1 Labelled Transition System

A Labelled Transition System (LTS) is a tuple $(\mathcal{S}, A, \rightarrow)$ where $\mathcal{S}$ is a set of states, A is a set of actions (or labels), and $\rightarrow \subseteq \mathcal{S} \times A \times \mathcal{S}$ is a transition relation. Whenever $(s, a, s') \in \rightarrow$, we write $s \xrightarrow{a} s'$ and say that a is enabled in s, and we can *execute* a in s yielding s'. Otherwise we say that a is disabled in s and write $s \stackrel{a}{\nrightarrow}$. The set of all enabled actions in a state s is denoted $en(s)$. A state s is said to be a *deadlock* if $en(s) = \emptyset$. For a possibly infinite sequence of actions $w = a_1 a_2 \cdots \in A^* \cup A^\omega$ and states $s_1, s_2, \ldots$ we call w an *action sequence* if $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$. If w is finite then this is written as $s_1 \xrightarrow{w} s_n$. By convention $s \xrightarrow{\varepsilon} s$ always holds, where ε is the empty action sequence. Any action sequence of length n from s to s' is written as $s \rightarrow^n s'$. If there exists an action sequence $w \in A^*$ such that $s \xrightarrow{w} s'$, we write $s \rightarrow^* s'$. The set of all reachable states from a state s is given by the set $reach(s) = \{s' \mid s \rightarrow^* s'\}$. The sequence of states induced by an action sequence is called a *path* and is written as $\pi = s_1 s_2 \cdots$. We use $\Pi(s)$ to denote the set of all paths starting from a state s, and $\Pi = \bigcup_{s \in \mathcal{S}} \Pi(s)$ is the set of all paths. The length of a path is given by the function $\ell : \Pi \rightarrow \mathbb{N} \cup \{\infty\}$. A position i in a path $\pi \in \Pi$ refers to state s_i in the path and is written as π_i. If π is infinite then $i \in \mathbb{N}$, otherwise $1 \leq i \leq \ell(\pi)$. We use $\Pi^{max}(s)$ to denote the set of all maximal paths starting from a state s which is defined as $\Pi^{max}(s) = \{\pi \in \Pi(s) \mid \ell(\pi) = \infty \text{ or } \pi_{\ell(\pi)} \text{ is a deadlock}\}$.

2 Computation Tree Logic

Let AP be a set of atomic propositions, $a \in AP$ an atomic proposition, and $(\mathcal{S}, A, \rightarrow)$ an LTS. We evaluate atomic propositions using the function $v : \mathcal{S} \rightarrow 2^{AP}$, where $v(s)$ is the set of atomic propositions satisfied in the state $s \in \mathcal{S}$. The CTL syntax and semantics are given as follows:

$$\begin{aligned}\varphi ::= \; & \mathit{true} \mid \mathit{false} \mid a \mid \mathit{deadlock} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \varphi_1 \Longrightarrow \varphi_2 \mid \varphi_1 \Longleftrightarrow \varphi_2 \\ & \mid AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A(\varphi_1 U \varphi_2) \mid E(\varphi_1 U \varphi_2)\end{aligned}$$

The semantics of formula φ is defined for a state $s \in \mathcal{S}$ as follows:

