PhD Open – Algorithmic verification of infinite-state systems via relaxations

LECTURE NOTES

Michael Blondin

UNIVERSITÉ DE SHERBROOKE

October 31, 2022

Preface

This document was written for a PhD Open lecture series taking place at the University of Warsaw in 2022 from October 26 to 28. They were mostly assembled from lecture notes of the course IGL752 (Université de Sherbrooke) and some of my publications [BFHH17, BH17, Blo20, BHO21, BE22]. In some parts, such as the vast majority of Chapters 2 and 3, large portions of these papers were copied verbatim. Consequently, these notes should only be used for this teaching purpose!

Contents

1 Petri nets

1.1 Formalism

Definition 1. A *Petri net* is a triple $\mathcal{N} = (P, T, F)$ where

- P is a finite set (*places*);
- T is a finite set disjoint from P (*transitions*);
- $F\colon ((P \times T) \cup (T \times P)) \to \mathbb{N}$ (*flow function*).

Example.

The following Petri net is depicted in Figure 1.1:

$$P = \{p, q\}, \qquad T = \{s, t\},$$

$$\begin{aligned} F(p, s) &= 1, & F(p, t) &= 0, \\ F(q, s) &= 0, & F(q, t) &= 3, \end{aligned}$$

$$\begin{aligned} F(s, p) &= 0, & F(t, p) &= 4, \\ F(s, q) &= 2, & F(t, q) &= 1. \end{aligned}$$

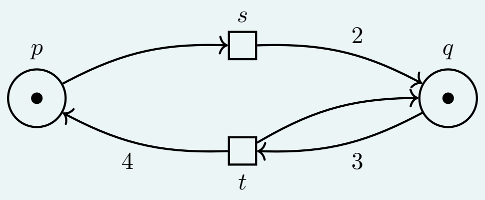


Figure 1.1: An example of a Petri net marked by $(1, 1)$. Places and transitions are respectively represented by circles and squares. Arcs represent the flow function.

A *marking* is a vector $\boldsymbol{m} \in \mathbb{N}^P$ that associates an amount of tokens $\boldsymbol{m}(p)$ to each place p. A transition $t \in T$ is *firable* in $\boldsymbol{m}$ if $\boldsymbol{m}(p) \geq F(p, t)$ for every place $p \in P$. If t is firable, then

$$\boldsymbol{m} \xrightarrow{t} \boldsymbol{m}' \text{ where } \boldsymbol{m}'(p) := \boldsymbol{m}(p) - F(p, t) + F(t, p) \text{ for all } p \in P.$$

We write $\boldsymbol{m} \longrightarrow \boldsymbol{m}'$ if there exists $t \in T$ such that $\boldsymbol{m} \xrightarrow{t} \boldsymbol{m}'$, and $\boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'$ if $\boldsymbol{m} = \boldsymbol{m}'$ or there exists a sequence of markings and transitions such that:

$$\boldsymbol{m} = \boldsymbol{m}_0 \xrightarrow{t_1} \boldsymbol{m}_1 \xrightarrow{t_2} \cdots \xrightarrow{t_k} \boldsymbol{m}_k = \boldsymbol{m}'.$$

Example.

In the Petri net from Figure 1.1, we have $(1, 1) \xrightarrow{*} (3, 3)$ since

$$(1, 1) \xrightarrow{s} (0, 3) \xrightarrow{t} (4, 1) \xrightarrow{s} (3, 3).$$

The *successors* and *predecessors* of a marking $\boldsymbol{m}$ are respectively defined by:

$$\text{Post}^*(\boldsymbol{m}) = \{\boldsymbol{m}' \in \mathbb{N}^P : \boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'\},$$
$$\text{Pre}^*(\boldsymbol{m}) = \{\boldsymbol{m}' \in \mathbb{N}^P : \boldsymbol{m}' \xrightarrow{*} \boldsymbol{m}\}.$$

We write $\boldsymbol{m} \geq \boldsymbol{m}'$ if $\boldsymbol{m}(p) \geq \boldsymbol{m}'(p)$ for every place $p \in P$.

Example.

We have $(1, 2, 3) \geq (1, 1, 0)$, but $(1, 2, 3) \not\geq (0, 1, 4)$.

1.2 Modeling systems

We provide examples from the literature that illustrate how Petri nets can model systems that can be formally analyzed.

1.2.1 Program synthesis

The authors of [FMW+17] and [GJJ+20] have recently employed the Petri net reachability problem for automated program synthesis. In their setting, one is given an API containing hundreds or thousands of functions, together with a type signature and a number of test cases. The goal is to automatically synthesize a loop-free program using functions from the API that respects the specified type signature and satisfies the given test cases.

java.awt.geom
`new AffineTransformation()`
`Shape Shape.createTransformedShape(AffineTransformation)`
`String Point2D.ToString()`
`double Point2D.getX()`
`double Point2D.getY()`
`void AffineTransformation.setToRotation(double, double, double)`
`void AffineTransformation.invert()`
`Area Area.createTransformedArea(AffineTransformation)`

Figure 1.2: A small sample of methods from library `java.awt.geom`.

Let us illustrate the approach with an example from [FMW+17]. Suppose we have access to library `java.awt.geom`, and we wish to synthesize a function `rotate` with type signature

```
Area rotate(Area object, Point2D point, double angle).
```

Naturally, the function should rotate the supplied `Area` around `point` by `angle` degrees. We assume the `java.awt.geom` library is sufficient for this task in that it contains the functions needed to synthesize the method. Figure 1.2 presents an excerpt of functions contained in the API.

The authors of [FMW+17] suggest to view an API as a Petri net whose places correspond to types and transitions correspond to API functions which, informally speaking, consume input types and produce an output type. Figure 1.3 illustrates the Petri net corresponding to the excerpt of API functions listed in Figure 1.2. To synthesize the `rotate` function above, we start with tokens in the places corresponding to the input parameters of our function.

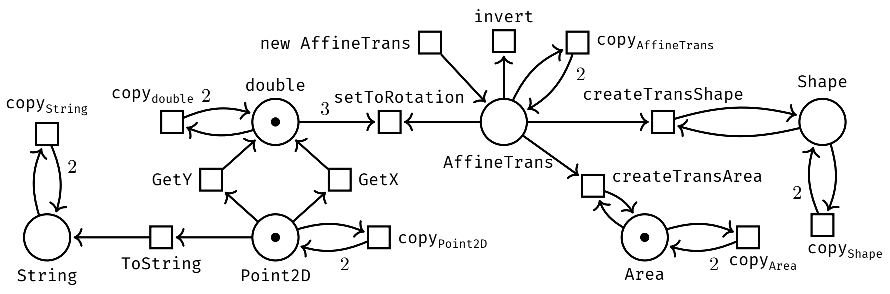


Figure 1.3: A Petri net modelling the API of Figure 1.2.

Thus, in Figure 1.3 we have one token in each of the places corresponding to `Area`, `Point2D` and `double`. The goal is then to reach a marking with a single token in the place corresponding to the return type. In our example, we aim for one token in `Area`, and no token in any other place.

This corresponds to invoking a sequence of functions that "use up" all input parameters, and finally return the correct type. To allow reuse of variables, additional "copy" transitions are introduced for each place; they take one token from a place and put two tokens back. If the target marking is reachable, then the witnessing path corresponds to a partial sketch of a program.

For example, the path

$$\begin{aligned}&\text{copy}_{\text{Point2D}} \to \text{GetY} \to \text{GetX} \to \text{new AffineTransformation} \to \\ &\qquad\qquad \text{copy}_{\text{AffineTransformation}} \to \text{setToRotation} \to \text{createTransformedArea}\end{aligned}$$

tells us which functions to apply, and in which order to apply them. Since Petri nets do not store information about the identity of tokens, when we have multiple objects of the same type, we do not know which to supply as an argument to which function. This can be figured out by a separate process involving SAT solving (see [FMW+17] for more details).

As discussed in [FMW+17], finding short paths of the Petri net is a natural goal. Indeed, since short programs are easier to test, there are fewer possibilities for the arguments of each function, and it is easier for humans to verify that the synthesized program has the desired functionality.

1.2.2 Concurrent program analysis

Perhaps most prominently, Petri nets have been used to model and analyze concurrent processes. Let us give a simple example illustrating how Petri nets can be used in order to detect race conditions in concurrent programs. Consider function `fun()` of Figure 1.4 in which `s` is a global shared Boolean variable. If there is a single thread running `fun()`, then the condition of the `if`-statement in Line 3 never evaluates to true and an error cannot occur. However, if there are two independently interleaved threads running `fun()`, it is possible that one thread reaches Line 3 whilst `s` is set to 1, which means an error could occur.

```
0 def fun():
1   s = 1
2   s = 0
3   if s == 1: raise Err()
```

Figure 1.4: Simple program with a potential race condition.

In more technical terms, we consider non-recursive Boolean programs in which an unbounded number of identical programs run in parallel. The authors of [GS92] showed that verifying safety properties of such concurrent programs amounts to verifying some Petri net.

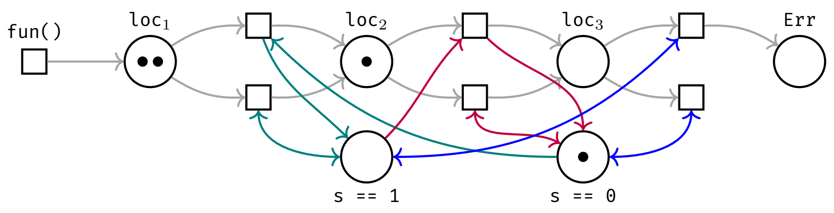


Figure 1.5: The Petri net modeling the program of Figure 1.4. A token in place `loc`$_i$ represents a thread at program location i, and a token in place `s == `b indicates that variable `s` has value b. Transition `fun()` spawns threads. A bidirectional arc $p \leftrightarrow t$ abbreviates two arcs: $p \rightarrow t$ and $t \rightarrow p$. Colors are only meant to help readability.

The Petri net obtained by applying the approach of [GS92] to the program from Figure 1.4 is depicted in Figure 1.5. The places on the top of the Petri net correspond to the program locations of Figure 1.4. Tokens in each of the places on the top count the number of threads which are currently at the respective program location, which is a form of counter abstraction. At any time, transition `fun()` can add tokens to `loc`$_1$, reflecting that a new thread executing `fun()` can be spawned at any point in time arbitrarily often. The two places on the bottom encode the state of the Boolean variable `s` which is updated whenever a transition moves tokens from `loc`$_1$ to `loc`$_2$, or from `loc`$_2$ to `loc`$_3$. Determining whether an error can occur then reduces to deciding whether at least one token can be procued in place `Err`, *i.e.*, whether there is an interleaving in which at least one thread produces an error.

1.2.3 Further applications

The authors of [FKP14] show how proofs involving counting arguments, which can, for instance, naturally prove properties of concurrent programs with recursive procedures, can automatically be synthesized by a reduction to the so-called Petri net reachability problem. The authors of [GM12] propose a model for reasoning about finite-data asynchronous programs. They show that proving liveness properties of such programs in their model is inter-reducible with the Petri net reachability problem.

In a broader context, it was shown that various verification problems for population protocols, a formal model of sensor networks, reduce to the Petri net reachability problem [EGLM17]. The authors of [DLV19] develop a method that allows for verifying rich models of data-driven workflows by a reduction to the so-called coverability problem for Petri nets. See also survey [Mur89] for further classical application areas of Petri nets and their extensions.

1.3 Verification

Let us define the aforementioned verification problems:

REACHABILITY PROBLEM

INPUT: Petri net $\mathcal{N} = (P, T, F)$ and markings $\boldsymbol{m}, \boldsymbol{m}' \in \mathbb{N}^P$

QUESTION: $\boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'$?

COVERABILITY PROBLEM

INPUT: Petri net $\mathcal{N} = (P, T, F)$ and markings $\boldsymbol{m}, \boldsymbol{m}' \in \mathbb{N}^P$

QUESTION: is there $\boldsymbol{m}'' \in \mathbb{N}^P$ such that $\boldsymbol{m} \xrightarrow{*} \boldsymbol{m}''$ and $\boldsymbol{m}'' \geq \boldsymbol{m}'$?

The above two problems are decidable; *i.e.* they can both be solved with an algorithm. However, their complexity is prohibitive: non primitive recursive [ST77, May81, Kos82, Lam92, Ler12, CLL$^+$19, Ler21, CO21] and EXPSPACE-complete [Lip76, Rac78], respectively. Nonetheless, they can both be solved in practice (as we shall see).

In this chapter, we focus on the coverability problem. We say that a marking $\boldsymbol{m}'$ is *coverable* from a marking $\boldsymbol{m}$ if there exists a marking $\boldsymbol{m}'' \in \mathbb{N}^P$ such that

$$\boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'' \text{ and } \boldsymbol{m}'' \geq \boldsymbol{m}'.$$

We say that $\boldsymbol{m}$ *can cover* $\boldsymbol{m}'$ if $\boldsymbol{m}'$ is coverable from $\boldsymbol{m}$.

1.3.1 Coverability graphs

Let us consider the Petri net depicted in Figure 1.6. Let us determine whether $\boldsymbol{m} = (1, 1)$ can cover a marking $\boldsymbol{m}'$. We could attempt constructing Post$^*(1, 1)$ as a reachability graph, as depicted on the left hand-side of Figure 1.7. However, this graph is infinite and it would *a priori* be impossible to determine at which point we may stop constructing this graph.

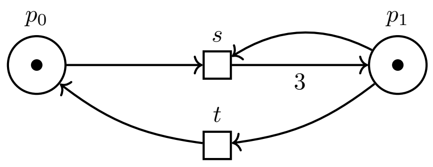


Figure 1.6: Another example of a Petri net.

