can only return to q_2 when p^i is empty and the transfer is complete. We return to the correct control-state q_2 for this transition, because the last step is only enabled if there is a token on $p(q_2)$.

So we have $Q' = Q \cup \{q^i\}$, $P' = P \cup \{p^i\} \cup \{p(q) \mid q \in Q\}$ and T' is derived from T as described above. We let $q'_0 = q_0$, $q'_f = q_f$ and $M'_0(p^i) = \sum_{p \in S} M_0(p)$, $M'_f(p^i) = \sum_{p \in S} M_f(p)$ and $M'_0(p) = M_0(p)$ and $M'_f(p) = M_f(p)$ for all places $p \in P$ and $M'_0(p(q)) = M'_f(p(q)) = 0$. Note that, by definition of SD-TN, source-places and target-places of transfers are disjoint. Therefore, the condition on the inhibitor arc enforces that all transfers are done completely (i.e., until p^i is empty, and thus all places in S are empty) and therefore the simulation is faithful. Thus we obtain $(q_0, M_0) \xrightarrow{*} (q_f, M_f)$ in N iff $(q'_0, M'_0) \xrightarrow{*} (q'_f, M'_f)$ in N' , as required. Since the reachability problem for Petri nets with one inhibitor arc is decidable [Rei08, Bon11], we obtain the decidability of the reachability problem for SD-TN.

Now we show the reverse reduction. Consider a Petri net with one inhibitor arc $N = (Q, P, T, (p^i, t^i))$ with initial configuration (q_0, M_0) and final configuration (q_f, M_f) . We construct an SD-TN $N' = (Q', P', T', Trans)$ with initial configuration (q'_0, M'_0) and final configuration (q'_f, M'_f) s.t. $(q_0, M_0) \xrightarrow{*} (q_f, M_f)$ iff $(q'_0, M'_0) \xrightarrow{*} (q'_f, M'_f)$.

Let $Q' = Q$, $P' = P \cup \{p_x\}$ where p_x is a new place, and $T' = T - \{t^i\}$. Let $t^i = (q_1, q_2, I, O)$. In N' , instead of t^i , we have the $Trans = \{(q_1, q_2, I, O, ST)\}$ where $ST = \{(p^i, p_x)\}$. Unlike in N , in N' the inhibited transition can fire even if p^i is nonempty. However, in this case the contents of p^i are moved to p_x where they stay forever. I.e., we can detect an unfaithful simulation by the fact that p_x is nonempty. Let $q'_0 = q_0$, $q'_f = q_f$, $M'_0(p_x) = 0$, $M_f(p_x) = 0$ and $M'_0(p) = M_0(p)$ and $M'_f(p) = M_f(p)$ for all other places p . Thus we get $(q_0, M_0) \xrightarrow{*} (q_f, M_f)$ in N iff $(q'_0, M'_0) \xrightarrow{*} (q'_f, M'_f)$ in N' , as required. Therefore, the reachability problem for SD-TN is polynomially equally hard as the reachability problem for Petri nets with one inhibitor arc. \square

The following corollary shows decidability of a slightly generalized reachability problem for SD-TN, which we will need in the proofs of the following sections.

Corollary 8.3. *Let N be an SD-TN and F a set of SD-TN configurations, which is defined by a boolean combination of finitely many constraints of the following forms.*

- (1) *control-state = q (for some state $q \in Q$)*
- (2) *exactly k tokens on place p (where $k \in \mathbb{N}$)*
- (3) *at least k tokens on place p (where $k \in \mathbb{N}$)*

Then the generalized reachability problem $(q_0, M_0) \xrightarrow{} F$ is decidable.*

Proof. First, the boolean formula can be transformed into disjunctive normal form and solved separately for each clause. Every clause is a conjunction of constraints of the types above. This problem can then be reduced to the basic reachability problem for a modified SD-TN N' and then solved by Theorem 8.2. One introduces a new final control-state q' and adds a construction that allows the transition from F to $(q', \{\})$ if and only if the constraints are satisfied. For type (2) one adds a transition that consumes exactly k tokens from place p . For type (3) one adds a transition that consumes exactly k tokens from place p , followed by a loop which can consume arbitrarily many tokens from place p . We obtain $(q_0, M_0) \xrightarrow{*} F$ in N iff $(q_0, M_0) \xrightarrow{*} (q', \{\})$ in N' . Decidability follows from Theorem 8.2. \square

9. ENCODING AC-PTPN COMPUTATIONS BY SD-TN

In this section, we fix an AC-PTPN \mathcal{N} , described by the tuple $(Q, P, T, Cost)$ and the cost-threshold v . We use the partial order $\leq := \leq^f$ on AC-PTPN configurations; see Def. 7.6. We describe an encoding of the configurations of \mathcal{N} as words over an alphabet Σ . We define $\Sigma := (P \times [cmax + 1]) \cup (Q \times \{y \mid 0 \leq y \leq v\}) \cup \{\#, \$\}$, i.e., the members of Σ are elements of $P \times [cmax + 1]$, the control-states of \mathcal{N} , and the two “separator” symbols $\#$ and $\$$. For a multiset $b = [a_1, \dots, a_n] \in (P \times [cmax + 1])^\circ$, we define the encoding $enc(b)$ to be the word $a_1 \dots a_n \in (P \times [cmax + 1])^*$. For a word $w = b_1 \dots b_n \in ((P \times [cmax + 1])^\circ)^*$, we define $enc(w) := enc(b_n) \# \dots \# enc(b_1)$, i.e., it consists of the reverse concatenation of the encodings of the individual multisets, separated by $\#$. For a marking $M = (w_1, b, w_2)$, we define $enc(M) := enc(w_2) \$ enc(b) \$ enc(w_1)$. In other words, we concatenate the encoding of the components in reverse order: first w_2 then b and finally w_1 , separated by $\$$. Finally for a configuration $c = ((q, y), M)$, we define $enc(c) := (q, y) enc(M)$, i.e., we append the pair (q, y) in front of the encoding of M . The function $enc()$ is extended from configurations to sets of configurations in the standard way. We call a finite automaton \mathcal{A} over Σ a *configuration-automaton* if whenever $w \in L(\mathcal{A})$ then $w = enc(c)$ for some AC-PTPN configuration c .

Lemma 9.1. *Given any finite set C of AC-PTPN configurations, one can construct a configuration-automaton \mathcal{A} s.t. $L(\mathcal{A}) = enc(C \uparrow)$.*

Proof. For every $c \in C$ we construct an automaton \mathcal{A}_c s.t. $L(\mathcal{A}_c) = enc(\{c\} \uparrow)$. Remember that here the upward-closure is taken w.r.t. \leq^f . Let $c = ((q, y), b_{-m} \dots b_{-1}, b_0, b_1 \dots b_n)$. We have $b_i = [b_i^1, \dots, b_i^{j(i)}]$ where $b_i^k \in P \times [cmax + 1]$. Let $\Sigma_1 = P_f \times [cmax + 1]$, i.e., only tokens on free-places can be added in the upward-closure. Let $L_1 = (\Sigma_1^+ \#)^*$. The language L_1 describes encodings of sets of tokens on free-places. Many such sets of tokens can be added during the upward closure w.r.t. \leq^f . Let $w_i = b_i^1 \dots b_i^{j(i)}$, i.e., w_i is an encoding of b_i . Let $L_2 = L_1 w_{-m} \Sigma_1^* \# L_1 w_{-1} \Sigma_1^* \# L_1 \dots w_{-1} \Sigma_1^* (\# L_1)^*$. So L_2 encodes the upward closure w.r.t. \leq^f of the part $b_{-m} \dots b_{-1}$ of the configuration c . Let $L_3 = w_{-0} \Sigma_1^*$. So L_3 encodes the upward closure w.r.t. \leq^f of the part b_0 of the configuration c . Let $L_4 = L_1 w_1 \Sigma_1^* \# L_1 w_2 \Sigma_1^* \# L_1 \dots w_n \Sigma_1^* (\# L_1)^*$. So L_4 encodes the upward closure w.r.t. \leq^f of the part $b_1 \dots b_n$ of the configuration c . Then $L(\mathcal{A}_c) = (q, y) L_2 \$ L_3 \$ L_4 = enc(\{c\} \uparrow)$.

Finally, $L(\mathcal{A}) = \bigcup_{c \in C} L(\mathcal{A}_c) = enc(C \uparrow)$. \square

Lemma 9.2. *We can construct a configuration-automaton \mathcal{A} s.t. $L(\mathcal{A}) = \text{enc}(S)$, where S is the set of all configurations of a given AC-PTPN.*

Proof. Let $\Sigma_1 = \{(q, y) \mid q \in Q, 0 \leq y \leq v\}$ and $\Sigma_2 = P \times [cmax + 1]$. Let $L_1 = \Sigma_2^*$ and $L_2 = L_1(\#\Sigma_2^+)^*$ and $L_3 = L_2\$L_1\L_2 . Then the language of \mathcal{A} is $\Sigma_1 L_3$, which is a regular language over Σ . \square

Lemma 9.3. *Consider an instance of the PTPN cost problem, a given threshold $v \in \mathbb{N}$, and a structure $(S, C, \leq, \rightarrow, \rightarrow_A, \rightarrow_B, \text{init}, F)$, instantiated as in Lemma 7.8.*

Then conditions 7.a and 7.b. of Def. 6.4 are decidable.

Proof.