$s \models true$	
$s \not\models false$	
$s \models a$	iff $a \in v(s)$
$s \models deadlock$	iff $en(s) = \emptyset$
$s \models \varphi_1 \wedge \varphi_2$	iff $s \models \varphi_1$ and $s \models \varphi_2$
$s \models \varphi_1 \vee \varphi_2$	iff $s \models \varphi_1$ or $s \models \varphi_2$
$s \models \neg\varphi$	iff $s \not\models \varphi$
$s \models \varphi_1 \implies \varphi_2$	iff $s \not\models \varphi_1$ or $s \models \varphi_2$
$s \models \varphi_1 \iff \varphi_2$	iff ($s \models \varphi_1$ iff $s \models \varphi_2$)
$s \models AX\varphi$	iff for all $s' \in \mathcal{S}$ if $s \to s'$ then $s' \models \varphi$
$s \models EX\varphi$	iff exists $s' \in \mathcal{S}$ s.t $s \to s'$ and $s' \models \varphi$
$s \models AG\varphi$	iff for all $\pi \in \Pi^{max}(s)$ and for all positions i in π we have $\pi_i \models \varphi$
$s \models EF\varphi$	iff exists $\pi \in \Pi^{max}(s)$ s.t. there exists a position i in π s.t. $\pi_i \models \varphi$
$s \models AF\varphi$	iff for all $\pi \in \Pi^{max}(s)$ there exists a position i in π s.t. $\pi_i \models \varphi$
$s \models EG\varphi$	iff exists $\pi \in \Pi^{max}(s)$ s.t. for all positions i in π we have $\pi_i \models \varphi$
$s \models A(\varphi_1 U \varphi_2)$	iff for all $\pi \in \Pi^{max}(s)$ there exists a position i in π s.t. $\pi_i \models \varphi_2$ and for all $1 \leq j < i$ we have $\pi_j \models \varphi_1$
$s \models E(\varphi_1 U \varphi_2)$	iff exists $\pi \in \Pi^{max}(s)$ and there exists a position i in π s.t. $\pi_i \models \varphi_2$ and for all $1 \leq j < i$ we have $\pi_j \models \varphi_1$

We use Φ_{CTL} to denote the set of all CTL formulae.

3 Atomic Propositions for Petri Net CTL

The satisfiability of CTL formulae in a Petri net is interpreted on the LTS generated by the net. We fix the set of atomic propositions AP based on the informal semantics in the MCC Property Language, which includes arithmetic expressions and fireability of transitions. Let $N = (P, T, W, I)$ be a Petri net. An atomic proposition $a \in AP$ is defined as:

$a ::= \; t \mid e_1 \bowtie e_2$

$e ::= \; c \mid p \mid e_1 \oplus e_2$

where $t \in T$, $c \in \mathbb{N}^0$, $\bowtie \in \{<, \leq, =, \neq, >, \geq\}$, $p \in P$, and $\oplus \in \{+, -, *\}$. The semantics of φ is defined for a marking M as follows:

$$
\begin{array}{ll}
M \models t & \text{iff } t \in en(M) \\
M \models e_1 \bowtie e_2 & \text{iff } eval_M(e_1) \bowtie eval_M(e_2)
\end{array}
$$

The semantics of an arithmetic expression in a marking M is given as follows:

$eval_M(c) = c$,
$eval_M(p) = M(p)$,
$eval_M(e_1 \oplus e_2) = eval_M(e_1) \oplus eval_M(e_2)$.

We use $\Phi_{Reach} \subseteq \Phi_{CTL}$ to denote a subset of formulae called *reachability* formulae. Reachability formulae can be on the form $EF\varphi$ or $AG\varphi$, where φ is defined as follows:

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid a \mid \mathit{deadlock} \mid e_1 \bowtie e_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \varphi_1 \implies \varphi_2 \mid \varphi_1 \iff \varphi_2$$

A reachability formula $AG\varphi$ is equivalent to $\neg EF\neg\varphi$. Henceforth, we assume all $AG\varphi$ reachability formulae have been transformed to EF formulae.

4 Integer Linear Program

For defining an integer linear program, we first need to define a linear equation. Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of variables and $\overline{x}$ a column vector over the variables X such that:

$$\overline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

A linear equation is given by $\overline{c}\ \overline{x} \bowtie k$, where $\bowtie \in \{=, <, \leq, >, \geq\}$, $k \in \mathbb{Z}$ is an integer, and $\overline{c}$ is a row vector of integers such that:

$$\overline{c} = \begin{bmatrix} c_1\ c_2 \cdots c_n \end{bmatrix} \quad \text{where } c_i \in \mathbb{Z} \text{ for all } 1 \leq i \leq n.$$

Definition 1 (Integer Linear Program). *An integer linear program $LP = \{\overline{c}_1\overline{x} \bowtie_1 k_1, \overline{c}_2\overline{x} \bowtie_2 k_2, \cdots, \overline{c}_m\overline{x} \bowtie_m k_m\}$ is a set of linear equations. A solution to LP is a mapping $u : X \to \mathbb{N}^0$ from variables to natural numbers and corresponding column vector $\overline{u}^T = [u(x_1)\ u(x_2) \cdots u(x_n)]$, such that for all $1 \leq i \leq m$ we have $\overline{c}_i\overline{u} \bowtie_i k_i$ is true. We use $\mathcal{E}^X_{lin}$ to denote the set of all linear programs over a set of variables X.*