In order to overcome this problem, we introduce the notion of coverability graphs. We extend $\mathbb{N}$ with a maximal element ω. More formally, we define the

$$\begin{tikzcd}
 & (1,1) \arrow[ld, "s"'] \arrow[rd, "t"] & & \\
(0,3) \arrow[rd, "t"] & & (2,0) & \\
 & (1,2) \arrow[ld, "s"'] \arrow[rd, "t"] & & \\
(0,4) \arrow[rd, dotted, no head] & & (2,1) \arrow[ld, "s"'] \arrow[rd, "t"] & \\
 & (1,3) \arrow[ld, dotted, no head] \arrow[rd, dotted, no head] & & (3,0) \\
 & & &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & (1,1) \arrow[ld, "s"'] \arrow[rd, "t"] & \\
(0,3) \arrow[rd, "t"] & & (2,0) \\
 & (1,\omega) \arrow[ld, "s"'] \arrow[rd, "t"] & \\
(0,\omega) \arrow[rr, "t"'] & & (\omega,\omega) \arrow[loop right, "{s,t}"]
\end{tikzcd}$$

Figure 1.7: Reachability graph (left) and coverability graph (right) of the Petri net from Figure 1.6 starting from marking $(1,1)$.

set $\mathbb{N}_\omega \coloneqq \mathbb{N} \cup \{\omega\}$ where

$$\begin{aligned}
n + \omega &\coloneqq \omega && \text{for every } n \in \mathbb{N}_\omega, \\
\omega - n &\coloneqq \omega && \text{for every } n \in \mathbb{N}, \\
\omega &> n && \text{for every } n \in \mathbb{N}.
\end{aligned}$$

An *extended marking* is a vector $\boldsymbol{m} \in \mathbb{N}_\omega^P$. The notions of transition firability and firing are naturally extended to such markings.

Let us reconsider the reachability graph depicted on the left of Figure 1.7. When we reach marking $(1,2)$, we observe that

$$(1,1) \xrightarrow{st} (1,2) \text{ and } (1,2) \geq (1,1).$$

Iterating the sequence st, we may increase the second place arbitrarily. Thus, we replace this marking by the extended marking $(1,\omega)$, which we call its *acceleration*. The symbol "ω" does not indicate that all values are reachable, but merely that the place can hold as many tokens as desired. Thus, when a marking is added, we can inspect its ancestors and proceed to such accelerations. We obtain a *finite* graph that represents all coverable markings; *e.g.* see the one depicted on the right hand-side of Figure 1.7.

The procedure to compute coverability graphs is described in Algorithm 1. The following proposition explains how to determine whether a given marking $\boldsymbol{m}'$ is coverable, using a coverability graph:

Proposition 1. *Let $\mathcal{N} = (P,T,F)$ be a Petri net, and let $\boldsymbol{m}, \boldsymbol{m}' \in \mathbb{N}^P$. Let G be a coverability graph constructed by* `cover`$(\mathcal{N}, \boldsymbol{m})$. *Marking $\boldsymbol{m}'$ is coverable from $\boldsymbol{m}$ iff G has a marking $\boldsymbol{m}''$ such that $\boldsymbol{m}'' \geq \boldsymbol{m}'$.*

Algorithm 1: Algorithm to compute coverability graphs.

```
Input: Petri net $\mathcal{N} = (P, T, F)$ and $\boldsymbol{m} \in \mathbb{N}^P$
Output: Coverability graph from $\boldsymbol{m}$

cover($\mathcal{N}, \boldsymbol{m}$):
    $V \leftarrow \emptyset$                                        // Nodes
    $E \leftarrow \emptyset$                                        // Arcs
    $W \leftarrow \{\boldsymbol{m}\}$                               // To process
    while $W \neq \emptyset$
        $\boldsymbol{m}' \leftarrow$ remove from $W$
        add $\boldsymbol{m}'$ to $V$
        for $t \in T$ firable from $\boldsymbol{m}'$
            $\boldsymbol{m}'' \leftarrow$ marking such that $\boldsymbol{m}' \xrightarrow{t} \boldsymbol{m}''$
            accel($\boldsymbol{m}''$)
            if $\boldsymbol{m}'' \notin V$ then                     // Already processed?
                add $\boldsymbol{m}''$ to $W$
            add $(\boldsymbol{m}', \boldsymbol{m}'')$ to $E$
    return $(V, E)$

    accel($\boldsymbol{x}$):
        for each ancestor $\boldsymbol{x}'$ of $\boldsymbol{x}$ in the graph $(V, E)$
            if $\boldsymbol{x}' \leq \boldsymbol{x}$ then
                for $p \in P$
                    if $\boldsymbol{x}'(p) < \boldsymbol{x}(p)$ then
                        $\boldsymbol{x}(p) \leftarrow \omega$
```

Observe that the algorithm terminates on every input, and hence that coverability graphs are finite:

Proposition 2. *Algorithm 1 terminates.*

Proof. In order to obtain a contradiction, suppose that there exists an input on which the algorithm does not terminate. The algorithm constructs an infinite graph G. Note that each node of G has at most $|T|$ immediate successors; and hence a *finite* amount of immediate successors. By Kőnig's lemma, G has an infinite simple path $\boldsymbol{m}_0 \longrightarrow \boldsymbol{m}_1 \longrightarrow \cdots$.

For every $i \in \mathbb{N}$, let

$$[\![\boldsymbol{m}_i]\!] := \{p \in P : \boldsymbol{m}_i(p) = \omega\}.$$

Since the path is infinite and P is finite, there exist indices $i_0 < i_1 < \cdots$ such that $[\![\boldsymbol{m}_{i_0}]\!] = [\![\boldsymbol{m}_{i_1}]\!] = \cdots$. Thus, by Dickson's lemma, there exist $j, k \in \mathbb{N}$ such that $j < k$ and $\boldsymbol{m}_{i_j} \leq \boldsymbol{m}_{i_k}$. Note that $\boldsymbol{m}_{i_j} \neq \boldsymbol{m}_{i_k}$, as the path would otherwise not be simple. So, $\boldsymbol{m}_{i_j}(p) < \boldsymbol{m}_{i_k}(p)$ for at least one place $p \in P$. If $\boldsymbol{m}_{i_k}(p) = \omega$, then $[\![\boldsymbol{m}_{i_j}]\!] \neq [\![\boldsymbol{m}_{i_k}]\!]$, which is a contradiction. Hence, we have

$\boldsymbol{m}_{i_k}(p) \in \mathbb{N}$. This means that $\boldsymbol{m}_{i_k}$ should have been accelerated by its ancestor $\boldsymbol{m}_{i_j}$, which is not the case since $[\![\boldsymbol{m}_{i_j}]\!] = [\![\boldsymbol{m}_{i_k}]\!]$. □

1.3.2 Backward algorithm

We study another algorithm that solves the coverability problem: the *backward algorithm*. Rather than identifying markings coverable from a source marking, this algorithm identifies markings that can cover a given target marking. The backward algorithm relies on the representation and manipulation of so-called upward-closed sets.

Let $\boldsymbol{m} \in \mathbb{N}^P$ be a marking. The *upward closure* if $\boldsymbol{m}$ is the set of markings $\uparrow\boldsymbol{m} := \{\boldsymbol{m}' \in \mathbb{N}^P : \boldsymbol{m}' \geq \boldsymbol{m}\}$. The *upward closure* of a set $X \subseteq \mathbb{N}^P$ is the set:

$$\uparrow X := \bigcup_{\boldsymbol{m} \in X} \uparrow\boldsymbol{m}.$$

We say that $X \subseteq \mathbb{N}^P$ if *upward closed* if $\uparrow X = X$.

The set of markings that can cover a given target marking is upward-closed:

Proposition 3. *Let $\boldsymbol{m}' \in \mathbb{N}^P$. The set of markings that can cover $\boldsymbol{m}'$ equals* $\uparrow\text{Pre}^*(\uparrow\boldsymbol{m}')$.

Proof. $\subseteq$) Let $\boldsymbol{m}$ be a marking that can cover $\boldsymbol{m}'$. By definition of coverability, there exists a marking $\boldsymbol{m}''$ such that $\boldsymbol{m} \xrightarrow{*} \boldsymbol{m}''$ and $\boldsymbol{m}'' \geq \boldsymbol{m}'$. Hence,

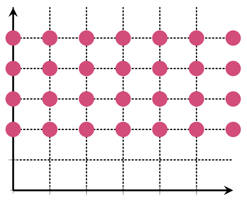
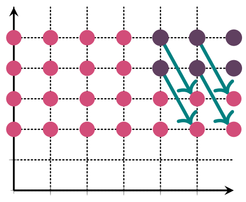
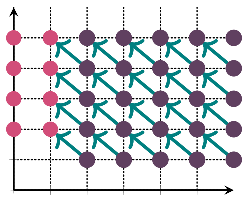
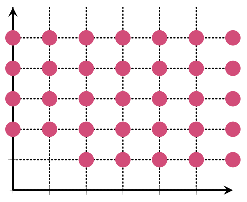
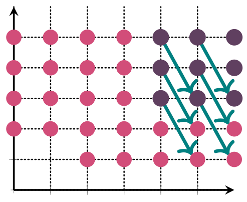
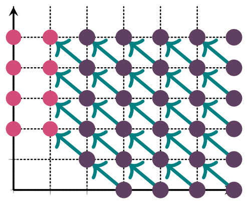
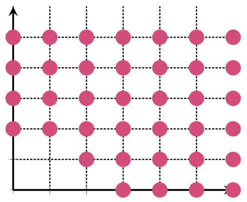
$$\begin{aligned}
\boldsymbol{m} &\in \text{Pre}^*(\boldsymbol{m}'') && \text{(by definition of Pre}^*\text{)} \\
&\subseteq \text{Pre}^*(\uparrow\boldsymbol{m}') && \text{(since } \boldsymbol{m}'' \in \uparrow\boldsymbol{m}'\text{)} \\
&\subseteq \uparrow\text{Pre}^*(\uparrow\boldsymbol{m}') && \text{(by the identity } X \subseteq \uparrow X\text{)}.
\end{aligned}$$

$\supseteq$) Let $\boldsymbol{m} \in \uparrow\text{Pre}^*(\uparrow\boldsymbol{m}')$. There exist markings $\boldsymbol{k}$ and $\boldsymbol{m}''$ such that

$$\begin{tikzcd}
\boldsymbol{m} \arrow[d, phantom, "\geq"] & \\
\boldsymbol{k} \arrow[r, "*"] & \boldsymbol{m}'' \arrow[d, phantom, "\geq"] \\
& \boldsymbol{m}'
\end{tikzcd}$$

Thus, by monotonicity, there exists a marking $\boldsymbol{m}'''$ such that $\boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'''$ and $\boldsymbol{m}''' \geq \boldsymbol{m}'' \geq \boldsymbol{m}'$. Informally, "monotonicity" means that a larger budget of tokens allows to fire (at least) as many transitions (see Exercise 1.6)). Thus, we conclude that $\boldsymbol{m}$ can cover $\boldsymbol{m}'$. □

The backward algorithm seeks to compute a representation of $\uparrow\text{Pre}^*(\uparrow\boldsymbol{m}')$. For example, consider the Petri net depicted in Figure 1.8 with $\boldsymbol{m}' = (0, 2)$. The backward algorithm begins with the set $\uparrow\boldsymbol{m}'$, and then iteratively computes the immediate predecessors of each of its marking, until stabilization:

Iter.	Predecessors under t_1	Predecessors under t_2	Current set
0	—	—	
1			
2			
3		set unchanged	

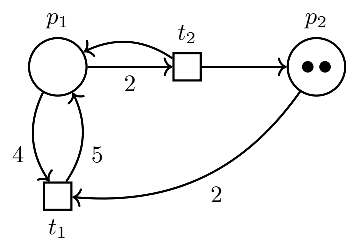


Figure 1.8: A Petri net.

Since upward-closed sets are infinite, this procedure is not, *a priori*, effective. To make it effective, we must be able to:

- represent upward-closed sets symbolically;
- compute immediate predecessors of all markings of a upward-closed set.

The first requirement is made possible through bases. A *basis* of an upward closed set X is a set B such that $\uparrow B = X$. A basis B is *minimal* if it is *not* the case that there exist $\boldsymbol{x}, \boldsymbol{y} \in B$ such that $\boldsymbol{x} \geq \boldsymbol{y}$ and $\boldsymbol{x} \neq \boldsymbol{y}$. In other words, B is minimal if all of its elements are incomparable. Figure 1.9 gives an example of an upward closed set and of its minimal basis.

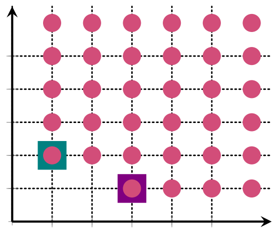


Figure 1.9: Illustration of an upward-closed set (circles). Its minimal basis $\{(1,2),(3,1)\}$ is represented with squares.