7.a: Consider a configuration c . We can trivially construct a configuration-automaton \mathcal{A} s.t. $L(\mathcal{A}) = \{\text{enc}(c)\}$. Thus the question $c \in \text{Pre}_{\rightarrow_A}^*(U \uparrow)$ can be decided by applying Lemma 9.4 to \mathcal{A} and U .

7.b: Consider finite sets of AC-PTPN configurations $U, X \subseteq C \uparrow$. By Lemma 9.1, we can construct configuration-automata $\mathcal{A}_1, \mathcal{A}_2$ with $L(\mathcal{A}_1) = \text{enc}(X \uparrow)$ and $L(\mathcal{A}_2) = \text{enc}(C \uparrow)$. Furthermore, by Lemma 9.2, we can construct a configuration-automaton \mathcal{A}_3 with $L(\mathcal{A}_3) = \text{enc}(S)$. Therefore, by elementary operations on finite automata, we can construct a configuration-automaton \mathcal{A}_4 with $L(\mathcal{A}_4) = \overline{L(\mathcal{A}_1)} \cap L(\mathcal{A}_3) \cap L(\mathcal{A}_2)$, and we obtain that $L(\mathcal{A}_4) = \text{enc}(\overline{X \uparrow} \cap C \uparrow)$. Note that the complement operation on words is not the same as the complement operation on the set of AC-PTPN configurations. Thus the need for intersection with \mathcal{A}_3 . The question $\exists z \in (\overline{X \uparrow} \cap C \uparrow). z \rightarrow_A^* U \uparrow$ of 7.b can be decided by applying Lemma 9.4 to \mathcal{A}_4 and U . \square

Lemma 9.4. *Given a configuration-automaton \mathcal{A} , C as in Lemma 7.8, and a finite set $U \subseteq C \uparrow$, it is decidable if there exists some AC-PTPN configuration $c_{\text{init}} \in \text{enc}^{-1}(L(\mathcal{A}))$ s.t. $c_{\text{init}} \rightarrow_A^* U \uparrow$.*

Proof. The idea is to translate the AC-PTPN into an SD-TN which simulates its computation. The automaton \mathcal{A} is also encoded into the SD-TN and runs in parallel. \mathcal{A} outputs an encoding of c_{init} , a nondeterministically chosen initial AC-PTPN configuration from $L(\mathcal{A})$. Since the SD-TN cannot encode sequences, it cannot store the order information in the sequences which are AC-PTPN configurations. Instead this is encoded into the behavior of \mathcal{A} , which outputs parts of the configuration c_{init} ‘just-in-time’ before they are used in the computation (with exceptions; see below). Several abstractions are used to unify groups of tokens with different fractional parts, whenever the PTPN is unable to distinguish them. AC-PTPN timed transitions of types 1 and 2 are encoded as SD-TN transfer transitions, e.g., all tokens with integer age advance to an age with a small fractional part. Since this operation must affect all tokens, it cannot be done by ordinary Petri net transitions, but requires the simultaneous-disjoint transfer of SD-TN. Another complication is that the computation of the AC-PTPN might use tokens (with high fractional part) from c_{init} , which the automaton \mathcal{A} has not yet produced. This is handled by encoding a ‘debt’ on future outputs of \mathcal{A} in special SD-TN places. These debts can later be ‘paid back’ by outputs of \mathcal{A} (but not by tokens created during the computation). At the end, the computation must reach an encoding of a configuration in $U \uparrow$ and all debts must be paid. This yields a reduction to a reachability problem for the constructed SD-TN, which is decidable by Theorem 8.2.

We devote the rest of the section to give the details of the proof.

We show the lemma for the case where U is a singleton $\{c_{\text{fin}}\}$. The result follows from the fact that U is finite and that $U \uparrow = \cup_{c \in U} c \uparrow$. We will define an SD-TN $\mathcal{T} =$

$(Q^{\mathcal{T}}, P^{\mathcal{T}}, T^{\mathcal{T}}, \text{Trans}^{\mathcal{T}})$, a finite set $C_{init}^{\mathcal{T}}$ of (initial) configurations, and a finite set of (final) ω -configurations $C_{final}^{\mathcal{T}}$ such that $\exists c_{init}^{\mathcal{T}} \in C_{init}^{\mathcal{T}}. \exists c_{final}^{\mathcal{T}} \in C_{final}^{\mathcal{T}}. c_{init}^{\mathcal{T}} \xrightarrow{*} c_{final}^{\mathcal{T}}$ in \mathcal{T} iff there is a $c_{init} \in \text{enc}^{-1}(L(\mathcal{A}))$ s.t. $c_{init} \xrightarrow{*_{\mathcal{A}}} U \uparrow$. The result then follows immediately from Theorem 8.2 (and Corollary 8.3). Let $c_{fin} = ((q_{fin}, y_{fin}), M_{fin})$ where M_{fin} is of the form $(b_{-m} \cdots b_{-1}, b_0, b_1 \cdots b_n)$ and b_i is of the form $((p_{i1}, k_{i1}), \dots, (p_{in_i}, k_{in_i}))$ for $i : -m \leq i \leq n$. Let the finite-state automaton \mathcal{A} be of the form $(Q^{\mathcal{A}}, T^{\mathcal{A}}, q_0^{\mathcal{A}}, F^{\mathcal{A}})$ where $Q^{\mathcal{A}}$ is the set of states, $T^{\mathcal{A}}$ is the transition relation, $q_0^{\mathcal{A}}$ is the initial state, and $F^{\mathcal{A}}$ is the set of final states. A transition in $T^{\mathcal{A}}$ is of the form (q_1, a, q_2) where $q_1, q_2 \in Q^{\mathcal{A}}$ and $a \in (P \times [cmax + 1]) \cup (Q \times \{y \mid 0 \leq v \leq y_{init}\}) \cup \{\#, \$\}$. We write $q_1 \xrightarrow{a} q_2$ to denote that $(q_1, a, q_2) \in T^{\mathcal{A}}$. During the operation of \mathcal{T} , we will run the automaton \mathcal{A} “in parallel” with \mathcal{N} . During the course of the simulation, the automaton \mathcal{A} will generate the encoding of a configuration c_{init} . We know that such an encoding consists of a control-state (q_{init}, y_{init}) followed by the encoding of a marking M_{init} , say of the form $(c_{-m'} \cdots c_{-1}, c_0, c_1 \cdots c_{n'})$. Notice that \mathcal{A} may output the encoding of any marking in its language, and therefore the values of m' and n' are not a priori known.

To simplify the presentation, we introduce a number of conventions for the description of \mathcal{T} . First we define a set \mathbf{X} of variables (defined below), where each variable $\mathbf{x} \in \mathbf{X}$ ranges over a finite domain $\text{dom}(\mathbf{x})$. A control-state q then is a mapping that assigns, to each variable $\mathbf{x} \in \mathbf{X}$, a value in $\text{dom}(\mathbf{x})$, i.e., $q(\mathbf{x}) \in \text{dom}(\mathbf{x})$. Consider a state q , variables $\mathbf{x}_1, \dots, \mathbf{x}_k$ where $\mathbf{x}_i \neq \mathbf{x}_j$ if $i \neq j$, and values $\mathbf{v}_1, \dots, \mathbf{v}_k$ where $\mathbf{v}_i \in \text{dom}(\mathbf{x}_i)$ for all $i : 1 \leq i \leq k$. We use $q[\mathbf{x}_1 \leftarrow \mathbf{v}_1, \dots, \mathbf{x}_k \leftarrow \mathbf{v}_k]$ to denote that state q' such that $q'(\mathbf{x}_i) = \mathbf{v}_i$ for all $i : 1 \leq i \leq k$, and $q'(\mathbf{x}) = q(\mathbf{x})$ if $\mathbf{x} \notin \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$. Furthermore, we introduce a set of *transition generators*, where each transition generator θ characterizes a (finite) set $[[\theta]]$ of transitions in \mathcal{T} . A transition generator θ is a tuple $(\text{PreCond}(\theta), \text{PostCond}(\theta), \text{In}(\theta), \text{Out}(\theta))$, where

- $\text{PreCond}(\theta)$ is a set $\{\mathbf{x}_1 = \mathbf{v}_1, \dots, \mathbf{x}_k = \mathbf{v}_k\}$, where $\mathbf{x}_i \in \mathbf{X}$ and $\mathbf{v}_i \in \text{dom}(\mathbf{x}_i)$ for all $i : 1 \leq i \leq k$.
- $\text{PostCond}(\theta)$ is a set $\{\mathbf{x}'_1 \leftarrow \mathbf{v}'_1, \dots, \mathbf{x}'_\ell \leftarrow \mathbf{v}'_\ell\}$, where $\mathbf{x}'_i \in \mathbf{X}$ and $\mathbf{v}'_i \in \text{dom}(\mathbf{x}'_i)$ for all $i : 1 \leq i \leq \ell$.
- $\text{In}(\theta), \text{Out}(\theta) \in (P^{\mathcal{T}})^{\odot}$.

The set $[[\theta]]$ contains all transitions of the form (q_1, q_2, I, O) where

- $q_1(\mathbf{x}_i) = \mathbf{v}_i$ for all $i : 1 \leq i \leq k$.
- $q_2 = q_1[\mathbf{x}'_1 \leftarrow \mathbf{v}'_1, \dots, \mathbf{x}'_\ell \leftarrow \mathbf{v}'_\ell]$.
- $I = \text{In}(\theta)$, and $O = \text{Out}(\theta)$.

In the constructions we will define a set Θ of transition generators and define $T^{\mathcal{T}} := \cup_{\theta \in \Theta} [[\theta]]$.