Let $N = (P, T, W, I)$ be a Petri net, $M_0 \in \mathcal{M}(N)$ an initial marking on N, $M \in \mathcal{M}(N)$ a marking on N, and $X = \{x_t \mid t \in T\}$ a set of variables. From this we construct the following linear program over X:

$$M_0(p) + \sum_{t \in T}(W(t,p) - W(p,t))x_t = M(p) \quad \text{for all } p \in P.$$

It is well-known that if $M \in \mathit{reach}(M_0)$ then there exists a solution to the linear program. If the linear program is infeasible, then we can discern that $M \notin \mathit{reach}(M_0)$.

5 Reductions of LTS

A reduction is a function from the set of states to the power set of actions, such that for each state the function returns the set of required actions.

Definition 2 (Reduction). *Let $T = (\mathcal{S}, A, \rightarrow)$ be an LTS. A reduction of T is a function $St : \mathcal{S} \rightarrow 2^A$.*

A reduction defines a subset of the transition relation of an LTS, and we annotate the transition relation with a reduction to define the reduced state space.

Definition 3 (Reduced transition relation). *Let $T = (\mathcal{S}, A, \rightarrow)$ be an LTS and St a reduction of T. A reduced transition relation is a relation $\underset{St}{\longrightarrow} \subseteq \rightarrow$ such that $s \underset{St}{\xrightarrow{a}} s'$ iff $s \xrightarrow{a} s'$ and $a \in St(s)$.*

Let $T = (\mathcal{S}, A, \rightarrow)$ be an LTS, $a \in \mathcal{S}$ a state, and St a reduction of T. The set $\overline{St(s)} = A \setminus St(s)$, is the set of all actions not in $St(s)$.

For a sequences of actions, the following condition identifies required actions, that allow us to permute the sequence, such that the permuted sequence begins with the required action.

W For all $s \in \mathcal{S}$, all $a \in St(s)$, and all $w \in \overline{St(s)}^*$, if $s \xrightarrow{wa} s'$ then $s \xrightarrow{aw} s'$.

Reductions that satisfy **W** are called *(weak)semistubborn* reductions, and for all $s \in \mathcal{S}$, we say that $St(s)$ is the *stubborn set* of s, and that an action $a \in St(s)$ is a *stubborn action.*

Lemma 1. *Let $T = (\mathcal{S}, A, \rightarrow)$ be an LTS and St be a reduction on T satisfying **W**. For all $s \in \mathcal{S}$, all $a \in St(s)$, and all $w \in \overline{St(s)}^*$, if $a \notin en(s)$ and $s \xrightarrow{w} s'$ then $a \notin en(s')$.*

5.1 Reachability Preserving Stubborn Reduction

When performing reachability analysis, we are searching for states that satisfy a given property. In the context of stubborn reduction, we refer to these states as goal states.

Let $T = (\mathcal{S}, A, \rightarrow)$ be an LTS, $s_0 \in \mathcal{S}$ an initial state, and $G \subseteq \mathcal{S}$ a set of goal states. For a reduction St to preserve paths to a goal state, the following condition needs to be satisfied:

R For all $s \in \mathcal{S}$ if $s \notin G$ and $s \xrightarrow{w} s'$ where $w \in \overline{St(s)}^*$ then $s' \notin G$.

Rule **R** states that, when starting in a non-goal state, the execution of non-stubborn transitions cannot reach any goal state in G. It also ensures that at least one stubborn action has to be executed in order to reach a goal state.