Proposition 4. *Any upward-closed set $X \subseteq \mathbb{N}^P$ has a minimal basis.*

Thus, we manipulate finite bases as representatives of upward-closed sets. In particular, it is possible to test membership in an upward-closed set represented by a basis B since:

$$\boldsymbol{m} \in \uparrow B \iff \text{there exists } \boldsymbol{k} \in B \text{ such that } \boldsymbol{m} \geq \boldsymbol{k}.$$

For the second requirement, we compute, for each element of the basis and each transition, the smallest marking that can cover it by firing this transition. Formally, let $t \in T$ be a transition and let $\boldsymbol{m} \in \mathbb{N}^P$ be a marking. The *smallest marking that can cover $\boldsymbol{m}$ by firing t* is the marking $\boldsymbol{m}_t$ such that

$$\boldsymbol{m}_t(p) := \max(\underbrace{F(p,t)}_{\text{“necessary budget”}}, \underbrace{\boldsymbol{m}(p) + F(p,t) - F(t,p)}_{\text{“backward firing”}}) \qquad \text{for all } p \in P.$$

The full procedure is described in Algorithm 2. Let us reconsider the Petri net illustrated in Figure 1.8 with $\boldsymbol{m}' = (0,2)$. This time, we execute the backward algorithm by manipulating directly a basis:

Iter.	Basis B	Predecessors	
0	$\{(0,2)\}$	$(0,2)_{t_1} = (4,4)$	$(0,2)_{t_2} = (2,1)$
1	$\{(0,2),(2,1)\}$	$(2,1)_{t_1} = (4,3)$	$(2,1)_{t_2} = (3,0)$
2	$\{(0,2),(2,1),(3,0)\}$	$(3,0)_{t_1} = (4,2)$	$(3,0)_{t_2} = (4,0)$
3	$\{(0,2),(2,1),(3,0)\}$	basis unchanged	

Algorithm 2: Backward algorithm.

```
Input: Petri net $\mathcal{N} = (P, T, F)$ and $\boldsymbol{m}' \in \mathbb{N}^P$
Output: A minimal basis of $\uparrow\text{Pre}^*(\uparrow\boldsymbol{m}')$
backward($\mathcal{N}, \boldsymbol{m}'$):
    $B' \leftarrow \{\boldsymbol{m}'\}$
    do
        $B \leftarrow B'$                                   // Current basis
        for $\boldsymbol{m} \in B$
            for $t \in T$
                add $\boldsymbol{m}_t$ to $B'$
        minimize($B'$)                          // Remove redundant markings
    while $B' \neq B$
    return $B$
minimize($X$):
    for $\boldsymbol{x} \in X$
        for $\boldsymbol{y} \in X$
            if $\boldsymbol{x} \leq \boldsymbol{y} \wedge \boldsymbol{x} \neq \boldsymbol{y}$ then remove $\boldsymbol{y}$ from $X$
```

Proposition 5. *Algorithm 2 terminates.*

Proof. For the sake of contradiction, assume that the algorithm does not terminate on some input. Let $G = (V, E)$ be the directed graph such that:

- $V \subseteq \mathbb{N}^P$ contains node $\boldsymbol{m}$ iff $\boldsymbol{m}$ ever appears in set B;
- E has an edge from $\boldsymbol{m}$ to $\boldsymbol{m}'$ iff $\boldsymbol{m} \geq \boldsymbol{m}'$.

We make the following observations:

- G has infinitely many nodes. Indeed, each iteration of the **do** loop creates a new marking that was not part of B.
- Every node of G can be reached from at least one node of in-degree zero. Indeed, assume that there exists an infinite simple path $\boldsymbol{m}_0 \leftarrow \boldsymbol{m}_1 \leftarrow \cdots$. By Dickson's lemma, there exist $i, j \in \mathbb{N}$ such that $i < j$ and $\boldsymbol{m}_i < \boldsymbol{m}_j$. By definition of E, we have $\boldsymbol{m}_j \geq \boldsymbol{m}_i$, which is a contradiction.
- G has finitely many nodes of in-degree zero. Indeed, let $Z = [\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots]$ be such nodes. If Z is infinite, then by Dickson's lemma, there exist $i, j \in \mathbb{N}$ such that $\boldsymbol{z}_i \leq \boldsymbol{z}_j$. Thus, G has an edge from $\boldsymbol{z}_j$ to $\boldsymbol{z}_i$, which contradicts the fact that $\boldsymbol{z}_i$ has in-degree zero.

We conclude that G must have a node $\boldsymbol{m}_0$ of in-degree zero that can reach infinitely many nodes. By Kőnig's lemma, there exists an infinite simple path $\boldsymbol{m}_0 \longrightarrow \boldsymbol{m}_1 \longrightarrow \cdots$. By Dickson's lemma, there exist $i, j \in \mathbb{N}$ such that $i < j$ and $\boldsymbol{m}_i < \boldsymbol{m}_j$. By definition of E, we have $\boldsymbol{m}_i \geq \boldsymbol{m}_j$, which is a contradiction. □

Proposition 6 ([BG11])**.** *In the worst case, the* **do** *loop of Algorithm 2 is iterated* $\mathcal{O}(2^{2^{\mathrm{poly(n)}}})$ *times, and basis* B *has size* $|B| \in \mathcal{O}(2^{2^{\mathrm{poly(n)}}})$*.*

1.4 Exercices

1.1) Consider the following program with a global Boolean variable x: ?

```
         x = false

         while ?:
           spawn proc()

         proc():
p₀:        if ¬x: x = true else: goto p₀
p₁:        while ¬x: pass
p₂:        // code
```

a) Model the program as a Petri net.

b) Formalize this property: "several processes can reach line p_2".

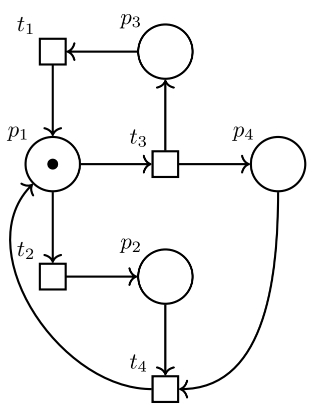
1.2) Consider the following program with a global Boolean variable x: ?

```
         x = false

         while ?:
           spawn proc2()

         proc2():
p₀:        if ¬x: x = true else: goto p₀
p₁:        while ¬x: pass
p₂:        x = ¬x
```

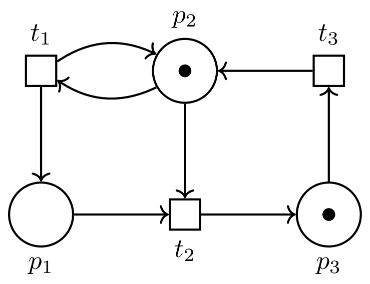
a) Model the program as a Petri net.

b) Formalize this property: "the program may terminate with x $=$ true".

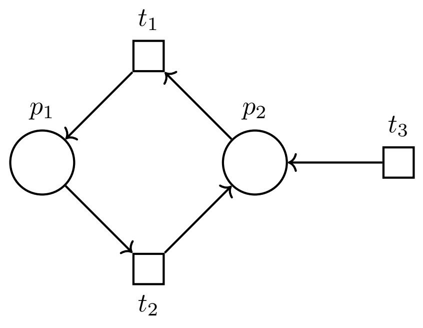
1.3) Compute a coverability graph of this Petri net from $(1, 0, 0, 0)$: ?



1.4) Execute the backward algorithm on this Petri net from $(0, 1, 1)$: **?**



1.5) Show that the order in which we construct a coverability graph may change its size. Consider this Petri net: **?**



1.6) Prove the monotonicity property, *i.e.* $\boldsymbol{m} \xrightarrow{t} \boldsymbol{m}'$ and $\boldsymbol{k} \geq \boldsymbol{m}$ implies the existence of $\boldsymbol{k}' \geq \boldsymbol{m}'$ such that $\boldsymbol{k} \xrightarrow{t} \boldsymbol{k}'$. **?**

1.7) Consider an n bit integer variable `x` that can be incremented.

a) Model `x` with 8 places for $n = 3$, and then generalize.

b) Model `x` with 6 places for $n = 3$, and then generalize.

c) Model the test "`x == 0`".

d) Assuming that `x` is signed and represented under two's complement, model the test "`x > 0`".

1.8) Show that the coverability problem reduces to the reachability problem. In other words, explain how to decide the coverability problem given an algorithm that solves the reachability problem.

1.9) Show that these problems are both equivalent to the reachability problem:

<u>ZERO REACHABILITY PROBLEM</u>

INPUT: Petri net $\mathcal{N} = (P, T, F)$ and marking $\boldsymbol{m} \in \mathbb{N}^P$

QUESTION: $\boldsymbol{m} \xrightarrow{*} \mathbf{0}$?

<u>ZERO PLACE REACHABILITY PROBLEM</u>

INPUT: Petri net $\mathcal{N} = (P, T, F)$, marking $\boldsymbol{m} \in \mathbb{N}^P$ and $p \in P$

QUESTION: $\exists \boldsymbol{m}' \in \mathbb{N}^P : \boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'$ and $\boldsymbol{m}'(p) = 0$?

1.10) Show that this problem is equivalent to the coverability problem:

<u>TRANSITION FIRABILITY</u>

INPUT: Petri net $\mathcal{N} = (P, T, F)$, marking $\boldsymbol{m} \in \mathbb{N}^P$ and $t \in T$

QUESTION: $\exists \boldsymbol{m}', \boldsymbol{m}'' \in \mathbb{N}^P : \boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'$ and $\boldsymbol{m}' \xrightarrow{t} \boldsymbol{m}''$?

1.11) Suppose the codomain of flow functions was $\{0, 1\}$ rather than $\mathbb{N}$. How could we emulate values from $\mathbb{N}$?

1.12) A *reset Petri net* is a Petri net that can also use reset arcs. A *reset arc* from a place p to a transition t empties place p upon firing transition t. Show that coverability graphs do not allow to solve the coverability problem for resets Petri nets, *i.e.* the coverability graph of a marked reset Petri net does not necessarily contain a correct representation of coverable markings.

Remark.

The backward algorithm works for reset Petri nets.

2

Relaxations

In the previous chapter, we covered two algorithms to solve the coverability problem, but none to solve the reachability problem. In this chapter, we introduce reachability relaxations that can, in particular, be used to prove unreachability.

2.1 Pseudo-reachability

Recall that the reachability problem has nonelementary complexity: solving it requires Ackermannian time. This may be surprising to someone not accustomed to Petri nets, as reachability may appear to boil down to solving an integer linear program. However, this is not the case: the difficulty lies in the firing (nonnegativity) constraints.

Nonetheless, this holds for deciding *pseudo-reachability*, *i.e.* whether $\boldsymbol{u} \overset{*}{\dashrightarrow} \boldsymbol{v}$ holds, where $\dashrightarrow$ is defined like $\longrightarrow$ but without any firing constraint, and so where places may temporarily hold negative numbers of tokens. Indeed, the order in which transitions are fired becomes irrelevant, and hence it suffices to solve the following system of linear Diophantine equations, which is known as the *marking equation*:

$$\exists \boldsymbol{x} \in \mathbb{N}^T : \boldsymbol{v} - \boldsymbol{u} = \sum_{t \in T} \boldsymbol{\Delta}(t) \cdot \boldsymbol{x}(t),$$

where $\boldsymbol{\Delta}(t) \in \mathbb{N}^P$ is such that $\boldsymbol{\Delta}(t)(p) := F(t, p) - F(p, t)$ for every $p \in P$.

Solving the marking equation amounts to checking the feasibility of an integer linear program, which is well-known to be NP-complete. While it may be considered intractable by some, this complexity pales in comparison to the ACKERMANN-compltenesss of reachability.

Observation.

If $\boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'$, then $\boldsymbol{m} \overset{*}{\dashrightarrow} \boldsymbol{m}'$. Thus, if $\boldsymbol{m} \overset{*}{\not\dashrightarrow} \boldsymbol{m}'$, then $\boldsymbol{m} \overset{*}{\not\rightarrow} \boldsymbol{m}'$.

Pseudo-reachability turns out to be relevant in practice. Indeed, by the above observation, we can (attempt to) prove that a marking representing an error state cannot be attained, by showing that it is not pseudo-reachable. For example, using *only* the marking equation, [ELM+14] could verify some safety properties to be satisfied by 84 concurrent systems out of 115 instances arising from mutual exclusion algorithms, communication protocols, multi-threaded C programs with shared-memory, ERLANG programs, and systems modeling message provenance analysis of a bug-tracking system and a medical messaging system.[1]

2.2 Continuous reachability

Using pseudo-reachability to approximate reachability comes at a great price: control over the order in which transitions can be fired is entirely lost. An alternative avenue consists in preserving (nonnegative) firing constraints, but relaxing the *discreteness* of markings: We allow the firing constraints and effects of transitions to be scaled by any factor $\lambda \in (0,1]$, provided the number of tokens remains nonnegative. Hence, in this setting, places may contain "pieces of tokens" (which could be seen as liquid).

Formally, we write $\boldsymbol{u} \overset{\lambda t}{\twoheadrightarrow} \boldsymbol{v}$ iff $\boldsymbol{u}(p) \geq \lambda \cdot F(p,t)$ for every incoming place $p \in P$ of t, and $\boldsymbol{v} = \boldsymbol{u} + \lambda \cdot \boldsymbol{\Delta}(t)$. Note that we use a double arrow tip for this relaxation, while dotted lines specified pseudo-reachability (we will use shortly $\dashrightarrow\!\!\!\!\twoheadrightarrow$ for a combination of both). This gives rise to the *continuous reachability* relation where sequences are drawn from

$$T^\dagger \coloneqq ((0,1] \times T)^* \text{ instead of } T^* \cong (\{1\} \times T)^*.$$

Example.

Consider the Petri net depicted in Figure 3.1. We have:

$$(3,1,0) \overset{1s}{\twoheadrightarrow} (2,2,0) \overset{\frac{2}{3}t}{\twoheadrightarrow} (2,0,2/3).$$

Note that neither s nor t is firable in the last above marking since place p_2 is empty.

[1] They could verify 96 instances by combining the marking equation with a more sophisticated SAT/SMT-based approach.

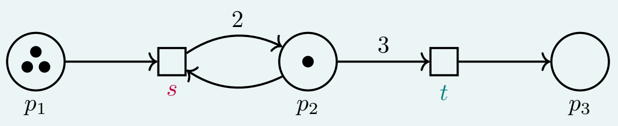


Figure 2.1: Example of a Petri net marked with $(3, 1, 0)$.

Remark.

Petri nets equipped with continuous reachability are perhaps more commonly known as *continuous Petri nets*. The latter were introduced by [DA87, DA10] to model, *e.g.*, physical systems depending on continuous variables.