Below we will define the components $Q^{\mathcal{T}}, P^{\mathcal{T}}, T^{\mathcal{T}}$, and $\text{Trans}^{\mathcal{T}}$ in the definition of \mathcal{T} , together with the set $C_{init}^{\mathcal{T}}$ and configuration $c_{final}^{\mathcal{T}}$.

The set $Q^{\mathcal{T}}$ is, as mentioned above, defined in terms of a set \mathbf{X} of variables. The set \mathbf{X} contains the following elements:

- **Mode** indicates the *mode* of the simulation. More precisely, a computation of \mathcal{T} will consist of three phases, namely an *initialization*, a *simulation*, and a *final phase*. Each phase is divided into a number of sub-phases referred to as *modes*.
- A variable **NState**, with $\text{dom}(\text{NState}) = Q$, that stores the current control-state $q_{\mathcal{N}}$.
- A variable **AState**, with $\text{dom}(\text{AState}) = Q_{\mathcal{A}}$, that stores the current state of \mathcal{A} .
- A variable **FState**(i, j) with $\text{dom}(\text{FState}(i, j)) = \{\text{true}, \text{false}\}$, for each $i : -m \leq i \leq n$ and $1 \leq j \leq n_i$. During the simulation phase, the systems tries to cover all the tokens in

the multisets of M_{fin} . Intuitively, $\mathbf{FState}(i, j)$ is a flag that indicates whether the token $(p_{i,j}, k_{i,j})$ has been covered.

- A variable **CoverFlag** that has one of the values **on** or **off**. The covering of tokens in M_{fin} occurs only during certain phases of the simulation. This is controlled by the value of the variable **CoverFlag**.
- A variable **CoverIndex** with $-m \leq \mathbf{CoverIndex} \leq n$ gives the next multiset whose tokens are to be covered.
- For each $p \in P$ and $k : 0 \leq k \leq cmax + 1$, we have a variable $\mathbf{RDebt}(p, k)$, whose use and domain are explained below. During the simulation, we will need to use tokens that have still not been generated by \mathcal{A} . To account for these tokens, we will implement a “debt scheme” in which tokens are used first, and then “paid back” by tokens that are later generated by \mathcal{A} . The variable $\mathbf{RDebt}(p, k)$ keeps track of the number of tokens (p, k) that have been used on read arcs (the debt on tokens consumed in input operations are managed through specific places described later.) For a place p and a transition t , let $Rmax(p, t)$ be the number of read arcs between p and t . Define $Rmax := \max_{p \in P, t \in T} Rmax(p, t)$. Then, $dom(\mathbf{RDebt}(p, k)) = \{0, \dots, Rmax\}$. The definition of the domain reflects the fact the largest amount of debt that we will generate due to tokens raveling through read arcs is bounded by $Rmax$.

The set $P^{\mathcal{T}}$ contains the following places:

- For each $p \in P$ and $k : 0 \leq k \leq cmax + 1$, the set $P^{\mathcal{T}}$ contains the place $\mathbf{ZeroPlace}(p, k)$. The number of tokens in $\mathbf{ZeroPlace}(p, k) \in P^{\mathcal{T}}$ reflects (although it may be not exactly equal to) the number of tokens in $p \in P$ whose ages have zero fractional parts.
- For each $p \in P$ and $k : 0 \leq k \leq cmax + 1$, the set $P^{\mathcal{T}}$ contains the places $\mathbf{LowPlace}(p, k)$ and $\mathbf{HighPlace}(p, k)$. These places play the same roles as above for tokens with ages that have *low* (close to 0) resp. *high* (close to 1) fractional parts.
- For each $p \in P$ and $0 \leq k \leq cmax + 1$, the set $P^{\mathcal{T}}$ contains the place $\mathbf{InputDebt}(p, k)$. The place represents the amount of debt due to tokens (p, k) traveling through input arcs. There is a priori no bound on the amount of debt on such tokens. Hence, this amount is stored in places (rather than in variables as is the case for read tokens.)

The Set $C_{init}^{\mathcal{T}}$ contains all configurations $(q_{init}^{\mathcal{T}}, M_{init}^{\mathcal{T}})$ satisfying the following conditions:

- $q_{init}^{\mathcal{T}}(\mathbf{Mode}) = \mathbf{Init}$. The initial mode is **Init**.
- $q_{init}^{\mathcal{T}}(\mathbf{AState}) = q_0^{\mathcal{A}}$. The automaton \mathcal{A} is simulated starting from its initial state $q_0^{\mathcal{A}}$.
- $q_{init}^{\mathcal{T}}(\mathbf{FState}(i, j)) = \mathbf{false}$ for all $i : -m \leq i \leq n$ and $1 \leq j \leq n_i$. Initially we have not covered any tokens in M_{fin} .
- $q_{init}^{\mathcal{T}}(\mathbf{RDebt}(p, k)) = 0$ for all $p \in P$ and $k : 0 \leq k \leq cmax + 1$. Initially, we do not have any debts due to read tokens.
- $M_{init}^{\mathcal{T}}(p) = 0$ for all places $p \in P^{\mathcal{T}}$. Initially, all the places of \mathcal{T} are empty.

Notice that the variables **CoverFlag** and **CoverIndex** are not restricted so **CoverFlag** may be **on** or **off** and **CoverIndex** may have any value $-m \leq \mathbf{CoverIndex} \leq n$. Although **NState** is not restricted either, its value will be defined in the first step of the simulation (see below.)

Next, we explain how \mathcal{T} works. In doing that, we also introduce all the members of the set $T^{\mathcal{T}}$.

Initialization In the initialization phase the SD-TN \mathcal{T} reads the initial control-state and then fills in the places according to M_{init} . From the definition of the encoding of a configuration, we know that the automaton \mathcal{A} outputs a pair (q, y) in its first transition. The first move of \mathcal{T} is to store this pair in its control-state. Thus, for each transition $q_1 \xrightarrow{(q,y)} q_2$ in \mathcal{A} where $q \in Q$ and $1 \leq y \leq y_{init}$, the set Θ contains θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Init}, \text{AState} = q_1\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{InitLow}, \text{NState} \leftarrow (q, y), \text{AState} \leftarrow q_2\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \{\text{LowPlace}(p, k)\}$.

In other words, once \mathcal{T} has input the initial control-state, it enters a new mode **InitLow**. In mode **InitLow**, we read the multisets $c_1 \dots c_m$ that represent tokens with low fractional parts. The system starts running \mathcal{A} one step at a time, generating the elements of c_m (that are provided by \mathcal{A} .) When it has finished generating all the tokens in c_m , it moves to the next multiset, generating the multisets one by one in the reverse order finishing with c_1 . We distinguish between two types of such tokens depending on how they will be used in the construction. More precisely, such a token is either *consumed* when firing transitions during the simulation phase or used for *covering* the multisets in M_{fin} . A token (of the form (p, k)), used for consumption, is put in a place **LowPlace** (p, k) . Recall that the relation \rightarrow_A in \mathcal{N} is insensitive to the order of the fractional parts that are small (fractional parts of the tokens in $c_1, \dots, c_{n'}$.) Therefore, tokens in $c_1, \dots, c_{n'}$ that have identical places p and identical integer parts k will all be put in the same place **LowPlace** (p, k) . Formally, for each transition $q_1 \xrightarrow{(p,k)} q_2$ in \mathcal{A} , the set Θ contains θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitLow}, \text{AState} = q_1\}$.
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \{\text{LowPlace}(p, k)\}$.

Each time a new multiset c_j is read from \mathcal{A} , the system decides whether it may be (partially) used for covering the next multiset b_i in M_{fin} . This decision is made by checking the value of the component **CoverFlag**. if **CoverFlag** = **off** then the tokens are only used for consumption during the simulation phase. However, if **CoverFlag** = **on** then the tokens generated by \mathcal{A} can also be used to cover those in M_{fin} . The multiset currently covered is given by the value of the component **CoverIndex**. More precisely, if **CoverIndex** = i for some $i : 1 \leq i \leq n$ then (part of) the multiset c_j that is currently being generated by \mathcal{A} ($j : 1 \leq j \leq n'$) may be used to cover (part of) the multiset b_i . At this stage, we only cover tokens with low fractional parts (those in the multisets b_1, \dots, b_n .) When using tokens for covering, the order on the fractional parts of tokens is relevant. The construction takes into consideration different aspects of this order as follows:

- According to the definition of the ordering \leq^f , the tokens in a given multiset c_j may only be used to cover those in one and the same multiset (say b_i .) This also agrees with the observation that the tokens represented in c_j correspond to tokens in the original TPN that have identical fractional parts (the same applies to b_i .) In fact, if this was not case, then we would be using tokens with identical fractional parts (in c_j) to cover tokens with different fractional parts. Analogously, the multiset b_i can be covered only by the elements of one multiset c_j .

- If $i' < i$ then the fractional parts of the tokens represented by $b_{i'}$ are smaller than those represented by b_i . The same applies to $c_{j'}$ and c_j if $j' < j$. Therefore, if c_j is used to cover b_i and $j' < j$ then $c_{j'}$ should be used to cover $b_{i'}$ for some $i' < i$. Furthermore, a multiset c_j is not necessarily used to cover any multiset, i.e., all the tokens represented by c_j may be used for consumption during the simulation (none of them being used for covering.) Similarly, it can be the case that a given b_i is not covered by any multiset c_j (all its tokens are covered by tokens that are generated during the simulation.) Also, a multiset c_j may only be partially used to cover b_i , i.e., some of its tokens may be used for covering b_i while some are consumed during the simulation. Finally, b_i may only be partially covered by c_j , i.e., some of its tokens are covered by c_j while the rest of tokens are covered by tokens generated during the simulation.