Theorem 1 (Reachability preservation). *Let $(\mathcal{S}, A, \rightarrow)$ be an LTS, $G \subseteq S$ a set of goal states, and $s_0 \in \mathcal{S}$. Let St be a reduction satisfying **W** and **R**. If $s_0 \rightarrow^n s$ where $s \in G$ then $s_0 \underset{St}{\longrightarrow}^m s'$ where $s' \in G$ and $m \leq n$. If $s_0 \underset{St}{\longrightarrow}^m s$ where $s \in G$ then $s_0 \rightarrow^m s$.*

6 Reductions of Petri Net

Instead of states and actions of LTS, we now refer to markings and transitions of Petri nets. We define goal states as goal markings that satisfy a given reachability property. Let $EF\ \varphi \in \Phi_{Reach}$ be a reachability formula and $G_\varphi = \{M \in \mathcal{M}(N) \mid M \models \varphi\}$ be the goal markings for φ, where N is a Petri net. The reduction procedure must identify transitions that are required to fire in order to reach the goal markings. All transitions that can alter the truth value of φ from *false* to *true* are interesting transitions. The interesting transitions of a marking M and formula φ, denoted $A_M(\varphi)$.

Assume $M \not\models \varphi$ and $t \in T$. Let $A_M(\varphi) \subseteq T$ such that if $M \xrightarrow{t} M'$ and $M' \models \varphi$ then $t \in A_M(\varphi)$. We define $A_M(\varphi)$ recursively on the syntactic category for reachability formulae. The interesting transitions for all Boolean formulae are shown in Table 1. The interesting transitions of a negation depend on what follows syntactically from the negation, and thus we describe this in a separate column. Table 1 does not describe $A_M(\neg\neg\varphi)$ because its set of interesting transitions is equivalent to that of $A_M(\varphi)$. We introduce the notation $\overline{\bowtie}$ that refers to the complement of of a comparison operator $\bowtie$. The complement operators are shown in Table 2.

We define the set of expressions that can be constructed with N as E_N, and two functions $incr_M : E_N \to 2^T$ and $decr_M : E_N \to 2^T$. These functions receive an expression e and return the set of transitions that, when fired, increase and decrease the evaluation of e, respectively. We present the interesting transitions for formulae of the form $e_1 \bowtie e_2$ in Table 3. We recursively define $incr_M$ and $decr_M$ on the syntax of expressions in Table 4.

Lemma 2. *Let $N = (P, T, W, I)$ be a Petri net, $M \in \mathcal{M}(N)$ a marking, and $EF\varphi \in \Phi_{Reach}$ a reachability formula. If $M \not\models \varphi$ and $M \xrightarrow{t'} M'$ where $t' \notin A_M(\varphi)$ then $M' \not\models \varphi$.*

Lemma 3. *Let $N = (P, T, W, I)$ be a Petri net, $M \in \mathcal{M}(N)$ a marking, φ a formula, and $w \in \overline{A_M(\varphi)}^*$ a sequence of non-interesting transitions. If $M \notin G_\varphi$ and $M \xrightarrow{w} M'$ then $M' \notin G_\varphi$.*

We can easily verify the **R** property by including all interesting transitions in the stubborn set. Ensuring the **W** property is done by examining the structure of the Petri net and the marking in question.

Proposition 1 (Reachability preserving closure for Petri nets). *Let $N = (P, T, W, I)$ be a Petri net with inhibitor arcs, $EF\varphi \in \Phi_{Reach}$ a reachability formula, and St a reduction such that for all $M \in \mathcal{M}(N)$:*

1 $A_M(\varphi) \subseteq St(M)$.
2 For all $t \in St(M)$, if $t \notin en(M)$ then
 - *exists p that disables t in M and $\bullet p \subseteq St(M)$, or*
 - *exists p that inhibits t in M and $p\bullet \subseteq St(M)$.*