Observation.

If $\boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'$, then $\boldsymbol{m} \overset{*}{\twoheadrightarrow} \boldsymbol{m}'$. Thus, if $\boldsymbol{m} \overset{*}{\not\twoheadrightarrow} \boldsymbol{m}'$, then $\boldsymbol{m} \not\xrightarrow{*} \boldsymbol{m}'$.

The continuous relaxation also incurs a cost. For example, if places count the number of threads at some location of a replicated concurrent program, then a thread may now split in *half*, which surely cannot happen in reality. Yet, some ratios are preserved between places, and we cannot obtain negative amounts as with pseudo-reachability. Moreover, continuous reachability turns out to be *easier* than the latter: it is P-complete.

2.2.1 Characterization of continuous reachability

This precise complexity has been established by [FH15] who described a polynomial time algorithm exploiting their following characterization:

Theorem 1. *We have* $\boldsymbol{u} \overset{*}{\twoheadrightarrow} \boldsymbol{v}$ *iff there exist* $\boldsymbol{u}', \boldsymbol{v}' \in \mathbb{R}^P_{\geq 0}$ *and* $\sigma, \sigma_{\blacktriangleright\blacktriangleright}, \sigma_{\blacktriangleleft\blacktriangleleft} \in T^\dagger$ *s.t. all three sequences use exactly the same transitions (possibly organized differently) and*

$$\boldsymbol{u} \overset{\sigma}{\dashrightarrow} \boldsymbol{v},\ \boldsymbol{u} \overset{\sigma_{\blacktriangleright\blacktriangleright}}{\twoheadrightarrow} \boldsymbol{u}',\ \boldsymbol{v}' \overset{\sigma_{\blacktriangleleft\blacktriangleleft}}{\twoheadrightarrow} \boldsymbol{v}.$$

By “organized differently,” we mean that transitions appearing in σ, $\sigma_{\blacktriangleright\blacktriangleright}$ and $\sigma_{\blacktriangleleft\blacktriangleleft}$ may be ordered differently, with distinct scaling factors, and with varying numbers of occurrences, *i.e.* what matters is whether a transition appears at least once or not at all.

Theorem 1 states that continuous reachability amounts to *continuous pseudo-reachability* — which relaxes *both* nonnegativity and discreteness — provided there exist sequences $\sigma_{\blacktriangleright\blacktriangleright}$ and $\sigma_{\blacktriangleleft\blacktriangleleft}$ that witness forward and backward firability (regardless of the markings reached).

Continuous pseudo-reachability. Let us give some insights on why Theorem 1 involves three sequences. A natural question is whether continuous reachability simply amounts to continuous *pseudo-reachability*, *i.e.*:

$$\boldsymbol{u} \xrightarrow{*}\!\!\!\!\twoheadrightarrow \boldsymbol{v} \overset{?}{\iff} \boldsymbol{u} \overset{*}{\dashrightarrow}\!\!\!\!\twoheadrightarrow \boldsymbol{v}.$$

Of course, the implication from left to right holds as continuous pseudo-reachability is less restrictive, but what about the one from right to left?

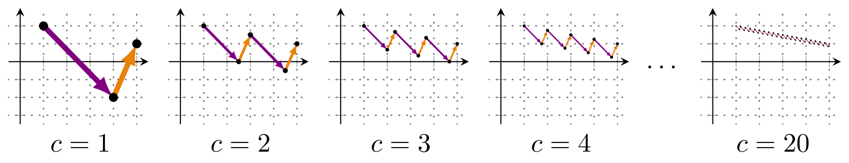


Figure 2.2: Illustration of $\text{✂}_c(st)$, where $(1, 2) \overset{st}{\dashrightarrow}\!\!\!\!\twoheadrightarrow (5, 1)$ with $\boldsymbol{\Delta}(s) = (3, -4)$ and $\boldsymbol{\Delta}(t) = (1, 3)$. The x-axis and y-axis each indicate the tokens count of a place, and each point depicts a (continuous) marking.

Let $\boldsymbol{u} \overset{\sigma}{\dashrightarrow}\!\!\!\!\twoheadrightarrow \boldsymbol{v}$. Consider the sequence

$$\text{✂}_c(\sigma) := \overbrace{(1/c)\sigma \ \cdots \ (1/c)\sigma}^{\text{repeated } c \text{ times}},$$

where $(1/c)\sigma$ corresponds to σ in which all transitions are multiplied by $1/c$. Note that $\text{✂}_c(\sigma)$ globally consumes and produces the same amount of tokens as σ, as each transition is scaled down by $1/c$ and copy/pasted c times. Figure 2.2 depicts an example of this transformation where $\sigma = st$ with $\boldsymbol{\Delta}(s) = (3, -4)$ and $\boldsymbol{\Delta}(t) = (1, 3)$. In this example, we have

$$(1, 2) \overset{\sigma}{\dashrightarrow}\!\!\!\!\twoheadrightarrow (5, 1), \text{ but } \textit{not } (1, 2) \xrightarrow{\sigma}\!\!\!\!\twoheadrightarrow (5, 1),$$

as a place drops below zero (see the y-axis for $c = 1$ in Figure 2.2). However, we *do* have

$$(1, 2) \xrightarrow{\text{✂}_3(\sigma)}\!\!\!\!\twoheadrightarrow (5, 1).$$

Therefore, if c tends to infinity, then $\text{✂}_c(\sigma)$ becomes a “straight line” with the same initial and target markings (see Figure 2.2 up to $c = 20$). Hence, we *informally* have

$$\boldsymbol{u} \xrightarrow{\text{✂}_\infty(\sigma)}\!\!\!\!\twoheadrightarrow \boldsymbol{v}.$$

Of course, we cannot set $c = \infty$ as the sequence must remain finite. But, $c = 3$ suffices in our example, and it is tempting to conclude that we can always pick c large enough.

Unfortunately, this is not always the case. Consider the example depicted in Figure 2.3 which is a slight modification of our previous example where we go from $(1, 1)$ to $(5, 0)$ rather than from $(1, 2)$ to $(5, 1)$. There, no matter the value of c, the second place (y-axis) will always be negative at some point, although increasingly closer to zero. This is due to the fact that we are trying to reach zero *from below*.

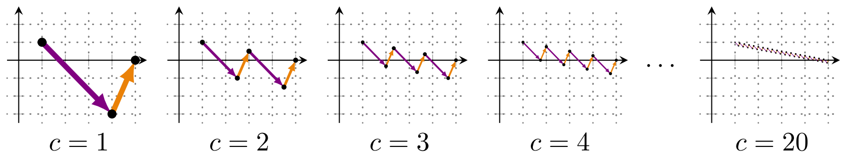


Figure 2.3: Illustration of $✂_c(st)$ as in Figure 2.2, but starting from marking $(1, 1)$ rather than $(1, 2)$.

It can be shown that we can find $c \in \mathbb{N}$ such that $\boldsymbol{x} \overset{\tau}{\dashrightarrow\!\!\!\!\twoheadrightarrow} \boldsymbol{y}$ implies $\boldsymbol{x} \xrightarrow{✂_c(\tau)} \boldsymbol{y}$, if:

1. $\boldsymbol{x}(p) > 0$ for each place p from which τ ever consumes tokens;
2. $\boldsymbol{y}(p) > 0$ for each place p in which τ ever produces tokens.

These conditions are the reason behind sequences $\sigma_{\blacktriangleright\blacktriangleright}$ and $\sigma_{\blacktriangleleft\blacktriangleleft}$ appearing in Theorem 1.

Proof sketch of Theorem 1. With these insights in mind, let us provide a proof sketch of the implication going from right to left in Theorem 1. We must turn markings $\boldsymbol{u}$ and $\boldsymbol{v}$ into markings satisfying (1) and (2). Pictorially, we have:

$$\begin{tikzcd}
\boldsymbol{u} \arrow[rrr, dotted, two heads, "\sigma"] \arrow[dr, two heads, "\sigma_{\blacktriangleright\blacktriangleright}"'] & & & \boldsymbol{v} & \\
& \boldsymbol{u}' & & & \boldsymbol{v}' \arrow[ul, two heads, "\sigma_{\blacktriangleleft\blacktriangleleft}"']
\end{tikzcd}$$

By rescaling a sequence $\boldsymbol{x} \xrightarrow{\tau \,=\, \lambda_1 t_1 \lambda_2 t_2 \cdots \lambda_n t_n} \boldsymbol{y}$ with exponentially smaller factors, we can "saturate" the target marking. More precisely, consider the following sequence for some suitable constant $d \in \mathbb{N}$:

$$\boldsymbol{x} \xrightarrow{\lightning(\tau) \,:=\, (\lambda_1/d)t_1(\lambda_2/d^2)t_2 \cdots (\lambda_n/d^n)t_n} \boldsymbol{y}'.$$

This operation, denoted $\lightning(\cdot)$, yields a marking $\boldsymbol{y}'$ with *as many nonempty* places as possible.

By rescaling globally with a small factor $\lambda \in (0, 1]$, we can further obtain a vector $\boldsymbol{y}''$ arbitrarily close to $\boldsymbol{x}$, as "almost nothing" is fired. The same idea also applies backwards, *i.e.* going from a target marking $\boldsymbol{y}$ to some small saturated marking $\boldsymbol{x}''$. Thus, using the same notation informally for both directions, we obtain:

$$\begin{tikzcd}
\boldsymbol{u} \arrow[rr, dotted, two heads, "\sigma"] \arrow[dr, "\underbrace{\lambda \cdot \lightning(\sigma_{\blacktriangleright\!\blacktriangleright})}_{\sigma'_{\blacktriangleright\!\blacktriangleright}}"'] & & \boldsymbol{v} & \\
& \boldsymbol{u}'' & & \boldsymbol{v}'' \arrow[ul, "\overbrace{\lambda \cdot \lightning(\sigma_{\blacktriangleleft\!\blacktriangleleft})}^{\sigma'_{\blacktriangleleft\!\blacktriangleleft}}"']
\end{tikzcd}$$

Recall that σ, $\sigma_{\blacktriangleright\!\blacktriangleright}$ and $\sigma_{\blacktriangleleft\!\blacktriangleleft}$ all use the exact same set of transitions. This is also the case for $\sigma'_{\blacktriangleright\!\blacktriangleright}$ and $\sigma'_{\blacktriangleleft\!\blacktriangleleft}$, as operation $\lightning(\cdot)$ only changes scaling factors, not transitions. Hence, markings $\boldsymbol{u}''$ and $\boldsymbol{v}''$ now both satisfy (1) and (2).

Moreover, $\boldsymbol{u}'' \overset{\sigma}{\dashrightarrow} \boldsymbol{v}''$ *almost* holds as these markings are respectively *very close* to $\boldsymbol{u}$ and $\boldsymbol{v}$. Fortunately, we can bridge this tiny gap. Indeed, since λ was picked sufficiently small and since all three sequences share the same transitions, we can "subtract" both $\sigma'_{\blacktriangleright\!\blacktriangleright}$ and $\sigma'_{\blacktriangleleft\!\blacktriangleleft}$ from σ, which corresponds to decreasing scaling factors occurring within σ.

Thus, we are done since $\boldsymbol{u} \xrightarrow{\sigma'_{\blacktriangleright\!\blacktriangleright}\, ✄_c(\sigma - \sigma'_{\blacktriangleright\!\blacktriangleright} - \sigma'_{\blacktriangleleft\!\blacktriangleleft})\, \sigma'_{\blacktriangleleft\!\blacktriangleleft}} \boldsymbol{v}$ holds for some $c \in \mathbb{N}$:

$$\begin{tikzcd}
\boldsymbol{u} \arrow[rrr, dotted, two heads, "\sigma"] \arrow[dr, "\sigma'_{\blacktriangleright\!\blacktriangleright}"'] & & & \boldsymbol{v} & \\
& \boldsymbol{u}'' \arrow[rrr, two heads, "✄_c(\sigma - \sigma'_{\blacktriangleright\!\blacktriangleright} - \sigma'_{\blacktriangleleft\!\blacktriangleleft})"'] & & & \boldsymbol{v}'' \arrow[ul, "\sigma'_{\blacktriangleleft\!\blacktriangleleft}"']
\end{tikzcd}$$

In essence, the algorithm of [FH15] identifies sequence σ in polynomial time via linear programming, while $\sigma_{\blacktriangleright\!\blacktriangleright}$ and $\sigma_{\blacktriangleleft\!\blacktriangleleft}$ are obtained by a graph saturation procedure. As the three sequences must use the same transitions, they progressively refine candidate transitions until a greatest fixed point is reached.

2.2.2 From continuous reachability to logic

While the algorithm of [FH15] for continuous reachability can theoretically run in polynomial time, it relies on solving several linear programs: between a linear and quadratic number depending on the implementation. Moreover, it is risky to use a numerical procedure, such as most industrial implementations of the simplex algorithm. Indeed, floating-point errors could technically lead to erroneous outcomes, wrongly concluding unreachability. This is not particularly desirable in the context of verification where unreachability typically corresponds to the absence of errors.