Formally, for each $q_1 \xrightarrow{(p,k)} q_2$ in \mathcal{A} , $1 \leq i \leq n$, $1 \leq j \leq n_i$ with $(p_{i,j}, k_{i,j}) = (p, k)$, we add θ to Θ , where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitLow}, \text{AState} = q_1, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}$.
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2, \text{FState}(i, j) \leftarrow \text{true}\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

The transition sets the flag $\text{FState}(i, j)$ to *true* indicating that the token has now been covered. A transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} indicates that we have finished generating the elements of the current multiset c_j . If $\text{CoverFlag} = \text{on}$ then we have also finished covering tokens in the multiset b_i . Therefore, we decide the next multiset $i' < i$ in which to cover tokens. Recall that not all multisets have to be covered and hence i' need not be equal to $i - 1$ (in fact the multisets $b_{i''}$ for $i' < i'' < i$ will not be covered by the multisets in M_{init} .) We also decide whether to use b_{j-1}^{init} to cover $b_{i'}$ or not. In the former case, we set CoverFlag to *on*, while in the latter case we set CoverFlag equal to *off*. Also, if $\text{CoverFlag} = \text{off}$ then we decide whether to use c_{j-1} for covering b_i or not. We cover these four possibilities by adding the following transition generators to Θ .

(i) For each transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} , $i : 1 \leq i \leq n$, and $i' : -m \leq i' < i$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitLow}, \text{AState} = q_1, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}$.
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2, \text{CoverIndex} \leftarrow i'\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

This is the case where CoverFlag is *on* and continues to be *on*. Notice that no covering takes place if $\text{CoverIndex} \leq 0$, and that the new value of CoverIndex is made strictly smaller than the current one.

(ii) For each transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} , and each $i, i' : 1 \leq i' < i \leq n$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitLow}, \text{AState} = q_1, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}$.
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2, \text{CoverFlag} \leftarrow \text{off}, \text{CoverIndex} \leftarrow i'\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

This is the case where CoverFlag is *on* but it is turned *off* for the next step.

(iii) For each transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} , we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitLow}, \text{AState} = q_1, \text{CoverFlag} = \text{off}\}$.
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2\}$.

- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

This is the case where **CoverFlag** is **off** and continues to be **off**.

(iv) For each transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} , we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitLow}, \text{AState} = q_1, \text{CoverFlag} = \text{off}\}$.
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2, \text{CoverFlag} \leftarrow \text{on}\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

This is the case where **CoverFlag** is **off** but it is turned on for the next step.

The process of generating tokens with low fractional parts continues until we encounter a transition of the form $q_1 \xrightarrow{\$} q_2$ in \mathcal{A} . According to the encoding of markings, this indicates that we have finished generating the elements of the multisets c_1, \dots, c_n . Therefore, we change mode from **InitLow** to **InitZero** (where we scan the multiset b_0 .) We have also to consider changing the variables **CoverFlag** and **CoverIndex** in the same way as above. Therefore, we add the following transition generators:

(i) For each transition $q_1 \xrightarrow{\$} q_2$ in \mathcal{A} , $i : 1 \leq i \leq n$, and $i' : -m \leq i' < i$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitLow}, \text{AState} = q_1, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{InitZero}, \text{AState} \leftarrow q_2, \text{CoverIndex} \leftarrow i'\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

(ii) For each transition $q_1 \xrightarrow{\$} q_2$ in \mathcal{A} , $i : 1 \leq i \leq n$, and $i' : -m \leq i' < i$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitLow}, \text{AState} = q_1, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{InitZero}, \text{AState} \leftarrow q_2, \text{CoverFlag} \leftarrow \text{off}, \text{CoverIndex} \leftarrow i'\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

(iii) For each transition $q_1 \xrightarrow{\$} q_2$ in \mathcal{A} , we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitLow}, \text{AState} = q_1, \text{CoverFlag} = \text{off}\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{InitZero}, \text{AState} \leftarrow q_2\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

(iv) For each transition $q_1 \xrightarrow{\$} q_2$ in \mathcal{A} , we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitLow}, \text{AState} = q_1, \text{CoverFlag} = \text{off}\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{InitZero}, \text{AState} \leftarrow q_2, \text{CoverFlag} \leftarrow \text{on}\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

In **InitZero** the places are filled according to c_0 . The construction is similar to the previous mode. The only differences are that the tokens to be consumed will be put in places **ZeroPlace**(p, k) and that no tokens are covered in M_{fin} .

For each transition $q_1 \xrightarrow{(p,k)} q_2$ in \mathcal{A} , the set Θ contains θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitZero}, \text{AState} = q_1\}$.
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2\}$.
- $\text{In}(\theta) = \emptyset$.

- $\text{Out}(\theta) = \{\text{ZeroPlace}(p, k)\}$.

Since the tokens are not used at this stage for covering the multisets of M_{fin} , no transition generators are added for that purpose. Also, in contrast to tokens belonging to $c_0, \dots, c_{n'}$ we cannot generate tokens belonging to $c_{-m'}, \dots, c_{-1}$ during the initialization phase. The reason is that, in the former case, we only need to keep track of the order of multisets whose tokens are used for covering (the ordering of the fractional parts in tokens used for consumption is not relevant.) Since the number n is given a priori in the construction (the marking M_{fin} is a parameter of the problem), we need only to keep track of tokens belonging to at most n different multisets. This does not hold in the case of the latter tokens, since the order of the multisets to which the tokens belong is relevant also in the case of tokens that will be consumed. Since m' is not a priori bounded, we postpone the generation of these tokens to the simulation phase, where we generate these tokens from \mathcal{A} “on demand”: each time we perform a timed transition, we allow the $\text{HighPlace}(p, k)$ tokens with the highest fractional part to be generated. This construction is made more precise in the description of the simulation phase.

The mode `InitZero` is concluded when we the next transition of \mathcal{A} is labeled with $\$$. This means that we have finished inputting the last multiset b_0 . We now move on to the simulation phase.

For each transition of the form $q_1 \xrightarrow{\$} q_2$ in \mathcal{A} , we add θ to Θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{InitZero}, \text{AState} = q_1\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Sim}, \text{AState} \leftarrow q_2\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

Simulation. The simulation phase consists of simulating a sequence of transitions, each of which is either discrete, of type 1, or of type 2. Each type 2 transition is preceded by at least one type 1 transition. Therefore, from `Sim` we next perform a discrete or a type 1 transition. The (non-deterministic) choice is made using the transition generators θ_1 and θ_2 where:

- $\text{PreCond}(\theta_1) = \{\text{Mode} = \text{Sim}\}$.
- $\text{PostCond}(\theta_1) = \{\text{Mode} \leftarrow \text{Disc}\}$.
- $\text{In}(\theta_1) = \emptyset$.
- $\text{Out}(\theta_1) = \emptyset$.
- $\text{PreCond}(\theta_2) = \{\text{Mode} = \text{Sim}\}$.
- $\text{PostCond}(\theta_2) = \{\text{Mode} \leftarrow \text{Type1.1}\}$.
- $\text{In}(\theta_2) = \emptyset$.
- $\text{Out}(\theta_2) = \emptyset$.

Discrete Transitions. A discrete transition $t = (q_1, q_2, \text{In}, \text{Read}, \text{Out})$ in \mathcal{N} is simulated by a set of transitions in \mathcal{T} . In defining this set, we take into consideration several aspects of the simulation procedure as follows:

- Basically, an interval \mathcal{I} on an arc leading from an input place $p \in \text{In}$ to t induces a set of transitions in $T^{\mathcal{T}}$; namely transitions where there are arcs from places $\text{ZeroPlace}(p, k)$ with $k \in \mathcal{I}$, and from places $\text{LowPlace}(p, k)$ and $\text{HighPlace}(p, k)$ with $(k + \epsilon) \in \mathcal{I}$ for some $\epsilon : 0 < \epsilon < 1$. An analogous construction is made for output and read places of t . Since a read arc does not remove the token from the place, there is both an input arc and output arc to the corresponding transition in \mathcal{T} .

- We recall that the tokens belonging to $c_{-m'}, \dots, c_{-1}$ are not generated during the initial phase, and that these tokens are gradually introduced during the simulation phase. Therefore, a transition may need to be fired before the required $\text{HighPlace}(p, k)$ -tokens have been produced by \mathcal{A} . Such tokens are needed for performing both input and read operations. In order to cover for tokens that are needed for input arcs, we use the set of places $\text{InputDebt}(p, k)$ for $p \in P$ and $0 \leq k \leq c_{\max} + 1$. Then, consuming a token from a place $\text{HighPlace}(p, k)$ may be replaced by putting a token in $\text{InputDebt}(p, k)$. The “debt” can be paid back using tokens that are later generated by \mathcal{A} . When \mathcal{T} terminates, we require all the debt places to be empty (all the debt has been paid back.) Also, we need an analogous (but different) scheme for the read arcs. The difference is due to the fact that the same token may be read several times (without being consumed.) Hence, once the debt has been introduced by the first read operation, it will not be increased by the subsequent read operations. Furthermore, several read operations may be covered by a (single) input operation (a token in a place may be read several times before it is finally consumed through an input operations.) To implement this, we use the variables $\text{RDebt}(p, k)$. Each time a number r of tokens (p, k) are “borrowed” for a read operation, we increase the value of $\text{RDebt}(p, k)$ to r (unless it already has a higher value.) Furthermore, each debt taken on a token (p, k) in an input operation subsumes a debt performed on the same token (p, k) in a read operation. Therefore, the value of an old read debt is decreased by the amount of the input debt taken during the current transition. In a similar manner to input debts, the read debt is later paid back. When \mathcal{T} terminates, we require all $\text{RDebt}(p, k)$ variables to be equal to 0 (all the read debts have been paid back.)
- The transition also changes the control-state of \mathcal{N} .