Formula φ	$A_M(\varphi)$	$A_M(\neg\varphi)$
$true$	$\emptyset$	$\emptyset$
$false$	$\emptyset$	$\emptyset$
t	$\bullet p$ for some $p \in \bullet t$ where $M(p) < W(p,t)$ or $p\bullet$ for some $p \in \circ t$ where $M(p) \geq I(p,t)$	$(\bullet t) \bullet \cup \bullet (\circ t)$
$deadlock$	$(\bullet t) \bullet \cup \bullet (\circ t)$ for some $t \in en(M)$	$\emptyset$
$e_1 \bowtie e_2$	See Table 3	$A_M(e_1 \overline{\bowtie} e_2)$
$\varphi_1 \wedge \varphi_2$	$A_M(\varphi_i)$ for some $i \in \{1,2\}$ where $M \not\models \varphi_i$	$A_M(\neg\varphi_1 \vee \neg\varphi_2)$
$\varphi_1 \vee \varphi_2$	$A_M(\varphi_1) \cup A_M(\varphi_2)$	$A_M(\neg\varphi_1 \wedge \neg\varphi_2)$
$\varphi_1 \implies \varphi_2$	$A_M(\neg\varphi_1 \vee \varphi_2)$	$A_M(\varphi_1 \wedge \neg\varphi_2)$
$\varphi_1 \iff \varphi_2$	$A_M(\varphi_1 \implies \varphi_2 \wedge \varphi_2 \implies \varphi_1)$	$A_M(\varphi_1 \iff \neg\varphi_2)$

Table 1: Interesting transitions of φ.

Operator $\bowtie$	$\overline{\bowtie}$
$<$	$\geq$
$\leq$	$>$
$=$	$\neq$
$\neq$	$=$
$>$	$\leq$
$\geq$	$<$

Table 2: Complement of comparison operator $\bowtie$.

Formula $e_1 \bowtie e_2$	$A_M(e_1 \bowtie e_2)$
$e_1 < e_2$	$decr_M(e_1) \cup incr_M(e_2)$
$e_1 \leq e_2$	$decr_M(e_1) \cup incr_M(e_2)$
$e_1 > e_2$	$incr_M(e_1) \cup decr_M(e_2)$
$e_1 \geq e_2$	$incr_M(e_1) \cup decr_M(e_2)$
$e_1 = e_2$	if $eval_M(e_1) > eval_M(e_2)$ then $decr_M(e_1) \cup incr_M(e_2)$ else if $eval_M(e_1) < eval_M(e_2)$ then $incr_M(e_1) \cup decr_M(e_2)$
$e_1 \neq e_2$	$incr_M(e_1) \cup decr_M(e_1) \cup incr_M(e_2) \cup decr_M(e_2)$

Table 3: Interesting transitions of $e_1 \bowtie e_2$.

Expression e	$incr_M(e)$	$decr_M(e)$
c	$\emptyset$	$\emptyset$
p	$\bullet p$	$p\bullet$
$e_1 + e_2$	$incr_M(e_1) \cup incr_M(e_2)$	$decr_M(e_1) \cup decr_M(e_2)$
$e_1 - e_2$	$incr_M(e_1) \cup decr_M(e_2)$	$decr_M(e_1) \cup incr_M(e_2)$
$e_1 * e_2$	$incr_M(e_1) \cup decr_M(e_1) \cup$ $incr_M(e_2) \cup decr_M(e_2)$	$incr_M(e_1) \cup decr_M(e_1) \cup$ $incr_M(e_2) \cup decr_M(e_2)$

Table 4: Increasing and decreasing transitions of e.

Algorithm 1: Construction of a reachability preserving stubborn set

```
input  : N = (P, T, W, I), M ∈ 𝓜(N), φ
output : St(M) where St satisfies W and R
X := ∅; unprocessed := A_M(φ);
while unprocessed ≠ ∅ do
    pick any t ∈ unprocessed;
    if t ∉ en(M) then
        if Exists p ∈ •t s.t. M(p) < W(p, t) then
            pick any p ∈ •t s.t. M(p) < W(p, t);
            unprocessed := unprocessed ∪ (•p \ X);
        else
            pick any p ∈ ∘t s.t. M(p) ≥ I(p, t);
            unprocessed := unprocessed ∪ (p • \X);
    else
        unprocessed := unprocessed ∪ ((•t) • \X) ∪ ((t•) ∘ \X);
    unprocessed := unprocessed \ {t};
    X := X ∪ {t};
return X;
```

3 For all $t \in St(M)$, if $t \in en(M)$ then
- $(\bullet t)\bullet \subseteq St(M)$, *and*
- $(t\bullet)\circ \subseteq St(M)$.

*then St satisfies **W** and **R**.*

In Algorithm 1 we illustrate pseudocode on how to construct a reachability preserving stubborn set that satisfies **W** and **R** for a given marking M and reachability formula $EF\varphi \in \Phi_{Reach}$.