The second issue can be addressed by using an exact implementation of the simplex algorithm (*e.g.* [ACDE07]) or an SMT solver supporting linear real arithmetic (*e.g.* [dMB08, BCD⁺11]), *i.e.* $\mathrm{FO}\langle\mathbb{R}, +, <\rangle$ (whose validity problem is decidable, and NP-complete for the existential fragment). However, there exists an alternative approach which relies on a *single* call to an SMT solver [BFHH17]. The idea consists in translating the continuous reachability relation into an existentially quantified formula from linear real arithmetic, based on the conditions of Theorem 1:

$$\psi_{\overset{*}{\dashrightarrow}}(\boldsymbol{u}, \boldsymbol{v}) = \exists \boldsymbol{x} \in \mathbb{R}^{T}_{\geq 0} : \varphi_{\text{mark-eq}}(\boldsymbol{u}, \boldsymbol{x}, \boldsymbol{v}) \wedge \varphi_{\blacktriangleright\!\blacktriangleright}(\boldsymbol{u}, \boldsymbol{x}) \wedge \varphi_{\blacktriangleleft\!\blacktriangleleft}(\boldsymbol{x}, \boldsymbol{v}).$$

Here, $\boldsymbol{x}$ represents sequence σ of Theorem 1 in the sense that $\boldsymbol{x}(t)$ indicates the sum of all scaling factors across occurrences of transition t in σ, and formula $\varphi_{\text{mark-eq}}(\boldsymbol{u}, \boldsymbol{x}, \boldsymbol{v})$ is the marking equation over $\mathbb{R}_{\geq 0}$:

$$\varphi_{\text{mark-eq}}(\boldsymbol{u}, \boldsymbol{x}, \boldsymbol{v}) := \left(\boldsymbol{v} - \boldsymbol{u} = \sum_{t \in T} \boldsymbol{\Delta}(t) \cdot \boldsymbol{x}(t)\right).$$

Formulas $\varphi_{\blacktriangleright\!\blacktriangleright}(\boldsymbol{u}, \boldsymbol{x})$ and $\varphi_{\blacktriangleleft\!\blacktriangleleft}(\boldsymbol{x}, \boldsymbol{v})$ check for the existence of firing sequences $\sigma_{\blacktriangleright\!\blacktriangleright}$ and $\sigma_{\blacktriangleleft\!\blacktriangleleft}$ of Theorem 1 with respect to transitions $\{t \in T : \boldsymbol{x}(t) > 0\}$. By adapting a construction of [VSS05], these two formulas can be made of *linear size*, which yields an overall formula $\psi_{\overset{*}{\twoheadrightarrow}}$ of linear size. The idea is to number each node $v \in P \cup T$ with a number $\boldsymbol{z}(v)$ that corresponds to the iteration at which the graph saturation procedure for $\varphi_{\blacktriangleright\!\blacktriangleright}$ and $\varphi_{\blacktriangleleft\!\blacktriangleleft}$ would mark them (where 0 stands for "undiscovered"), *e.g.*:

$$\begin{aligned}
&\varphi_{\blacktriangleright\!\blacktriangleright}(\boldsymbol{u}, \boldsymbol{x}) := \exists \boldsymbol{z} \in \mathbb{R}_{\geq 0}^{P \cup T} : \\
&\quad \bigwedge_{t \in T} \left[(\boldsymbol{x}(t) > 0) \rightarrow \left((\boldsymbol{z}(t) > 0) \land \bigwedge_{p \text{ entering } t} (0 < \boldsymbol{z}(p) < \boldsymbol{z}(t))\right)\right] \land \\
&\quad \bigwedge_{p \in P} \left[(\boldsymbol{z}(p) > 0) \rightarrow \left((\boldsymbol{u}(p) > 0) \lor \bigvee_{t \text{ entering } p} ((\boldsymbol{x}(t) > 0) \land (\boldsymbol{z}(t) < \boldsymbol{z}(p)))\right)\right].
\end{aligned}$$

Experimental results show that solving $\psi_{\overset{*}{\twoheadrightarrow}}$ allows to efficiently verify safety of concurrent systems in practice [BFHH17]. Moreover, solving $\psi_{\overset{*}{\twoheadrightarrow}}$ works well as a pruning method within a complete procedure such as the backward algorithm [ACJT00], which fits within the more general framework of combining forward invariant generation with backward reachability analysis [GLS18].

A natural question arising from this approach is whether the full power of existential linear arithmetic is needed. As it turns out, it is *not* the case: continuous reachability is characterized by a fragment of linear arithmetic which admits a *polynomial time* decision procedure [BH17]. This fragment $\mathcal{L}$ is a syntactic restriction where variables are quantified over $\mathbb{R}_{\geq 0}$, and where a formula is a conjunction of *convex semi-linear Horn clauses*, which are of the form:

$$(\boldsymbol{a} \cdot \boldsymbol{x} \succ b) \lor \bigvee_{1 \leq i \leq m} \bigwedge_{1 \leq j \leq n_i} \boldsymbol{x}(j) > 0,$$

where each $\boldsymbol{a}(\ell) \in \mathbb{R}$, $b \in \mathbb{R}$ and $\succ \in \{=, \geq, >\}$.

Clauses of the form "$\boldsymbol{a} \cdot \boldsymbol{x} = b$" and "$\boldsymbol{x}(\ell) > 0 \lor \bigvee_{1 \leq i \leq m} \bigwedge_{1 \leq j \leq n_i} \boldsymbol{x}(j) > 0$" can respectively implement $\varphi_{\text{mark-eq}}$ (immediate) and $\varphi_{\blacktriangleright\!\blacktriangleright} \land \varphi_{\blacktriangleleft\!\blacktriangleleft}$ (much less obvious).

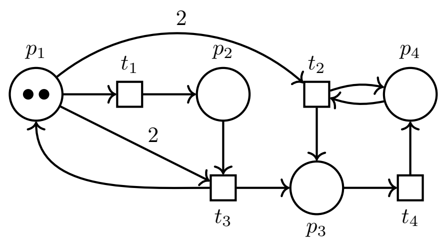
Observe that clauses of the form "$\boldsymbol{a} \cdot \boldsymbol{x} \succ b$" correspond to constraints of linear programs. Moreover, $\mathcal{L}$ can express the family

$$\{\neg(y_0 > 0) \lor y_1 > 0 \lor \cdots \lor y_k > 0\}_{k > 0}$$

which cannot be defined by any convex polytope. Hence, in terms of expressiveness, $\mathcal{L}$ strictly lies in between linear programs and linear arithmetic.

2.3 Exercices

2.1) Consider the following Petri net: **?**



Say whether $\boldsymbol{m}_{\text{src}} \overset{*}{\dashrightarrow} \boldsymbol{m}_{\text{tgt}}$ and $\boldsymbol{m}_{\text{src}} \overset{*}{\twoheadrightarrow} \boldsymbol{m}_{\text{tgt}}$ for:

a) $\boldsymbol{m}_{\text{src}} := (2, 0, 0, 0)$ and $\boldsymbol{m}_{\text{tgt}} := (0, 0, 0, 1)$.

b) $\boldsymbol{m}_{\text{src}} := (2, 0, 0, 0)$ and $\boldsymbol{m}_{\text{tgt}} := (0, 0, 1, 0)$.

c) $\boldsymbol{m}_{\text{src}} := (2, 0, 0, 0)$ and $\boldsymbol{m}_{\text{tgt}} := (1, 0, 1, 0)$.

2.2) What happens if we omit $\sigma_{\twoheadleftarrow}$ from the characterization of Theorem 1?

2.3) Show that $\overset{*}{\dashrightarrow}$ and $\overset{*}{\twoheadrightarrow}$ are incomparable. **?**

2.4) Let $\mathcal{N} = (P, T, F)$ be a Petri net. For every $p \in P$, let ${}^\bullet p := \{t \in T : F(t, p) > 0\}$ and $p^\bullet := \{t \in T : F(p, t) > 0\}$. For every $Q \subseteq P$, let

$$^\bullet Q := \bigcup_{p \in Q} {}^\bullet p \text{ and } Q^\bullet := \bigcup_{p \in Q} p^\bullet.$$

A *siphon* is a subset of places Q such that ${}^\bullet Q \subseteq Q^\bullet$.

a) Let $Q \subseteq P$ be a siphon. Given a marking $\boldsymbol{m}$, let $\boldsymbol{m}(Q) := \sum_{q \in Q} \boldsymbol{m}(q)$. Show that if $\boldsymbol{m}(Q) = 0$ and $\boldsymbol{m} \twoheadrightarrow \boldsymbol{m}'$, then $\boldsymbol{m}'(Q) = 0$.

b) Show that siphons are closed under union.

c) Show that, for every marking $\boldsymbol{m}$, there exists a maximal siphon Q (w.r.t. inclusion) such that $\boldsymbol{m}(Q) = 0$.

d) Let $S \subseteq P$ and let $\boldsymbol{m}$ be a marking. Let $\mathcal{N}_S$ be the (sub)Petri net induced by S, *i.e.* transitions from $\overline{S}$ are removed from $\mathcal{N}$. Show that the following statements are equivalent:

   i. There exist $\boldsymbol{m}'$ and a weighted sequence σ such that $\boldsymbol{m} \xrightarrow{\sigma}\!\!\!\!\!\twoheadrightarrow \boldsymbol{m}'$ and σ uses exactly the transitions from S.

   ii. It is the case that $\boldsymbol{m}(Q) > 0$ for every siphon Q of $\mathcal{N}_S$.

e) Explain how to algorithmically test 2.4)(d)i using 2.4)(d)ii.

2.5) Explain how to decide coverability under $\overset{*}{\dashrightarrow}$ and $\overset{*}{\twoheadrightarrow}$. Try to do so without modifying the given Petri net.

2.6) The *structural cyclicity* problem asks whether $\boldsymbol{m} \xrightarrow{+} \boldsymbol{m}$ for all markings of a given Petri net, where "+" stands for any nonempty sequence. This problem can be solved in polynomial time. Why? ?

2.7) Given two Petri nets $\mathcal{N}$ and $\mathcal{N}'$ over the same set of places P, and two markings $\boldsymbol{m}_{\text{src}}, \boldsymbol{m}'_{\text{src}} \in \mathbb{R}^P_{\geq 0}$, it is possible to decide whether

$$\{\boldsymbol{m} \in \mathbb{R}^P_{\geq 0} : \boldsymbol{m}_{\text{src}} \overset{*}{\twoheadrightarrow} \boldsymbol{m} \text{ in } \mathcal{N}\} \subseteq \{\boldsymbol{m} \in \mathbb{R}^P_{\geq 0} : \boldsymbol{m}'_{\text{src}} \overset{*}{\twoheadrightarrow} \boldsymbol{m} \text{ in } \mathcal{N}'\}.$$

Why?

2.8) Relaxations can be used within the backward algorithm in order to decide the coverability problem more efficiently (in practice). Why?

3

Directed reachability

In the previous chapter, we introduced some relaxations that can help to witness unreachability. In this chapter, we will see that these same relaxations can be used to witness reachability.

3.1 Weighted graphs

A *(labeled directed) graph* is a triple $G = (V, E, A)$, where V is a set of *nodes*, A is a finite set of elements called *actions*, and $E \subseteq V \times A \times V$ is the set of *edges* labeled by actions. We say that G has *finite out-degree* if the set of outgoing edges $\{(w, a, w') \in E : w = v\}$ is finite for every $v \in V$. Similarly, it has *finite in-degree* if the set of ingoing edges is finite for every $v \in V$. If G has both finite out- and in-degree, then we say that G is *locally finite*.

A *path* π is a finite sequence of nodes $(v_i)_{1 \leq i \leq n}$ and actions $(a_i)_{1 \leq i < n}$ such that $(v_i, a_i, v_{i+1}) \in E$ for all $1 \leq i < n$. We say that π is a *path from v to w* (or a *v-w path*) if $v = v_1$ and $w = v_n$, and its *label* is $a_1 a_2 \cdots a_{n-1}$, where ε denotes the empty sequence.

A *weighted* graph is a tuple $G = (V, E, A, \mu)$ where (V, E, A) is a graph with a *weight function* $\mu \colon E \to \mathbb{R}_{>0}$. The *weight* of path π is the weight of its edges, *i.e.* $\mu(\pi) \coloneqq \sum_{1 \leq i < n} \mu(v_i, a_i, v_{i+1})$. A *shortest path from v to w* is a v-w path π minimizing $\mu(\pi)$. We define $\text{dist}_G \colon V \times V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ as the *distance function* where $\text{dist}_G(v, w)$ is the weight of a shortest path from v to w, with $\text{dist}_G(v, w) \coloneqq \infty$ if there is none.

We assume throughout this chapter that weighted graphs have a *minimal weight*, *i.e.* that $\min\{\mu(e) : e \in E\}$ exists. For graphs with finite out-degree, this ensures that if a path exists between two nodes, then a shortest one exists.[1] This mild assumption always holds in our setting.

[1] Otherwise, there could be increasingly better paths, *e.g.* of weights $1, 1/2, 1/4, \ldots$.

3.2 Directed search algorithms

The approach of this chapter relies on classical pathfinding procedures guided by node selection strategies. Their generic scheme is described in Algorithm 3. Its termination with a value $d \neq \infty$ indicates that the weighted graph $G = (V, E, A, \mu)$ has a path from s to t of weight d, whereas termination with $d = \infty$ signals that $\text{dist}_G(s, t) = \infty$.

Algorithm 3: Directed search algorithm.

```
1  $g \leftarrow [s \mapsto 0, v \mapsto \infty : v \neq s]$
2  $C \leftarrow \{s\}$
3  while $C \neq \emptyset$
4      $v \leftarrow \arg\min_{v \in C} S(g, v)$
5      if $v = t$ then return $g(t)$
6      for $(v, a, w) \in E$
7          if $g(v) + \mu(v, a, w) < g(w)$ then
8              $g(w) \leftarrow g(v) + \mu(v, a, w)$
9              $C \leftarrow C \cup \{w\}$
10     $C \leftarrow C \setminus \{v\}$
11 return $\infty$
```

Algorithm 3 maintains a set of *frontier nodes* C and a mapping $g \colon V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ such that $g(w)$ is the weight of the best known path from s to w, *i.e.* it satisfies this invariant:

$$\text{if } g(v) \neq \infty, \text{ then } g(v) \text{ is the weight of a path from } s \text{ to } v \text{ in } G \text{ whose nodes were all expanded, except possibly } v. \quad (*)$$

In Line 4, a *selection strategy* S determines which node v to *expand* next. Starting from Line 6, a successor w of v is added to the frontier if its distance improves.