To formally define the set of transitions in \mathcal{T} induced by discrete transitions, we use a number of definitions. We define $x \dot{-} y := \max(y - x, 0)$. For $k \in \mathbb{N}$ and an interval \mathcal{I} , we write $k \models \mathcal{I}$ to denote that $(k + \epsilon) \in \mathcal{I}$ for some (equivalently all) $\epsilon : 0 < \epsilon < 1$. During the simulation phase, there are two mechanisms for simulating the effect of a token traveling through an (input, read, or output) arc in \mathcal{N} , namely, (i) by letting a token travel from (or to) a corresponding place; and (ii) by “taking debt”. Therefore, we define a number of “transformers” that translate tokens in \mathcal{N} to corresponding ones in \mathcal{T} as follows:

- $\text{ZeroPlaceTransf}(p, \mathcal{I}) := \{\text{ZeroPlace}(p, k) \mid (0 \leq k \leq c_{\max} + 1) \wedge (k \in \mathcal{I})\}$. The \mathcal{N} -token is simulated by a \mathcal{T} -token in a place that represent tokens with zero fractional parts.
- $\text{LowPlaceTransf}(p, \mathcal{I}) := \{\text{LowPlace}(p, k) \mid (0 \leq k \leq c_{\max} + 1) \wedge (k \models \mathcal{I})\}$. The \mathcal{N} -token is simulated by a \mathcal{T} -token in a place that represent tokens with low fractional parts. Notice that we use the relation \models since the fractional part of the token is not zero.
- $\text{HighPlaceTransf}(p, \mathcal{I}) := \{\text{HighPlace}(p, k) \mid (0 \leq k \leq c_{\max} + 1) \wedge (k \models \mathcal{I})\}$. The \mathcal{N} -token is simulated by a \mathcal{T} -token in a place that represent tokens with high fractional parts.
- $\text{InputDebtTransf}(p, \mathcal{I}) := \{\text{InputDebt}(p, k) \mid (0 \leq k \leq c_{\max} + 1) \wedge (k \models \mathcal{I})\}$. The \mathcal{N} -token is simulated by taking debt on an input token.
- $\text{ReadDebtTransf}(p, \mathcal{I}) := \{\text{ReadDebt}(p, k) \mid (0 \leq k \leq c_{\max} + 1) \wedge (k \models \mathcal{I})\}$. The \mathcal{N} -token is simulated by taking debt on a read token.

We extend the transformers to multisets, so for a multiset $b = [(p_1, \mathcal{I}_1), \dots, (p_\ell, \mathcal{I}_\ell)]$, let $\text{ZeroPlaceTransf}(b) := \{[(p_1, k_1), \dots, (p_\ell, k_\ell)] \mid \forall i : 1 \leq i \leq \ell : (p_i, k_i) \in \text{ZeroPlaceTransf}(p_i, \mathcal{I}_i)\}$. We extend the other definitions to multisets analogously.

An RDebt -mapping α is a function that maps each $\text{RDebt}(p, k)$ to a value in $\{0, \dots, R_{\max}\}$. In other words, the function describes the state of the debt on read tokens.

Now, we are ready to define the transitions in \mathcal{T} that are induced by discrete transitions in \mathcal{N} . Each such a transition is induced by a number of objects, namely:

- A transition $t = (q_1, q_2, In, Read, Out) \in T$. This is the transition in \mathcal{N} that is to be simulated in \mathcal{T} .
- The current remaining cost $y : Cost(t) \leq y \leq y_{init}$. The remaining cost has to be at least as large as the cost of the transition to be fired.
- An RDebt-mapping α describing the current debt on read tokens.
- Multisets $In^{Zero}, In^{Low}, In^{High}, In^{Debt}$ where

$$In = In^{Zero} + In^{Low} + In^{High} + In^{Debt}.$$

Intuitively, the tokens traveling through arcs of t are covered by four types of tokens:

- In^{Zero} : \mathcal{N} -tokens that will be transformed into \mathcal{T} -tokens in places encoding ages with zero fractions parts.
- In^{Low} : \mathcal{N} -tokens that will be transformed into \mathcal{T} -tokens in places encoding ages with low fractions parts.
- In^{High} : \mathcal{N} -tokens that will be transformed into \mathcal{T} -tokens in places encoding ages with high fractions parts.
- In^{Debt} : \mathcal{N} -tokens that will be covered by taking debt.
- Multisets $Read^{Zero}, Read^{Low}, Read^{High}, Read^{Debt}$ where

$$Read = Read^{Zero} + Read^{Low} + Read^{High} + Read^{Debt}.$$

The roles of these multisets are similar to the above.

- Multisets $Out^{Zero}, Out^{Low}, Out^{High}$ where

$$Out = Out^{Zero} + Out^{Low} + Out^{High}.$$

The roles of the multisets $Out^{Zero}, Out^{Low}, Out^{High}$ are similar to their counter-parts above.

For each such collection of objects (i.e., for each $t, 0 \leq y \leq y_{init}, \alpha, In^{Zero}, In^{Low}, In^{High}, In^{Debt}, Read^{Zero}, Read^{Low}, Read^{High}, Read^{Debt}, Out^{Zero}, Out^{Low}, Out^{High}$), we add the transition generator θ where:

- $PreCond(\theta) = \{Mode = Disc, NState = (q_1, y)\} \cup \alpha$, i.e., the current mode is *Disc*, the current state of \mathcal{N} is (q_1, y) , and the current debt on read tokens is given by α .
- $PostCond(\theta) = \{Mode \leftarrow Sim, NState \leftarrow (q_2, y - Cost(t))\} \cup \{RDebt(p, k) \leftarrow \max(\alpha \cdot In^{Debt'}, Read^{Debt'})(p, k) \mid (p \in P) \wedge (0 \leq k \leq cmax + 1)\}$, where
 - $In^{Debt'} = InputDebtTransf(In^{Debt})$.
 - $Read^{Debt'} = ReadDebtTransf(Read^{Debt})$.

In other words, we change the mode back to *Sim*, and change the control-state of \mathcal{N} to $(q_2, y - Cost(t))$. The new read debts are defined as follows: We reduce the current debt α using the new debt on input tokens $In^{Debt'}$, then we update the amount again using the new debt $Read^{Debt'}$.

- $In(\theta) = In^{Zero'} + In^{Low'} + In^{High'} + Read^{Zero'} + Read^{Low'} + Read^{High'}$, where
 - $In^{Zero'} = ZeroPlaceTransf(In^{Zero})$.
 - $In^{Low'} = LowPlaceTransf(In^{Low})$.
 - $In^{High'} = HighPlaceTransf(In^{High})$.

- $Read^{Zero'} = ZeroPlaceTransf (Read^{Zero})$.
- $Read^{Low'} = LowPlaceTransf (Read^{Low})$.
- $Read^{High'} = HighPlaceTransf (Read^{High})$.

The multisets In^{Zero} , In^{Low} , In^{High} represent tokens that will be consumed due to input arcs. These tokens are distributed among places according to whether their fractional parts are zero, low, or high. A similar reasoning holds for the multisets $Read^{Zero}$, $Read^{Low}$, $Read^{High}$.

- $Out(\theta) = Out^{Zero'} + Out^{Low'} + Out^{High'} + Out^{Debt'} + Read^{Zero'} + Read^{Low'} + Read^{High'}$, where
 - $Out^{Zero'} = ZeroPlaceTransf (Out^{Zero})$.
 - $Out^{Low'} = LowPlaceTransf (Out^{Low})$.
 - $Out^{High'} = HighPlaceTransf (Out^{High})$.
 - $Out^{Debt'} = HighPlaceTransf (In^{Debt})$.
 - $Read^{Zero'} = ZeroPlaceTransf (Read^{Zero})$.
 - $Read^{Low'} = LowPlaceTransf (Read^{Low})$.
 - $Read^{High'} = HighPlaceTransf (Read^{High})$.

The read multisets are defined as in the previous item. The multisets Out^{Zero} , Out^{Low} , and Out^{High} play the same roles as their input and read counterparts. The multiset $Out^{Debt'}$ represents the increase in the debt on read tokens.

Transitions of Type 1. The simulation of a type 1 transition is started when the mode is **Type1.1**. We recall that a type 1 transition encodes that time passes so that all tokens of integer age in b_0 will now have a positive fractional part, but no tokens reach an integer age. This phase is performed in two steps. First, in **Type1.1** (that is repeated an arbitrary number of times), some of these tokens are used for covering the multisets of M_{fin} in a similar manner to the previous phases. In the second step we change mode to **Type1.2**, at the same time switching on or off the component **CoverFlag** in a similar manner to the initialization phase. In **Type1.2**, the (only set) of transfer transitions encodes the effect of passing time. More precisely all tokens in a place $ZeroPlace(p, k)$ will be moved to the place $LowPlace(p, k)$, for $k : 1 \leq k \leq cmax + 1$. From **Type1.2** the mode will be changed to **Type2.1**.