Lemma 4. *Algorithm 1 terminates.*

Lemma 5. *When Algorithm 1 terminates, the reduction St computed by the algorithm satisfies **W** and **R**.*

7 The Siphon-Trap Property

It is possible to check for deadlock freedom in Petri nets by examining structural entities within a Petri net called *siphons* and *traps*. For this we only consider 1-weighted Petri nets without inhibitor arcs. A Petri net $N = (P, T, W, I)$ is 1-weighted if $W : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$, i.e. every regular arc have a weight of 0 or 1. N have no inhibitor arcs if for all $p \in P$ and $t \in T$ we have $I(p, t) = \infty$, i.e. the inhibitor arcs have no effect on the enabledness of the transitions.

Definition 4 (Siphon). *Let $N = (P, T, W, I)$ be a 1-weighted Petri net with no inhibitor arcs and M_0 an initial marking on N. A siphon D of N, is a non-empty set of places $D \subseteq P$, where $\bullet D \subseteq D\bullet$. We say that D is marked if there exists a place $p \in D$ with $M_0(p) > 0$.*

Definition 5 (Trap). *Let $N = (P, T, W, I)$ be a 1-weighted Petri net with no inhibitor arcs and M_0 an initial marking on N. A trap Q of N, is a non-empty set of places $Q \subseteq P$, where $Q\bullet \subseteq \bullet Q$. We say that Q is marked if there exists a place $p \in Q$ with $M_0(p) > 0$.*

Definition 6 (Siphon-Trap Property). *Let $N = (P, T, W, I)$ be a 1-weighted Petri net with no inhibitor arcs and M_0 an initial marking on N. We say that N has the siphon-trap property if for every siphon $D \subseteq P$ there exists a trap $Q \subseteq D$ s.t. Q is marked.*

Proposition 2 (Commoner-Hack). *Let N be a 1-weighted Petri net with no inhibitor arcs and M_0 an initial marking on N. If N has the siphon-trap property then no deadlock is reachable from M_0.*

7.1 Siphon-Trap Property Using Integer Linear Programming

Let $N = (P, T, W, I)$ be a 1-weighted Petri net with no inhibitor arcs and M_0 an initial marking on N. We know that N has the siphon-trap property if for every siphon $D \subseteq P$ there exists a trap $Q \subseteq D$ s.t. Q is marked. Let D be a siphon of N. The unique maximal trap of D is the union of all traps within D, written Q_{max} where traps are closed under union. We can convert the problem into an appropriate form:

$$\begin{aligned}
\text{for all siphons } D \subseteq P \ \text{ exists a trap } Q \subseteq D \text{ s.t. } Q \text{ is marked } &\iff \\
\neg(\text{exists a siphon } D \subseteq P \ \text{ s.t. for all traps } Q \subseteq D \ \textit{s.t.}\ Q \text{ is not marked}) &\iff \\
\neg(\text{exists a siphon } D \text{ s.t. the maximal trap } Q_{max} \text{ of } D \text{ is not marked})&
\end{aligned}$$

We want to prove that there exists a siphon whose maximal trap is not marked in order to disprove the siphon-trap property. If we cannot prove this, then the Petri net must have the siphon-trap property.

Let $N = (P, T, W, I)$ be a 1-weighted Petri net with no inhibitor arcs, M_0 an initial marking on N, and $d \in \mathbb{N}^0$ a natural number indicating the depth of the procedure. We have a sequence of sets $X_0, X_1, \ldots, X_d$ such that:

$$P \supseteq X_0 \supseteq X_1 \supseteq \ldots \supseteq X_d$$

The set X_0 represents the initially selected siphon and each subsequent set represents a candidate maximal trap for the siphon, moving towards either the maximal trap or the empty set. For each place p we have $d+1$ decision variables such that for all $0 \leq i \leq d$ we have $p^i \in \{0, 1\}$, and $p^i = 1$ if and only if $p \in X_i$.