Let $h \colon V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ estimate the distance from all nodes to a target $t \in V$. The selection strategies sending (g, v) respectively to $g(v)$, $g(v) + h(v)$ or $h(v)$ yield the classical Dijkstra's, A* and greedy best-first search (*GBFS*) algorithms.

When instantiating S with Dijkstra's selection strategy, a return value $d \neq \infty$ is guaranteed to equal $\text{dist}_G(s, t)$. This is not true for A* and GBFS. However, if h fulfills the following *consistency* properties, then A* also has this guarantee: $h(t) = 0$ and $h(v) \leq \mu(v, a, w) + h(w)$ for every $(v, a, w) \in E$ (see, *e.g.*, [RN09]).

In the setting of infinite graphs, unlike GBFS, A* and Dijkstra's selection strategies guarantee termination if $\text{dist}_G(s, t) \neq \infty$. Yet, we introduce *unbounded heuristics* for which termination is also guaranteed for GBFS. Note that these guarantees would vanish in the presence of zero weights.

3.2.1 Guarantees on GBFS

An *infinite path* π is a sequence of nodes $(v_i)_{i \in \mathbb{N}}$ and actions $(a_i)_{i \in \mathbb{N}}$ such that $(v_i, a_i, v_{i+1}) \in E$ for all $i \in \mathbb{N}$. We say that π is *bounded* w.r.t. h if its nodes are pairwise distinct and there exists $b \in \mathbb{R}_{\geq 0}$ with $h(v_i) \leq b$ for all $i \geq 0$. We say that h is *unbounded* if it admits no bounded sequence. The following lemma enables to prove termination of GFBS in the presence of unbounded heuristics.

Lemma 1. *If G is locally finite and h is unbounded, then the following holds:*

1. *The set of paths of weight at most $c \in \mathbb{R}_{\geq 0}$ starting from node s is finite.*
2. *Let $W \subseteq V$. The set $\text{dist}_G(W, t) \coloneqq \{\text{dist}_G(w, t) : w \in W\}$ has a minimum.*
3. *No node is expanded infinitely often by Algorithm 3.*

Proof. Let $d \coloneqq \min\{\mu(e) : e \in E\}$.

1. Any path of weight at most c traverses at most $k \coloneqq \lceil c/d \rceil$ edges. Since the graph has finite out-degree, the number of paths from s using at most k edges is finite.

2. Suppose the claim false. We have $\text{dist}_G(v_0, t) > \text{dist}_G(v_1, t) > \cdots$ for some $v_0, v_1, \ldots \in W$. Let $k \coloneqq \lceil \text{dist}_G(v_0, t)/d \rceil$. Let $V_{\leq k}$ be the set of nodes that can reach t by traversing at most k edges. Since G has finite in-degree, $V_{\leq k}$ is finite. Moreover, any node $v \in V \setminus V_{\leq k}$ is such that $\text{dist}_G(v, t) > k \cdot d \geq \text{dist}_G(v_0, t)$. Hence, $\{v_0, v_1, \ldots\} \subseteq V_{\leq k}$ is finite, which is a contradiction.

3. For the sake of contradiction, assume a node v is expanded infinitely often. Each time node v is expanded, it is removed from C. Hence, it is reinserted infinitely often in C. Moreover, each time this happens, value $g(v)$ is decreased. Let $q_0, q_1, \ldots \in \mathbb{R}_{\geq 0}$ denote these increasingly smaller values. By $(*)$, there is a path π_i from s to v of weight q_i in G. By (1), $\{\pi_i : i \in \mathbb{N}\}$ is finite as the weight of these paths is at most q_0. This contradicts $q_0 > q_1 > \cdots$. □

Theorem 2. *Algorithm 3 with the greedy best-first search selection strategy always finds reachable targets for locally finite graphs and unbounded heuristics.*

Proof. Assume $\text{dist}_G(s, t) \neq \infty$. For the sake of contradiction, suppose t is never expanded. Let K_i be the subgraph of G induced by nodes expanded at least once within the first i iterations of the **while** loop. In particular, K_1 is the graph made only of node s. Let $K = K_1 \cup K_2 \cup \cdots$. By Lemma 1 (3), no node is expanded infinitely often, hence K is infinite. Moreover, K has finite out-degree, and each node of K is reachable from s in K by $(*)$. Thus, by König's lemma, K contains an infinite path $v_0, v_1, \ldots \in V$ of pairwise distinct nodes.

Let w be a node of K minimizing $\text{dist}_G(w, t)$. Such a node is well-defined by Lemma 1 (2). We have $\text{dist}_G(w, t) \neq \infty$ as t is reachable from s and the latter belongs to $K_1 \subseteq K$. By minimality of $w \neq t$, there exists an edge (w, a, w') of G such that $\text{dist}_G(w', t) < \text{dist}_G(w, t)$ and w' does not appear in K. Note that w' is added to C at some point, but is never expanded as it would otherwise belong to K. Let i be the smallest index such that w belongs to K_i.

Since h is unbounded, there exists j such that $h(v_j) > h(w')$ and v_j is expanded after iteration i of the while loop. This is a contradiction as w' would have been expanded instead of v_j. □

3.3 Directed reachability

In this section, we explain how to instantiate Algorithm 3 for finding short(est) firing sequences witnessing reachability in (weighted) Petri nets. Since Dijkstra's selection strategy does not require any heuristic, we focus on A* and greedy best-first search which require consistent and unbounded heuristics.

3.3.1 Distance under-approximations

A *weighted Petri net* is a tuple $\mathcal{N} = (P, T, F, \mu)$ such that (P, T, F) is a Petri net and $\mu\colon T \to \mathbb{R}_{>0}$ is a *weight function* assigning weights to transitions. A *distance under-approximation* of such a weighted Petri net is a function $d\colon \mathbb{N}^P \times \mathbb{N}^P \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ such that for all $\boldsymbol{m}, \boldsymbol{m}', \boldsymbol{m}'' \in \mathbb{N}^P$:

- $d(\boldsymbol{m}, \boldsymbol{m}') \leq \mathrm{dist}_{\mathcal{N}}(\boldsymbol{m}, \boldsymbol{m}')$,
- $d(\boldsymbol{m}, \boldsymbol{m}'') \leq d(\boldsymbol{m}, \boldsymbol{m}') + d(\boldsymbol{m}', \boldsymbol{m}'')$ (*triangle inequality*), and
- d is effective, *i.e.* there is an algorithm that evaluates d on all inputs.

We naturally obtain a heuristic from d for a directed search towards marking $\boldsymbol{m}_{\text{tgt}}$. Indeed, let $h\colon \mathbb{N}^P \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be defined by $h(\boldsymbol{m}) \coloneqq d(\boldsymbol{m}, \boldsymbol{m}_{\text{tgt}})$. The following proposition — whose proof is deferred to Exercise 3.2) — shows that h is a suitable heuristic for A*:

Proposition 7. *Mapping h is a consistent heuristic.*

3.3.2 From relaxations to distance under-approximations

To unify all three relaxations $\dashrightarrow$, $\twoheadrightarrow$ and $\dashrightarrow$ from Chapter 2, we sometimes use the notation $\longrightarrow_{\mathbb{G}}$ where $\mathbb{G} = \mathbb{Z}$ stands for $\dashrightarrow$, $\mathbb{G} = \mathbb{R}_{\geq 0}$ stands for $\twoheadrightarrow$, and $\mathbb{G} = \mathbb{R}$ stands for $\dashrightarrow\!\!\!\!\twoheadrightarrow$.

We write $\boldsymbol{m} \xrightarrow{\delta t}_{\mathbb{G}} \boldsymbol{m}'$ to emphasize the scaling factor δ, where $\delta = 1$ whenever $\mathbb{G} = \mathbb{Z}$.

Let $d_{\mathbb{G}} : \mathbb{N}^P \times \mathbb{N}^P \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be defined as $d_{\mathbb{G}}(\boldsymbol{m}, \boldsymbol{m}') \coloneqq \infty$ if $\boldsymbol{m} \overset{*}{\not\to}_{\mathbb{G}} \boldsymbol{m}'$, and otherwise:

$$d_{\mathbb{G}}(\boldsymbol{m}, \boldsymbol{m}') \coloneqq \min \left\{ \sum_{i=1}^{n} \delta_i \cdot \mu(t_i) : \boldsymbol{m} \xrightarrow{\delta_1 t_1 \cdots \delta_n t_n}_{\mathbb{G}} \boldsymbol{m}' \right\}.$$

In words, $d_{\mathbb{G}}(\boldsymbol{m}, \boldsymbol{m}')$ is the weight of a shortest path from $\boldsymbol{m}$ to $\boldsymbol{m}'$ in the graph induced by the relaxed step relation $\longrightarrow_{\mathbb{G}}$, where weights are scaled accordingly.

We now show that any $d_{\mathbb{G}}$, which we call the $\mathbb{G}$-*distance*, is a distance under-approximation, and first show effectiveness of all $d_{\mathbb{G}}$.

Proposition 8. *The functions $d_\mathbb{Z}$, $d_\mathbb{R}$, $d_{\mathbb{R}_{\geq 0}}$ are effective.*

Proof. By the marking equation, we have:

$$d_\mathbb{G}(\boldsymbol{m}, \boldsymbol{m}') = \min \left\{ \sum_{t \in T} \mu(t) \cdot \boldsymbol{\sigma}(t) : \boldsymbol{\sigma} \in \mathbb{G}_{\geq 0}^T, \boldsymbol{m}' = \boldsymbol{m} + \sum_{t \in T} \boldsymbol{\sigma}(t) \cdot \boldsymbol{\Delta}(t) \right\}.$$

Therefore, $d_\mathbb{R}(\boldsymbol{m}, \boldsymbol{m}')$ (resp. $d_\mathbb{Z}(\boldsymbol{m}, \boldsymbol{m}')$) are computable by (resp. integer) linear programming, which is is complete for P (resp. NP), in its variant where one must check whether the minimal solution is at most some bound.

Let us now prove the case of $d_{\mathbb{R}_{\geq 0}}$. As discussed in Section 2.2.2, $\xrightarrow{*}\!\!\!\!\rightarrow$ can be expressed in the existential fragment of linear real arithmetic, *i.e.* FO$\langle \mathbb{R}, +, < \rangle$, the first-order theory of the reals with addition and order. More precisely, there exists a linear-time computable formula $\psi \in \exists$FO$\langle \mathbb{R}, +, < \rangle$ such that $\psi(\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{m}')$ holds iff

there exists a sequence $\sigma \in T^\dagger$ s.t. $\boldsymbol{m} \xrightarrow{\sigma}\!\!\!\!\rightarrow \boldsymbol{m}'$ and $\boldsymbol{\sigma} = \boldsymbol{x}$.

Let $\Phi(\boldsymbol{m}, \boldsymbol{m}', \ell) := \exists \boldsymbol{x} \in \mathbb{R}_{\geq 0}^T : \psi(\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{m}') \wedge \ell = \sum_{t \in T} \mu(t) \cdot \boldsymbol{x}(t)$. Formula $\Phi \in \exists$FO$\langle \mathbb{R}, +, < \rangle$ can be constructed in linear time and is such that $\Phi(\boldsymbol{m}, \boldsymbol{m}', \ell)$ holds for $\boldsymbol{m}, \boldsymbol{m}' \in \mathbb{R}_{\geq 0}^P$ and $\ell \in \mathbb{R}_{\geq 0}$ iff $\ell = d_{\mathbb{R}_{\geq 0}}(\boldsymbol{m}, \boldsymbol{m}')$. Thus, $d_{\mathbb{R}_{\geq 0}}$ is computable as an instance of a decidable optimization modulo theories problem. □

Altogether, we conclude that $d_\mathbb{G}$ is a distance under-approximation. Furthermore, we can show that $d_\mathbb{G}$ yields *unbounded* heuristics, which, by Theorem 2, ensure termination of GBFS on reachable instances:

Theorem 3. *Let $\mathbb{G} \in \{\mathbb{Z}, \mathbb{R}, \mathbb{R}_{\geq 0}\}$, then $d_\mathbb{G}$ is a distance under-approximation. Moreover, the heuristics arising from it are unbounded.*

Proof. The first was part of the statement was fully shown in the main text. Let us prove the second part more formally. Let $\mathcal{N} = (P, T, F, \mu)$ be a weighted Petri net, let $\boldsymbol{m}_\text{tgt}$ be a target marking, and let $h_\mathbb{G}$ be the heuristic obtained from $d_\mathbb{G}$ for $\boldsymbol{m}_\text{tgt}$. Note that $h_\mathbb{R}(\boldsymbol{m}) \leq h_\mathbb{G}(\boldsymbol{m})$ for every marking $\boldsymbol{m}$ and every $\mathbb{G} \in \{\mathbb{Z}, \mathbb{R}, \mathbb{R}_{\geq 0}\}$. Hence, if $h_\mathbb{R}$ is unbounded, so are all three heuristics. Thus, it suffices to prove the case $\mathbb{G} = \mathbb{R}$.