To describe **Type1.1** formally we add, for each $i : 1 \leq i \leq n$, $j : 1 \leq j \leq n_i$, $p \in P$, $k : 0 \leq k \leq cmax + 1$ with $(p, k) = (p_{i,j}, k_{i,j})$, a transition generator θ where:

- $PreCond(\theta) = \{Mode = Type1.1, CoverFlag = on, CoverIndex = i\}$.
- $PostCond(\theta) = \{FState(i, j) \leftarrow true\}$.
- $In(\theta) = \{ZeroPlace(p, k)\}$.
- $Out(\theta) = \emptyset$.

On switching to **Type1.2**, we change the variables **CoverFlag** and **CoverIndex** in a similar manner to the previous phases. Therefore, we add the following transition generators:

(i) For each $i : 1 \leq i \leq n$, and $i' : -m \leq i' < i$, we add θ where:

- $PreCond(\theta) = \{Mode = Type1.1, CoverFlag = on, CoverIndex = i\}$.
- $PostCond(\theta) = \{Mode \leftarrow Type1.2, CoverFlag \leftarrow off, CoverIndex \leftarrow i'\}$.
- $In(\theta) = \emptyset$.
- $Out(\theta) = \emptyset$.

(ii) For each $i : 1 \leq i \leq n$, and $i' : -m \leq i' < i$, we add θ where

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type1.1}, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Type1.2}, \text{CoverIndex} \leftarrow i'\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

(iii) We add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type1.1}, \text{CoverFlag} = \text{off}\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Type1.2}\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

(iv) We add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type1.1}, \text{CoverFlag} = \text{off}\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Type1.2}, \text{CoverFlag} \leftarrow \text{on}\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

The set of transfer transitions is defined by the transfer transition generator θ

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type1.2}\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Type2.1}\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.
- $ST(\theta) = \{(\text{ZeroPlace}(p, k), \text{LowPlace}(p, k)) \mid (p \in P) \wedge (0 \leq k \leq cmax + 1)\}$.

Transitions of Type 2. Recall that transitions of type 2 encode what happens to tokens with the largest fractional parts when an amount of time passes sufficient for making these ages equal to the next integer (but not larger.) There are two sources of such tokens. The generation of tokens according to these two sources divides the phase into two steps. The first source are tokens that are currently in places of the form $\text{HighPlace}(p, k)$. In **Type2.1**, (some of) these tokens reach the next integer, and are therefore moved to the corresponding places encoding tokens with zero fractional parts. As mentioned above, only some (but not all) of these tokens reach the next integer. The reason is that they are generated during the computation (not by \mathcal{A}), and hence they have arbitrary fractional parts.

The second source are tokens that are provided by the automaton \mathcal{A} (recall that these tokens are not generated during the initialization phase.) In **Type2.2**, we run the automaton \mathcal{A} one step at a time. At each step we generate the next token by taking a transition $q_1 \xrightarrow{(p,k)} q_2$. In fact, such a token (p, k) is used in two ways: either it moves to the place $\text{ZeroPlace}(p, k)$, or it is used to pay the debt we have taken on tokens. The debt is paid back either (i) by removing a token from $\text{InputDebt}(p, k)$; or (ii) by decrementing the value of the variable $\text{RDebt}(p, k)$. A transition $q_1 \xrightarrow{\#} q_2$ means that we have read the last element of the current multiset. This finishes simulating the transitions of type 1 and 2 and the mode is moved back to **Sim** starting another iteration of the simulation phase.

Formally, we describe the movement of tokens in **Type2.1** by adding, for each $p \in P$ and $k : 0 \leq k \leq cmax + 1$, a transition generator θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type2.1}\}$.
- $\text{PostCond}(\theta) = \emptyset$.
- $\text{In}(\theta) = \{\text{HighPlace}(p, k)\}$.
- $\text{Out}(\theta) = \{\text{ZeroPlace}(p, \max(k + 1, cmax + 1))\}$.

At any time, we can change mode from **Type2.1** to **Type2.2**:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type2.1}\}.$
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Type2.2}\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \emptyset.$

We can also move back from **Type2.1** to **Sim** without letting the automaton generate any tokens:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type2.1}\}.$
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Sim}\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \emptyset.$

We simulate **Type2.2** as follows. To describe the movement of tokens places representing tokens with zero fractional parts we add, for each transition $q_1 \xrightarrow{(p,k)} q_2$ in \mathcal{A} , a transition generator θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type2.2}, \text{AState} = q_1\}.$
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \{\text{ZeroPlace}(p, k)\}.$

To describe the payment of debts on input tokens we add, for each transition $q_1 \xrightarrow{(p,k)} q_2$ in \mathcal{A} , a transition generator θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type2.2}, \text{AState} = q_1\}.$
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2\}.$
- $\text{In}(\theta) = \{\text{InputDebt}(p, k)\}.$
- $\text{Out}(\theta) = \emptyset.$

To describe the payment of debts on read tokens we add, for each transition $q_1 \xrightarrow{(p,k)} q_2$ in \mathcal{A} , and $r : 1 \leq r \leq Rmax$, a transition generator θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type2.2}, \text{AState} = q_1, \text{RDebt}(p, k) = r\}.$
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2, \text{RDebt}(p, k) \leftarrow r - 1\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \emptyset.$

As usual, transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} indicates means that we have read the last element of the current multiset. We can now move back to the mode **Sim**, changing the variables **CoverFlag** and **CoverIndex** in a similar manner to the previous phases.

(i) For each transition of the form $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} , $i : 1 \leq i \leq n$, and $i' : -m \leq i' < i$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type2.2}, \text{AState} = q_1, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}.$
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Sim}, \text{AState} \leftarrow q_2, \text{CoverFlag} \leftarrow \text{off}, \text{CoverIndex} \leftarrow i'\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \emptyset.$

(ii) For each transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} , $i : 1 \leq i \leq n$, and $i' : -m \leq i' < i$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type2.2}, \text{AState} = q_1, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}.$
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Sim}, \text{AState} \leftarrow q_2, \text{CoverFlag} \leftarrow \text{on}, \text{CoverIndex} \leftarrow i'\}.$

- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

(iii) For each transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} , we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type2.2}, \text{AState} = q_1, \text{CoverFlag} = \text{off}\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Sim}, \text{AState} \leftarrow q_2\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

(iv) For each transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} , we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Type2.2}, \text{AState} = q_1, \text{CoverFlag} = \text{off}\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Sim}, \text{AState} \leftarrow q_2, \text{CoverFlag} \leftarrow \text{on}\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

The Final Phase. From the simulation mode we can at any time enter the final mode.

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Sim}\}$.
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Final1}\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

The main tasks of the final phase are (i) to cover the multisets in M_{fin} ; and (ii) to continue paying back the *debt* tokens (recall that the debt was partially paid back in the simulation of type 2 transitions.) At the end of the final phase, we expect all tokens in M_{fin} to have been covered and all debt to have been paid back. The final phase consists of two modes. In **Final1** we cover the multisets in M_{fin} using the tokens that have already been generated. In **Final2**, we resume running \mathcal{A} one step at a time. The tokens generated from \mathcal{A} are used both (i) for paying back debt; and (ii) for covering the multisets b_{-1}, \dots, b_{-m} (in that order.)

Formally, we add the following transition generators. First, we continue covering the multisets b_1, \dots, b_n . For each $p \in P$, $1 \leq i \leq n$, and $1 \leq j \leq n_i$ with $(p_{i,j}, k_{i,j}) = (p, k)$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final1}\}$.
- $\text{PostCond}(\theta) = \{\text{FState}(i, j) \leftarrow \text{true}\}$.
- $\text{In}(\theta) = \text{LowPlace}(p, k)$.
- $\text{Out}(\theta) = \emptyset$.

We cover the multiset b_0 by moving tokens from places of the form **ZeroPlace**(p, k). For each $p \in P$ and $1 \leq j \leq n_0$ with $(p_{0,j}, k_{0,j}) = (p, k)$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final1}\}$.
- $\text{PostCond}(\theta) = \{\text{FState}(0, j) \leftarrow \text{true}\}$.
- $\text{In}(\theta) = \text{ZeroPlace}(p, k)$.
- $\text{Out}(\theta) = \emptyset$.

We cover the multisets b_{-1}, \dots, b_{-m} by moving tokens from places of type **HighPlace**(p, k). For each $p \in P$, $-m \leq i \leq -1$, $1 \leq j \leq n_i$ with $(p_{i,j}, k_{i,j}) = (p, k)$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final1}\}$.
- $\text{PostCond}(\theta) = \{\text{FState}(i, j) \leftarrow \text{true}\}$.
- $\text{In}(\theta) = \text{HighPlace}(p, k)$.
- $\text{Out}(\theta) = \emptyset$.

We can change mode to `Final2`:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final1}\}.$
- $\text{PostCond}(\theta) = \{\text{Mode} \leftarrow \text{Final2}\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \emptyset.$

In `Final2`, we start running \mathcal{A} . The tokens can be used for paying input debts. For each transition $q_1 \xrightarrow{(p,k)} q_2$ in \mathcal{A} , we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final2}, \text{AState} = q_1\}.$
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2\}.$
- $\text{In}(\theta) = \{\text{InputDebt}(p, k)\}.$
- $\text{Out}(\theta) = \emptyset.$

The tokens can also be used for paying read debts. For each transition $q_1 \xrightarrow{(p,k)} q_2$ in \mathcal{A} , and $k : 1 \leq r \leq Rmax$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final2}, \text{AState} = q_1, \text{RDebt}(p, k) = r\}.$
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2, \text{RDebt}(p, k) \leftarrow r - 1\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \emptyset.$

Finally, the tokens can be used for covering. For each transition $q_1 \xrightarrow{(p,k)} q_2$ in \mathcal{A} , $i : -m \leq i \leq -1$, $j : 1 \leq j \leq n_i$, $p \in P$, $k : 0 \leq k \leq cmax + 1$ with $(p, k) = (p_{i,j}, k_{i,j})$, we have θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final2}, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}.$
- $\text{PostCond}(\theta) = \{\text{FState}(i, j) \leftarrow \text{true}\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \emptyset.$

A transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} indicates that we have read the last element of the current multiset. We now let \mathcal{A} generate the next multiset. We change the variables `CoverFlag` and `CoverIndex` in a similar manner to the previous phases.