Additionally, we introduce $d+1$ decision variables for each transition t, written as $post_t^d$, such that for all $0 \leq i \leq d$ we have $post_t^i \in \{0, 1\}$, and $post_t^i = 1$ if and only if there exists a place $p \in t\bullet$ such that $p^i = 1$. Equation 1 ensures if $post_t^i = 1$ then there exists a place $p \in t\bullet$ such that $p^i = 1$, and Equation 2 ensures if there exists a place $p \in t\bullet$ such that $p^i = 1$ then $post_t^i = 1$.

$$-post_t^i + \sum_{p \in t\bullet} p^i \geq 0 \qquad \forall i \in \{0, \ldots, d\}, \forall t \in T \tag{1}$$

$$p^i - post_t^i \leq 0 \qquad \forall i \in \{0, \ldots, d\}, \forall t \in T, \forall p \in t\bullet \tag{2}$$

We need to specify integer linear equations such that there exists a solution if the following conditions are true:

a $\bullet X_0 \subseteq X_0\bullet$, X_0 is a siphon of N.
b $X_0 \neq \emptyset$, the initial siphon is not empty.
c For all $0 \leq i \leq d$ we have $X_{i+1} \subseteq X_i$, we never add places as we iterate.
d For all $t \in T$ we have $p \in \bullet t$ and $p \in X_{i+1}$ if and only if there exists $p' \in t\bullet$ s.t. $p' \in X_i$.
e For all $p \in X_d$ we have $M_0(p) = 0$, or X_d is not a trap.

The reason we need the second part of condition e is because after d iterations we are not guaranteed to converge on the maximal trap. In order to guarantee convergence, we need the depth to be equal to the number of places, i.e. $d = |P|$.

Equation 3 ensures condition a.

$$-p^0 + \sum_{q \in \bullet t} q^0 \geq 0 \qquad \forall t \in T, \forall p \in t\bullet \tag{3}$$

If p is in the initial siphon, i.e. $p^0 = 1$, and it is given a token when t is fired, then we must have at least one place $q^0 = 1$ in the siphon where a token is removed when t is fired, otherwise the equation is not satisfied.

Equation 4 ensures condition b.

$$\sum_{p \in P} p^0 \geq 1 \tag{4}$$

At least one place must be assigned a value of 1 to ensure the initial siphon X_0 is non-empty, otherwise the equation is not satisfied.

Equation 5 ensures condition c.

$$-p^{i+1} + p^i \geq 0 \qquad \forall i \in \{0, \dots, d\}, \forall p \in P \tag{5}$$

If $p^{i+1} = 1$ then we must also have that $p^i = 1$, otherwise the equation is not satisfied. No places can be added in later iterations.

Equation 6 ensures the left-to-right implication of condition d.

$$-p^{i+1} + post_t^i \geq 0 \qquad \forall i \in \{0, \dots, d\}, \forall p \in P, \forall t \in p\bullet \tag{6}$$

Equation 7 ensures the right-to-left implication of condition d.

$$-p^{i+1} + p^i + \sum_{t \in p\bullet} post_t^i \leq |p\bullet| \qquad \forall i \in \{0, \dots, d\}, \forall p \in P \tag{7}$$

We iteratively remove places from the identified siphon until we are either left with the empty set or the maximal trap, iterating d times. A place $p \in X_i$ is removed from the siphon in the ith step by assigning its decision variable p^{i+1} to 0 in step $i+1$, where $p^i = 1$. If place p is not part of the siphon in step i, i.e. $p \notin X_i$ and $p^i = 0$, then it stays outside of the siphon in step $i+1$ and $p^{i+1} = 0$, as we do not add any places. A place p is removed in the ith step if and only if there exists a transition $t \in p\bullet$ s.t. $t\bullet \nsubseteq X_i$.

Once the removal procedure reaches depth d, we are left with one of three cases: Either X_d is the maximal trap, not a trap at all, or the empty set. In either case, we need to check if the set is unmarked. If it is unmarked then X_0 is a siphon with no marked trap, and therefore disproves the siphon-trap property. Let $z \in \mathbb{N}^0$ be a decision variable. Equation 8 ensures the first part of condition e. Equation 9 ensures the second part of condition e.

$$p^{d+1} - z \leq 0 \qquad \forall p \in P \text{ where } M_0(p) > 0 \tag{8}$$

$$\sum_{p \in P} p^{d+1} + z = \sum_{p \in P} p^d \tag{9}$$

By the construction and reasoning from the integer linear program specification above, we conclude with the following theorem.