For the sake of contradiction, suppose $h_\mathbb{R}$ is not unbounded. There exists $b \in \mathbb{R}_{\geq 0}$ and an infinite sequence of pairwise distinct markings $\boldsymbol{m}_0, \boldsymbol{m}_1, \ldots \in \mathbb{N}^P$ with $h_\mathbb{R}(\boldsymbol{m}_i) \leq b$ for every $i \geq 0$. Let $\boldsymbol{x}_i \in \mathbb{R}_{\geq 0}^T$ be a solution to the marking equation over $\mathbb{R}_{\geq 0}$ that yields $h_\mathbb{R}(\boldsymbol{m}_i)$, *i.e.* such that $h_\mathbb{R}(\boldsymbol{m}_i) = \sum_{t \in T} \mu(t) \cdot \boldsymbol{x}_i(t)$ is minimized subject to

$$\boldsymbol{m}_\text{tgt} = \boldsymbol{m}_i + \sum_{t \in T} \boldsymbol{x}_i(t) \cdot \boldsymbol{\Delta}(t). \tag{3.1}$$

By Dickson's lemma, there exist indices $i_0 < i_1 < \cdots$ such that $\boldsymbol{m}_{i_0} \leq \boldsymbol{m}_{i_1} \leq \cdots$. Since these markings are pairwise distinct, we may assume w.l.o.g.

the existence of a place $p \in P$ such that $\boldsymbol{m}_{i_0}(p) < \boldsymbol{m}_{i_1}(p) < \cdots$ (otherwise, we could extract such a subsequence).

Let us define the following constants:

$$c \coloneqq \min\{\mu(t) : t \in T\} \text{ and } d \coloneqq \frac{b \cdot |T| \cdot \max\{|\boldsymbol{\Delta}(t)(p)| : t \in T\}}{c}.$$

Let $j \geq 0$ be such that $\boldsymbol{m}_{\text{tgt}}(p) - \boldsymbol{m}_{i_j}(p) < -d$. Such an index j exists as p takes arbitrarily large values along our infinite sequence. By (3.1), we have:

$$\sum_{t \in T} \boldsymbol{x}_{i_j}(t) \cdot \boldsymbol{\Delta}(t)(p) = \boldsymbol{m}_{\text{tgt}}(p) - \boldsymbol{m}_{i_j}(p) < -d.$$

So, there exists $s \in T$ such that $\boldsymbol{\Delta}(s)(p) < 0$ and $\boldsymbol{x}_{i_j}(s) > b/c$. Indeed, if it was not the case, it would be impossible to obtain a negative value smaller than $-d$.

We are done since we obtain the following contradiction:

$$\begin{aligned}
h_{\mathbb{R}}(\boldsymbol{m}_{i_j}) &= \sum_{t \in T} \mu(t) \cdot \boldsymbol{x}_{i_j}(t) && \text{(by definition)} \\
&\geq \mu(s) \cdot \boldsymbol{x}_{i_j}(s) && \text{(by } \mu(t) > 0 \text{ and } \boldsymbol{x}_{i_j}(t) \geq 0 \text{ for each } t \in T) \\
&> \mu(s) \cdot (b/c) && \text{(by } \mu(s) > 0 \text{ and } \boldsymbol{x}_{i_j}(s) > b/c) \\
&\geq \mu(s) \cdot (b/\mu(s)) && \text{(by } \mu(s) \geq c) \\
&= b \\
&\geq h_{\mathbb{R}}(\boldsymbol{m}_{i_j}) && \text{(by boundedness).}
\end{aligned}$$

□

3.3.3 Directed reachability using distance under-approximations

We have all the ingredients to use Algorithm 3 for answering reachability queries.

A *distance under-approximation scheme* is a mapping $\mathcal{D}$ that associates a distance under-approximation $\mathcal{D}(\mathcal{N})$ to each weighted Petri net $\mathcal{N}$. Let $h_{\mathcal{D}(\mathcal{N}),\boldsymbol{m}_{\text{tgt}}}$ be the heuristic obtained from $\mathcal{D}(\mathcal{N})$ for marking $\boldsymbol{m}_{\text{tgt}}$. By instantiating Algorithm 3 with this heuristic, we can search for a short(est) firing sequence witnessing that $\boldsymbol{m}_{\text{tgt}}$ is reachable. Of course, constructing the reachability graph of $\mathcal{N}$ would be at least as difficult as answering this query, or impossible if it is infinite. Hence, we provide $G_{\mathbb{N}}(\mathcal{N})$ *symbolically* through $\mathcal{N}$ and let Algorithm 3 explore it on-the-fly by progressively firing its transitions.

For each $\mathbb{G} \in \{\mathbb{Z}, \mathbb{R}, \mathbb{R}_{\geq 0}\}$, the function $\mathcal{D}_{\mathbb{G}}$ mapping a weighted Petri net $\mathcal{N}$ to its $\mathbb{G}$-distance $d_{\mathbb{G}}$ is a distance under-approximation scheme with consistent and unbounded heuristics by Proposition 7, Theorem 2 and Theorem 3. Although Algorithm 3 is geared towards finding paths, it can prove *non*-reachability even for infinite reachability graphs. Indeed, at some point, every candidate marking $\boldsymbol{m} \in C$ may be such that $h_{\mathcal{D}(\mathcal{N}),\boldsymbol{m}_{\text{tgt}}}(\boldsymbol{m}) = \infty$, which halts with ∞. There is no guarantee that this happens, but, as reported *e.g.* by [ELM+14, BFHH17], the $\mathbb{G}$-distance for domains $\mathbb{G} \in \{\mathbb{Z}, \mathbb{R}, \mathbb{R}_{\geq 0}\}$ does well for witnessing non-reachability in practice, often from the very first marking $\boldsymbol{m}_{\text{src}}$.

Example.

We illustrate the approach with a toy example and $\mathcal{D}_{\mathbb{R}}$, *i.e.* the scheme based on $\dashrightarrow\!\!\!\!\!\!\rightarrow$. Consider the Petri net $\mathcal{N}$ illustrated on the left of Figure 3.1 marked with $\boldsymbol{m}_{\text{src}} := (0, 0)$. Suppose we wish to determine whether $\boldsymbol{m}_{\text{src}}$ can reach marking $\boldsymbol{m}_{\text{tgt}} := (0, 1)$ in $\mathcal{N}$.

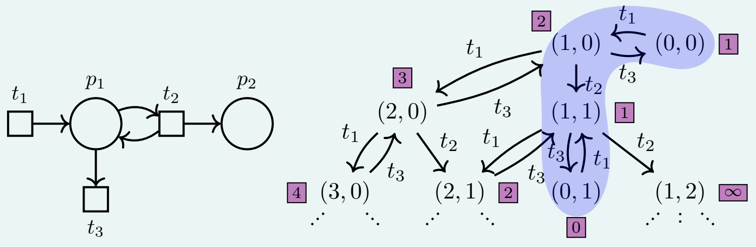


Figure 3.1: *Left:* A Petri net $\mathcal{N}$. *Right:* Search of Algorithm 3 over the graph $G_{\mathbb{N}}(\mathcal{N})$ from $(0, 0)$ to $(0, 1)$, where each number in a box next to a marking is its heuristic value. Only the colored region is expanded.

We consider the case where Algorithm 3 follows a greedy best-first search, but the markings would be expanded in the same way with A*.

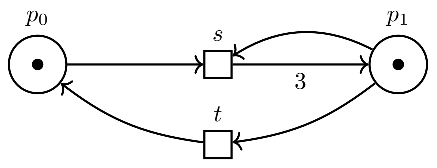
Since $\boldsymbol{\Delta}(t_2) = (0, 1)$, the heuristic considers that $\boldsymbol{m}_{\text{src}}$ can reach $\boldsymbol{m}_{\text{tgt}}$ in a single step using transition t_2 (it is unaware of the guard). Marking $(1, 0)$ is expanded and its heuristic value increases to 2 as the marking equation considers that both t_2 and t_3 must be fired (in some unknown order).

Markings $(2, 0)$ and $(1, 1)$ are both discovered with respective heuristic values 3 and 1. The latter is more promising, so it is expanded and target $(0, 1)$ is discovered. Since its heuristic value is 0, it is immediately expanded and the correct distance $\text{dist}_{\mathcal{N}}(\boldsymbol{m}_{\text{src}}, \boldsymbol{m}_{\text{tgt}}) = 3$ is returned.

Note that, in this example, the only markings expanded are precisely those occurring on the shortest path.

3.4 Exercices

3.1) Execute A* on this Petri net to decide whether $(1,1)$ can reach $(3,0)$: **?**



3.2) Let d be a distance under-approximation and let $h\colon \mathbb{N}^P \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be defined by $h(\boldsymbol{m}) := d(\boldsymbol{m}, \boldsymbol{m}_{\text{tgt}})$. Show that h is a consistent heuristic. **?**

3.3) Given an example of a marked Petri net and a consistent heuristic where greedy best-first search never finds the reachable target marking.

3.4) Explain how to use Algorithm 3 for Petri net coverability rather than Petri net reachability. **?**

3.5) Give a Petri net $\mathcal{N} = (P, F, T)$ and markings $\boldsymbol{m}_{\text{src}}, \boldsymbol{m}_{\text{tgt}} \in \mathbb{N}^P$ such that **?**

$$\boldsymbol{m}_{\text{src}} \xrightarrow{*}\!\!\!\!\twoheadrightarrow \boldsymbol{m}_{\text{tgt}},$$

and where A*and GBFS (correctly) conclude that $\boldsymbol{m}_{\text{src}} \not\xrightarrow{*} \boldsymbol{m}_{\text{tgt}}$ using the heuristic based on $\twoheadrightarrow$.

3.6) Give a Petri net $\mathcal{N} = (P, F, T)$ and markings $\boldsymbol{m}_{\text{src}}, \boldsymbol{m}_{\text{tgt}} \in \mathbb{N}^P$ such that **?**

$$\boldsymbol{m}_{\text{src}} \overset{*}{\dashrightarrow} \boldsymbol{m}_{\text{tgt}},$$

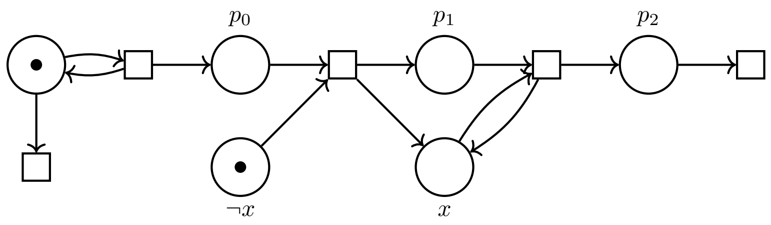
and where A*and GBFS (correctly) conclude that $\boldsymbol{m}_{\text{src}} \not\xrightarrow{*} \boldsymbol{m}_{\text{tgt}}$ using the heuristic based on $\dashrightarrow$.

Solutions

Chapter 1

1.1)

a)

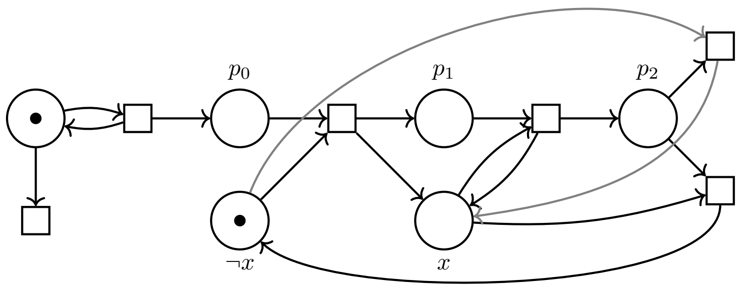


b) $\exists \boldsymbol{m}'$ such that

$$\begin{pmatrix}1 & 0 & 0 & 0\\ & 1 & 0 & \end{pmatrix} \xrightarrow{\;*\;} \boldsymbol{m}' \text{ and } \boldsymbol{m}' \geq \begin{pmatrix}0 & 0 & 0 & 2\\ & 0 & 0 & \end{pmatrix}.$$

1.2)

a)



b)

$$\begin{pmatrix}1 & 0 & 0 & 0\\ & 1 & 0 & \end{pmatrix} \xrightarrow{\;*\;} \begin{pmatrix}0 & 0 & 0 & 0\\ & 0 & 1 & \end{pmatrix}.$$

1.3)

$$\begin{tikzcd}
 & \boxed{1,0,0,0} \arrow[dl, bend right, "t_2"'] \arrow[dr, bend left, "t_3"] & & \\
\boxed{0,1,0,0} & & \boxed{0,0,1,1} \arrow[d, "t_1"] & \\
 & & \boxed{1,0,0,\omega} \arrow[dl, bend right, "t_2"'] \arrow[dr, bend left, "t_3"] & \\
 & \boxed{0,1,0,\omega} \arrow[ur, bend right, "t_4"'] & & \boxed{0,0,1,\omega} \arrow[ul, bend left, "t_1"]
\end{tikzcd}$$

1.4)

Iter.	**Basis** B	**Predecessors**	
0	$\{(0,1,1)\}$	$(0,1,1)_{t_1} = (0,1,1)$ $(0,1,1)_{t_2} = (1,2,0)$ $(0,1,1)_{t_3} = (0,0,2)$	
1	$\{(0,1,1), (1,2,0), (0,0,2)\}$	$(1,2,0)_{t_1} = (0,2,0)$ $(1,2,0)_{t_2} = (2,3,0)$ $(1,2,0)_{t_3} = (1,1,1)$	$(0,0,2)_{t_1} = (0,1,2)$ $(0,0,2)_{t_2} = (1,1,1)$ $(0,0,2)_{t_3} = (0,0,3)$
2	$\{(0,1,1), (0,2,0), (0,0,2)\}$	$(0,2,0)_{t_1} = (0,2,0)$ $(0,2,0)_{t_2} = (1,3,0)$ $(0,2,0)_{t_3} = (0,1,1)$	
3	$\{(0,1,1), (0,2,0), (0,0,2)\}$	basis unchanged	

1.5) A depth-first firing of t_2, t_3 and t_1 leads to five nodes. The firing of t_2, t_1 and t_3 leads to four nodes.