(i) For each transition of the form $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} , $i : -m \leq i \leq -1$, and $i' : -m \leq i' < i$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final2}, \text{AState} = q_1, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}.$
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2, \text{CoverFlag} \leftarrow \text{off}, \text{CoverIndex} \leftarrow i'\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \emptyset.$

(ii) For each transition $q_1 \xrightarrow{\#} q_2$ in \mathcal{A} , $i : 1 \leq i \leq n$, and $i' : -m \leq i' < i$, we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final2}, \text{AState} = q_1, \text{CoverFlag} = \text{on}, \text{CoverIndex} = i\}.$
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2, \text{CoverIndex} \leftarrow i'\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \emptyset.$

(iii) For each transition $q_1 \xrightarrow{\$} q_2$ in \mathcal{A} , we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final2}, \text{AState} = q_1, \text{CoverFlag} = \text{off}\}.$
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2\}.$
- $\text{In}(\theta) = \emptyset.$
- $\text{Out}(\theta) = \emptyset.$

(iv) For each transition $q_1 \xrightarrow{\$} q_2$ in \mathcal{A} , we add θ where:

- $\text{PreCond}(\theta) = \{\text{Mode} = \text{Final2}, \text{AState} = q_1, \text{CoverFlag} = \text{off}\}$.
- $\text{PostCond}(\theta) = \{\text{AState} \leftarrow q_2, \text{CoverFlag} \leftarrow \text{on}\}$.
- $\text{In}(\theta) = \emptyset$.
- $\text{Out}(\theta) = \emptyset$.

The Set C_{final}^T contains all configurations (q_{fin}^T, M_{fin}^T) satisfying the following conditions:

- $q_{fin}^T(\text{NState}) = q_{fin}$. The AC-PTPN is in its final control-state.
- $q_{fin}^T(\text{FState}(i, j)) = \text{true}$ for all $i : -m \leq i \leq n$ and $1 \leq j \leq n_i$. We have covered all tokens in M_{fin} .
- $q_{fin}^T(\text{RDebt}(p, k)) = 0$ for all $p \in P$ and $k : 0 \leq k \leq cmax + 1$. We have paid back all debts on read tokens.
- $M_{fin}(InputDebt(p, k)) = 0$ for all $p \in P$ and $0 \leq k \leq cmax + 1$. We have paid back all debts on input tokens. \square

10. UNDECIDABILITY FOR NEGATIVE COSTS

The cost threshold coverability problem for PTPN is undecidable if negative transition costs are allowed.

In fact, as we see from the proof of Theorem 10.1 below, the undecidability proof holds even if the costs of places are restricted to be non-negative integers, and the costs of transitions are restricted to be non-positive. Moreover, the undecidability proof does not require real-valued clocks, but works even if clock values are natural numbers, i.e., in the discrete-time case.

Theorem 10.1. *The cost threshold problem for PTPN $\mathcal{N} = (Q, P, T, Cost)$ is undecidable.*

Proof. We prove that it is undecidable whether a given control-state can be reached with cost ≤ 0 , through a reduction from the control-state reachability problem for Minsky 2-counter machines [Min67].

We recall that a 2-counter machine M , operating on two counters x_0 and x_1 , is a triple (Q, δ, q_{init}) , where Q is a finite set of *control states*, δ is a finite set of *transitions*, and $q_{init} \in Q$ is the *initial control state*. A transition $r \in \delta$ is a triple (q_1, op, q_2) , where op is of one of the three forms (for $i = 0, 1$): (i) x_i++ which increments the value of x_i by one; (ii) x_i-- which decrements the value of x_i by one; or (iii) $x_i = 0?$ which checks whether the value of x_i is equal to zero. A *configuration* c of M is a triple (q, y_0, y_1) , where $q \in Q$ and $y_0, y_1 \in \mathbb{N}$. The configuration gives the control-state together with the values of the counters x_0 and x_1 . The initial configuration c_{init} is $(q_{init}, 0, 0)$. The operational semantics of M is defined in the standard manner. In the control-state reachability problem, we are given a counter machine M and a (final) control-state q_{fin} , and check whether we can reach a configuration of the form (q_{fin}, y_0, y_1) (for arbitrary y_0 and y_1) from c_{init} .

Given an instance of the control-state reachability problem for 2-counter machines, with $M = (Q, \delta, q_{init})$ and $q_{fin} \in Q$, we derive an instance of the cost threshold coverability problem for a PTPN where we have only non-negative costs on places and only non-positive costs on transitions; and where the threshold vector is given by 0.

We define the PTPN $\mathcal{N} = (Q', P, T, Cost)$ as follows. Each control state $q \in Q$ has a copy in Q' . For each counter x_i we have a place $x_i \in P$ with $C(x_i) = 1$. The number of

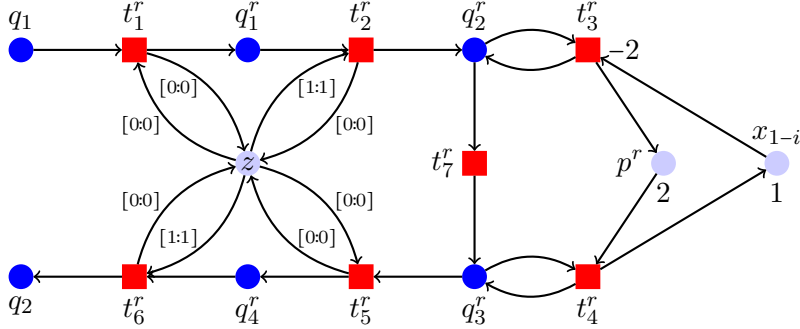


Figure 3: Simulating zero testing in a TPTN. Timed transitions are filled. The cost of a place or transition is shown only if it is different from 0.

tokens in place x_i gives the value of the corresponding counter. To simplify the notation, we adopt the convention that, unless otherwise stated, the time intervals on transitions in our PTPN are $[0 : \infty)$.

Increment and decrement transitions (on a counter x_i) are simulated straightforwardly by changing control state and adding/removing a token from the corresponding place. More precisely, for a transition $r = (q_1, x_i++, q_2) \in \delta$, there is a transition $r \in T$ such that $In(r) = \{q_1\}$, $Out(r) = \{q_2, x_i\}$, and $C(r) = 0$. For a transition $r = (q_1, x_i--, q_2) \in \delta$ there is a transition $r \in T$ such that $In(r) = \{q_1, x_i\}$, $Out(r) = \{q_2\}$, and $C(r) = 0$. The details of simulating a zero testing transition $r = (q_1, x_i = 0?, q_2) \in \delta$ are shown in Figure 3. The main idea is to put a positive cost on the counter places x_i and x_{1-i} . If the system ‘cheats’ and takes the transition from a configuration where the counter value x_i is positive (the corresponding place is not empty), then the transition will impose a cost which cannot be compensated in the remainder of the computation. On the other hand, since the other counter x_{1-i} also has a positive cost, we will also pay an extra (unjustified) price corresponding to the number of tokens in x_{1-i} . Therefore, we use a number of auxiliary places and transitions which make it possible to reimburse unjustified cost for tokens on counter x_{1-i} . The reimbursement is carried out (at most completely, but possibly just partially) by cycling around the tokens in x_{1-i} . For technical reasons (see below), the construction ensures that the only parts of a computation in which time elapses are those that simulate the zero testing of counters (the other parts have zero duration). Concretely, we have seven transitions $t_1^r, \dots, t_7^r \in T$. Furthermore, we have four extra control states q_1^r, \dots, q_4^r , and two extra places p^r, z . The place z is shared among all transitions in T used to simulate transitions in δ that zero-test the value of a counter. The operation of \mathcal{N} starts by putting a single token of age zero in z . The costs are given by $C(t_3^r) = -2$, $C(p^r) = 2$, $C(x_{1-i}) = 1$; while the cost of the other places and transitions are all equal to 0. Intuitively, \mathcal{N} simulates the transition $r = (q_1, x_i = 0?, q_2)$ by first firing the transition t_1^r moving to control state q_1^r which signals the start of the simulation procedure. The transition checks whether the token inside z has still age zero (by removing and putting back a token of age zero). The computation stays in q_1^r and transition t_2^r will be fired after exactly one time unit (this is ensured by checking that age of the token in z is exactly equal to one). Furthermore, the age of the token in z is reset to zero, and the new control state will be q_2^r . The delay will add a cost which is equal to the number of tokens in x_i and x_{1-i} . More precisely, if n is

the number of tokens in place x_{1-i} , then the mentioned unit time delay will add a cost of n . We observe also that t_2^r will move the computation to q_2^r . This enables the next phase which will make it possible to reclaim the (unjustified) cost we have for the tokens in the place x_{1-i} . We can now fire the transition t_3^r m times, where $m \leq n$, thus moving m tokens from x_{1-i} to p^r and gaining $2m$ (i.e., paying $-2m$). Eventually, t_7^r will fire, moving control to q_3^r . From q_3^r , transition t_4^r can fire k times (where $k \leq m$) moving k tokens back to x_{1-i} . The places p^r and x_{1-i} will now contain $m - k$ resp. $n + k - m$ tokens. Eventually, transition t_5^r will fire, moving control to q_4^r and ensuring that no time has elapsed since the firing of t_2^r (and hence no extra costs added due to delays). Finally, transition t_6^r will fire, ensuring a delay of one time unit (thus costing $2(m - k) + (n + k - m) = n + m - k$), moving control to q_2 , and resuming the “normal” simulation of M . The total cost ℓ for the whole sequence of transitions is $\ell = n - 2m + n + m - k = 2n - m - k$. This means $0 \leq \ell$ and that $\ell = 0$ only if $k = m = n$, i.e., all the tokens of x_{1-i} are moved to p^r and back to x_{1-i} . In other words, we can reimburse all the unjustified cost (but not more than that).