1.6) Since $\boldsymbol{k} \geq \boldsymbol{m}$, there exists $\boldsymbol{d} \in \mathbb{N}^P$ such that $\boldsymbol{k} = \boldsymbol{m} + \boldsymbol{d}$. We have $\boldsymbol{k}(p) \geq \boldsymbol{m}(p) \geq F(p,t)$ for each place p, and hence t is firable in $\boldsymbol{k}$. Let $\boldsymbol{k}' := \boldsymbol{m}' + \boldsymbol{d}$. We have:

$$\boldsymbol{k} = (\boldsymbol{m} + \boldsymbol{d}) \xrightarrow{t} (\boldsymbol{m}' + \boldsymbol{d}) = \boldsymbol{k}' \geq \boldsymbol{m}'. \qquad \square$$

Chapter 2

2.1) For $\boldsymbol{m}_{\text{src}} \overset{*}{\dashrightarrow} \boldsymbol{m}_{\text{tgt}}$:

a) Yes via $t_2 t_4$.

b) Yes via $t_1 t_3$.

c) No.

The marking equation amounts to:

$$\begin{pmatrix}2\\0\\0\\0\end{pmatrix} + \begin{pmatrix}-1 & -2 & -1 & 0\\ 1 & 0 & -1 & 0\\ 0 & 1 & 1 & -1\\ 0 & 0 & 0 & 1\end{pmatrix}\begin{pmatrix}t_1\\t_2\\t_3\\t_4\end{pmatrix} = \begin{pmatrix}1\\0\\1\\0\end{pmatrix}$$

By summing the first two equations and dividing by -2, we get $t_2 + t_3 = 1/2$. As the sum of two integers cannot be rational, there is no integral solution to the marking equation.

For $\boldsymbol{m}_{\text{src}} \overset{*}{\twoheadrightarrow} \boldsymbol{m}_{\text{tgt}}$:

a) Yes:

$$(2,0,0,0) \xrightarrow{(1/2)t_1\ (1/2)t_3\ (1/2)t_4\ (1/2)t_2\ (1/2)t_4} (0,0,0,1).$$

b) No, since the maximal backward firing set is empty.

c) No.

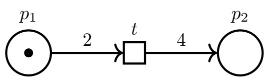
The maximal forward firing set is $X \coloneqq \{t_1, t_2, t_3, t_4\}$. The maximal backward firing set is $Y \coloneqq \{t_1, t_3\}$. Since $X \cap Y = \{t_1, t_3\}$, we must check the marking equation with support at most $\{t_1, t_3\}$:

$$\begin{aligned} 2 - t_1 - t_3 &= 1,\\ 0 + t_1 - t_3 &= 0,\\ 0 + t_3 &= 1,\\ 0 &= 0. \end{aligned}$$

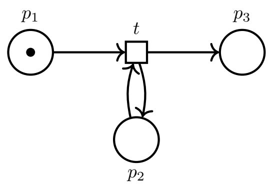
From the second and third equations, we get $t_1 = t_3 = 1$. Using the first equation, we obtain the contradiction $0 = 1$.

2.3)

- In this Petri net, we have $(1,0) \overset{*}{\twoheadrightarrow} (0,2)$ and $(1,0) \overset{*}{\not\dashrightarrow} (0,2)$:



- In this Petri net, we have $(1,0,0) \overset{*}{\not\twoheadrightarrow} (0,0,1)$ and $(1,0,0) \overset{*}{\dashrightarrow} (0,0,1)$:



2.6) This has been shown in [DL15] with a dedicated algorithm. Yet, the result can be recast in the logical framework of continuous reachability.

First, note that $\mathbf{0} \xrightarrow{+} \mathbf{0}$ iff $\boldsymbol{m} \xrightarrow{+} \boldsymbol{m}$ for every marking $\boldsymbol{m}$. Moreover,

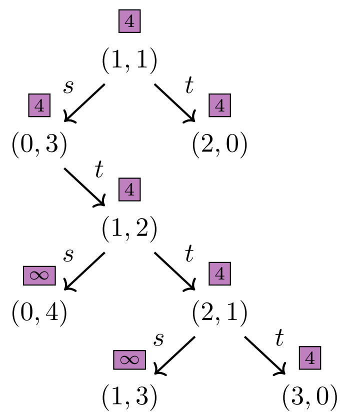
$$\mathbf{0} \xrightarrow{+} \mathbf{0} \iff \mathbf{0} \overset{+}{\twoheadrightarrow} \mathbf{0},$$

and the latter amounts to this formula from $\mathcal{L}$:

$$\exists \boldsymbol{x} \in \mathbb{R}_{\geq 0}^T : \varphi_{\text{mark-eq}}(\mathbf{0}, \boldsymbol{x}, \mathbf{0}) \land \varphi_{\blacktriangleright\blacktriangleright}(\mathbf{0}, \boldsymbol{x}) \land \varphi_{\blacktriangleleft\blacktriangleleft}(\boldsymbol{x}, \mathbf{0}) \land \bigvee_{t \in T} \boldsymbol{x}(t) > 0.$$

Chapter 3

3.1)



3.2) Let $\boldsymbol{m}, \boldsymbol{m}' \in \mathbb{N}^P$ and $t \in T$ be such that $\boldsymbol{m} \xrightarrow{t} \boldsymbol{m}'$. We have:

$$\begin{aligned}
h(\boldsymbol{m}) &= d(\boldsymbol{m}, \boldsymbol{m}_{\text{tgt}}) && \text{(by def. of } h\text{)} \\
&\leq d(\boldsymbol{m}, \boldsymbol{m}') + d(\boldsymbol{m}', \boldsymbol{m}_{\text{tgt}}) && \text{(by the triangle inequality)} \\
&\leq \text{dist}_{\mathcal{N}}(\boldsymbol{m}, \boldsymbol{m}') + d(\boldsymbol{m}', \boldsymbol{m}_{\text{tgt}}) && \text{(by distance under-approx.)} \\
&\leq \mu(t) + d(\boldsymbol{m}', \boldsymbol{m}_{\text{tgt}}) && \text{(since } \boldsymbol{m} \xrightarrow{t} \boldsymbol{m}'\text{)} \\
&= \mu(t) + h(\boldsymbol{m}') && \text{(by def. of } h\text{).}
\end{aligned}$$

Moreover, $h(\boldsymbol{m}_{\text{tgt}}) = d(\boldsymbol{m}_{\text{tgt}}, \boldsymbol{m}_{\text{tgt}}) \leq \text{dist}_{\mathcal{N}}(\boldsymbol{m}_{\text{tgt}}, \boldsymbol{m}_{\text{tgt}}) = 0$, where the last equality follows from the fact that weights are positive. □

3.4) Algorithm 3 can be adapted to search for *some* marking from a given target set $X \subseteq \mathbb{N}^P$. The idea consists simply in using a heuristic $h_X : \mathbb{N}^P \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ estimating the weight of a shortest path to *any* target:

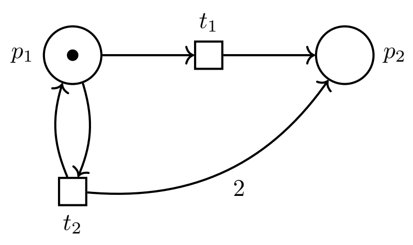
$$h_X(\boldsymbol{m}) \coloneqq \min\{h_{\mathcal{D}(\mathcal{N}), \boldsymbol{m}_{\text{tgt}}}(\boldsymbol{m}) : \boldsymbol{m}_{\text{tgt}} \in X\}.$$

This is convenient for partial reachability instances occurring in practice, *i.e.*

$X \coloneqq \left\{\boldsymbol{m}_{\text{tgt}} \in \mathbb{N}^P : \boldsymbol{m}_{\text{tgt}}(p) \sim_p \boldsymbol{c}(p)\right\}$ where $\boldsymbol{c} \in \mathbb{N}^P$ and each $\sim_p \in \{=, \geq\}$.

This includes the case of coverability where each $\sim_p$ is $\geq$.

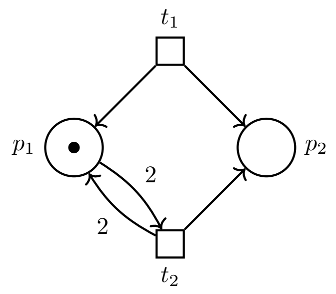
3.5) Let $\mathcal{N}$ be the Petri net below. Let $\boldsymbol{m}_{\text{src}} \coloneqq (1, 0)$ and $\boldsymbol{m}_{\text{tgt}} \coloneqq (0, 2)$.



GBFS constructs this graph:

$$\begin{tikzcd}
& \overset{\boxed{3/2}}{(1,0)} \arrow[dl, "t_2"'] \arrow[dr, "t_1"] & \\
\boxed{\infty}\ (0,1) & & (1,2)\ \boxed{\infty}
\end{tikzcd}$$

3.6) Let $\mathcal{N}$ be the Petri net below. Let $\boldsymbol{m}_{\text{src}} \coloneqq (0,0)$ and $\boldsymbol{m}_{\text{tgt}} \coloneqq (1,2)$.



GBFS constructs this graph:

$$\begin{tikzcd}
\overset{\boxed{2}}{(0,0)} \arrow[r, "t_1"] & \overset{\boxed{1}}{(1,1)} \arrow[r, "t_1"] & \overset{\boxed{\infty}}{(2,2)}
\end{tikzcd}$$

Bibliography

[ACDE07] David Applegate, William J. Cook, Sanjeeb Dash, and Daniel G. Espinoza. Exact solutions to linear programming problems. *Operations Research Letters*, 35(6):693–699, 2007.

[ACJT00] Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1-2):109–127, 2000.

[BCD$^+$11] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovi'c, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Proc. 23rd International Conference on Computer Aided Verification (CAV)*, pages 171–177, 2011.

[BE22] Michael Blondin and Javier Esparza. Separators in continuous petri nets. In *Proc. 25th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS)*, pages 81–100, 2022.

[BFHH17] Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. The logical view on continuous Petri nets. *ACM Transactions on Computational Logic (TOCL)*, 18(3):24:1–24:28, 2017.

[BG11] Laura Bozzelli and Pierre Ganty. Complexity analysis of the backward coverability algorithm for VASS. In *Proc. 5th International Workshop on Reachability Problems (RP)*, pages 96–109, 2011.

[BH17] Michael Blondin and Christoph Haase. Logics for continuous reachability in Petri nets and vector addition systems with states. In *Proc. 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017.

[BHO21] Michael Blondin, Christoph Haase, and Philip Offtermatt. Directed reachability for infinite-state systems. In *Proc. 27th International*

Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pages 3–23, 2021.

[Blo20] Michael Blondin. The ABCs of Petri net reachability relaxations. *ACM SIGLOG News*, 7(3), 2020.

[CLL$^+$19] Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In *Proc.* 51^{st} *Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 24–33, 2019.

[CO21] Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2021. à paraître.

[DA87] René David and Hassane Alla. Continuous Petri nets. In *Proc. 8th European Workshop on Application and Theory of Petri nets*, volume 340, pages 275–294, 1987.

[DA10] René David and Hassane Alla. *Discrete, Continuous, and Hybrid Petri nets*. Springer, 2 edition, 2010.

[DL15] Frank Drewes and Jérôme Leroux. Structurally cyclic Petri nets. *Logical Methods in Computer Science (LMCS)*, 11(4), 2015.

[DLV19] Alin Deutsch, Yuliang Li, and Victor Vianu. Verification of hierarchical artifact systems. *ACM Transactions on Database Systems (TODS)*, 44(3):12:1–12:68, 2019.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proc.* 14^{th} *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 337–340, 2008.

[EGLM17] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017.

[ELM$^+$14] Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp J. Meyer, and Filip Nikšić. An SMT-based approach to coverability analysis. In *Proc.* 26^{th} *International Conference on Computer Aided Verification (CAV)*, volume 8559, pages 603–619, 2014.

[FH15] Estíbaliz Fraca and Serge Haddad. Complexity analysis of continuous Petri nets. *Fundamenta Informaticae*, 137(1):1–28, 2015.

[FKP14] Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proofs that count. In *Proc. 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 151–164. ACM, 2014.

[FMW$^+$17] Yu Feng, Ruben Martins, Yuepeng Wang, Işil Dillig, and Thomas W. Reps. Component-based synthesis for complex APIs. In *Proc. 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 599–612. ACM, 2017.

[GJJ$^+$20] Zheng Guo, Michael James, David Justo, Jiaxiao Zhou, Ziteng Wang, Ranjit Jhala, and Nadia Polikarpova. Program synthesis by type-guided abstraction refinement. *Proc. ACM on Programming Languages (POPL)*, 4(12):1–28, 2020.

[GLS18] Thomas Geffroy, Jérôme Leroux, and Grégoire Sutre. Occam's razor applied to the Petri net coverability problem. *Theoretical Computer Science*, 750:38–52, 2018.

[GM12] Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 34(1):6:1–6:48, 2012.

[GS92] Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.

[Kos82] S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proc.* 14th *Symposium on Theory of Computing (STOC)*, pages 267–281, 1982.

[Lam92] Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theoretical Computer Science*, 99(1):79–104, 1992.

[Ler12] Jérôme Leroux. Vector addition systems reachability problem (A simpler solution). In *Turing-100 – The Alan Turing Centenary*, pages 214–228, 2012.

[Ler21] Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2021. à paraître.

[Lip76] Richard J. Lipton. The reachability problem requires exponential space. Technical Report 63, Department of Computer Science, Yale University, 1976.

[May81] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc.* 13th *Symposium on Theory of Computing (STOC)*, pages 238–246, 1981.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[Rac78] Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6:223–231, 1978.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition, 2009.

[ST77] George S. Sacerdote and Richard L. Tenney. The decidability of the reachability problem for vector addition systems (preliminary version). In *Proc. 9th Symposium on Theory of Computing (STOC)*, pages 61–76, 1977.

[VSS05] Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational horn clauses. In *Proc. 20th International Conference on Automated Deduction on Automated Deduction*, pages 337–352, 2005.