This implies correctness of the construction as follows. Suppose that the given instance of the control-state reachability problem has a positive answer. Then, there is a faithful simulation in \mathcal{N} (which will eventually put a token in the place q_F). In particular, each time we perform a transition which tests the value of counter x_i , the corresponding place is indeed empty and hence we pay no cost for it. We can also choose to reimburse all the unjustified cost paid for counter x_{1-i} . Notice that, letting time pass in the parts of the computation that are not part of the simulation of a zero-testing transition, may only contribute with non-negative costs, and that we can always choose not to have any delays in those parts. It follows that the computation has an accumulated cost equal to 0. On the other hand, suppose that the given instance of the control-state reachability problem has a negative answer. Then the only way for \mathcal{N} to put a token in q_F is to ‘cheat’ during the simulation of a zero testing transition (as described above). However, in such a case the accumulated cost for simulating the transition is positive. Since simulations of increment and decrement transitions have zero costs, and simulations of zero testing transitions have non-negative costs, the extra cost paid for cheating cannot be recovered later in the computation. This means that the accumulated cost for the whole computation will be strictly positive, and thus we have a negative instance of the cost threshold coverability problem (with our chosen threshold of 0). \square

11. CONCLUSION AND EXTENSIONS

We have shown that the infimum of the costs to reach a given control-state is computable in priced timed Petri nets with continuous time. This subsumes the corresponding results for less expressive models such as priced timed automata [BBBR07] and priced discrete-timed Petri nets [AM09].

For simplicity of presentation, we have used a one-dimensional cost model, i.e., with a cost $\in \mathbb{R}_{\geq 0}$, but our result on decidability of the Cost-Threshold problem can trivially be generalized to a multidimensional cost model (provided that the cost is linear in the elapsed time). However, in a multidimensional cost model, the Cost-Optimality problem is not defined, because the infimum of the costs does not exist, due to trade-offs between different components. E.g., one can construct a PTPN (and even a priced timed automaton) with

a 2-dimensional cost where the feasible costs are $\{(x, 1 - x) \mid x \in \mathbb{R}_{\geq 0}, 0 < x \leq 1\}$, i.e., with uncountably many incomparable values.

Another simple generalization is to make token storage costs on places dependent on the current control-state, e.g., storing one token on place p for one time unit costs 2 if in control-state q_1 , but 3 if in control-state q_2 . Our constructions can trivially be extended to handle this.

Other extensions are much harder. If the token storage costs are not linear in the elapsed time then the infimum of the costs is not necessarily an integer. In particular, the corner-point abstraction of Section 4 and our translation to an A-PTPN problem would not work. It is an open question how to compute optimal costs in such cases.

Finally, some extensions make the cost-problems undecidable. As shown in Section 10, reachability of a given control-state with cost zero becomes undecidable if general integer costs (including negative costs, i.e., rewards) are allowed. This negative result holds even for the simpler case of discrete-time PTPN, i.e., when clocks values are natural numbers.

If one considers the reachability problem (instead of our control-state reachability problem) then the question is undecidable for TPN [RGdFE99], even without considering costs.

ACKNOWLEDGMENTS

We thank the anonymous referees for their detailed comments.

REFERENCES

- [AČJT00] P.A. Abdulla, K. Čerāns, B. Jonsson, and Y. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160:109–127, 2000.
- [AD94] R. Alur and D. Dill. A theory of timed automata. *TCS*, 126:183–235, 1994.
- [ADMN04] P.A. Abdulla, J. Deneux, P. Mahata, and A. Nylén. Forward reachability analysis of timed Petri nets. In *Proc. FORMATS-FTRTFT'04*, volume 3253 of *LNCS*, pages 343–362. Springer, 2004.
- [AJ96] P.A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
- [AM09] P.A. Abdulla and R. Mayr. Minimal cost reachability/coverability in priced timed Petri nets. In *Proc. of FOSSACS 2009*, volume 5504 of *LNCS*. Springer, 2009.
- [AM11] P.A. Abdulla and R. Mayr. Computing optimal coverability costs in priced timed Petri nets. In *26th Annual IEEE Symposium on Logic in Computer Science (LICS 2011)*. IEEE, 2011.
- [AMM07] P.A. Abdulla, P. Mahata, and R. Mayr. Dense-timed Petri nets: Checking zenoness, token liveness and boundedness. *Logical Methods in Computer Science*, 3(1), 2007.
- [AN01] P.A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *Proc. ICATPN'2001: 22nd Int. Conf. on application and theory of Petri nets*, volume 2075 of *LNCS*, pages 53–70, 2001.
- [AN02] P.A. Abdulla and A. Nylén. Undecidability of LTL for timed Petri nets. In *INFINITY 2002, 4th International Workshop on Verification of Infinite-State Systems*, 2002.
- [ATP01] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In *HSCC*, pages 49–62, 2001.
- [BBBR07] P. Bouyer, T. Brihaye, V. Bruyère, and J. Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, 2007.
- [BCH⁺05] B. Bérard, F. Cassez, S. Haddad, O. Roux, and D. Lime. Comparison of different semantics for time Petri nets. In *Proceedings of ATVA 2005*, volume 3707 of *LNCS*, pages 81–94. Springer, 2005.
- [BFH⁺01] G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC 2001)*, volume 2034 of *LNCS*, pages 147–161. Springer, 2001.

- [BFL⁺08] P. Bouyer, U. Fahrenberg, K. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2008)*, volume 5215 of *LNCS*, pages 33–47. Springer, 2008.
- [BFLM11] P. Bouyer, U. Fahrenberg, K. Larsen, and N. Markey. Quantitative analysis of real-time systems using priced timed automata. *Communications of the ACM (CACM)*, 54(9):78–87, 2011.
- [Bon11] R. Bonnet. The reachability problem for vector addition systems with one zero-test. In Filip Murlak and Piotr Sankowski, editors, *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*, volume 6907 of *LNCS*, pages 145–157. Springer, 2011.
- [Bow96] F. D. J. Bowden. Modelling time in Petri nets. In *Proc. Second Australian-Japan Workshop on Stochastic Models*, 1996.
- [CFI96] G. Cécé, A. Finkel, and S.P. Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.
- [Cia94] G. Ciardo. Petri nets with marking-dependent arc cardinality: Properties and analysis. In *Proc. 15th Int. Conf. Applications and Theory of Petri Nets*, volume 815 of *LNCS*, pages 179–198. Springer-Verlag, 1994.
- [dFERA00] D. de Frutos Escrig, V. Valero Ruiz, and O. Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In *ICATPN 2000*, volume 1825 of *LNCS*, pages 187–206, 2000.
- [Dic13] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Amer. J. Math.*, 35:413–422, 1913.
- [FS01] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *TCS*, 256(1-2):63–92, 2001.
- [Hig52] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7):326–336, 1952.
- [HSS12] S. Haddad, S. Schmitz, and Ph. Schnoebelen. The ordinal-recursive complexity of timed-arc Petri nets, data nets, and other enriched nets. In *27th Annual IEEE Symposium on Logic in Computer Science (LICS 2012)*. IEEE, 2012.
- [JT08] M. Jurdzinski and A. Trivedi. Concavely-priced timed automata. In *Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2008)*, volume 5215 of *LNCS*, pages 48–62. Springer, 2008.
- [LBB⁺01] K.G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proc. 13th Int. Conf. on Computer Aided Verification*, volume 2102 of *LNCS*, 2001.
- [Lip76] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.
- [May84] E. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13:441–460, 1984.
- [Min67] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
- [NW88] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons Inc., New York, 1988.
- [Pet62] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, 1962.
- [Pet77] J.L. Peterson. Petri nets. *Computing Surveys*, 9(3):221–252, 1977.
- [Rei08] K. Reinhardt. Reachability in Petri nets with inhibitor arcs. *Electronic Notes in Theoretical Computer Science*, 223:239–264, 2008.
- [RGdFE99] V. Valero Ruiz, F. Cuartero Gomez, and D. de Frutos Escrig. On non-decidability of reachability for timed-arc Petri nets. In *Proc. 8th Int. Workshop on Petri Net and Performance Models (PNPM'99), 8-10 October 1999, Zaragoza, Spain*, pages 188–196, 1999.
- [Srb08] J. Srba. Comparing the expressiveness of timed automata and timed extensions of Petri nets. In *Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2008)*, volume 5215 of *LNCS*, pages 15–32. Springer, 2008.
- [VJ85] R. Valk and M. Jantzen. The residue of vector sets with applications to decidability problems in Petri nets. *Acta Inf.*, 21, 1985.