Theorem 2. *If the integer linear program specified in equations 1 through 9 is infeasible then N has no deadlock.*

8 Formula Simplification

To perform formula simplification, we need a way to identify contradictions and impossibilities in the formula.

8.1 Simplification Procedure

We define a function, that given a formula, produces a simplified formula and a set of integer linear programs. We say that such a function is a *simplification* function.

Definition 7 (Simplification). *Let $N = (P, T, W, I)$ be a Petri net, M_0 an initial marking on N, and $X = \{x_t \mid t \in T\}$ a set of variables. A simplification for marking M_0 is a function simplify : $\Phi_{CTL} \to \Phi_{CTL} \times 2^{\mathcal{E}^X_{lin}}$.*

φ	Rewritten φ
t	$p_1 \geq W(p_1,t) \wedge \cdots \wedge p_n \geq W(p_n,t) \wedge$ $p_1 < I(p_1,t) \wedge \cdots \wedge p_n < I(p_n,t)$ where $n = \lvert P \rvert$
$e_1 \neq e_2$	$e_1 > e_2 \vee e_1 < e_2$
$e_1 = e_2$	$e_1 \leq e_2 \wedge e_1 \geq e_2$
$\neg(\varphi_1 \wedge \varphi_2)$	$\neg\varphi_1 \vee \neg\varphi_2$
$\neg(\varphi_1 \vee \varphi_2)$	$\neg\varphi_1 \wedge \neg\varphi_2$
$\varphi_1 \implies \varphi_2$	$\neg\varphi_1 \vee \varphi_2$
$\varphi_1 \iff \varphi_2$	$(\varphi_1 \wedge \varphi_2) \vee (\neg\varphi_1 \wedge \neg\varphi_2)$
$\neg AX\varphi$	$EX\neg\varphi$
$\neg EX\varphi$	$AX\neg\varphi$
$\neg AF\varphi$	$EG\neg\varphi$
$\neg EF\varphi$	$AG\neg\varphi$
$\neg AG\varphi$	$EF\neg\varphi$
$\neg EG\varphi$	$AF\neg\varphi$

Table 5: Rewriting rules for φ.

φ	$\mathit{simplify}(M_0, \varphi)$
true	$(\mathit{true}, \{\{0 \leq 1\}\})$
false	$(\mathit{false}, \emptyset)$
deadlock	$(\mathit{deadlock}, \{\{0 \leq 1\}\})$

Table 6: Trivial cases of *simplify*.

The function *merge* : $2^{\mathcal{E}^X_{lin}} \times 2^{\mathcal{E}^X_{lin}} \to 2^{\mathcal{E}^X_{lin}}$ combines two LPS and is defined as *merge* $(LPS_1, LPS_2)= \{LP_1 \cup LP_2 \mid LP_1 \in LPS_1$ and $LP_2 \in LPS_2\}$.

Algorithm 2: Simplify $\varphi_1 \wedge \varphi_2$

Function $\mathit{simplify}(\varphi_1 \wedge \varphi_2)$
  $(\varphi'_1, LPS_1) \leftarrow \mathit{simplify}(\varphi_1)$
  if $\varphi'_1 = \mathit{false}$ **then**
    return $(\mathit{false}, \emptyset)$
  $(\varphi'_2, LPS_2) \leftarrow \mathit{simplify}(\varphi_2)$
  if $\varphi'_2 = \mathit{false}$ **then**
    return $(\mathit{false}, \emptyset)$
  else if $\varphi'_2 = \mathit{true}$ **then**
    return (φ'_1, LPS_1)
  else if $\varphi'_1 = \mathit{true}$ **then**
    return (φ'_2, LPS_2)
  $LPS \leftarrow \mathit{merge}(LPS_1, LPS_2)$
  if $\{LP \cup BASE \mid LP \in LPS\}$ *has no solution* **then**
    return $(\mathit{false}, \emptyset)$
  else
    return $(\varphi'_1 \wedge \varphi'_2, LPS)$

$BASE$ is an integer linear program of a Petri net $N = (P, T, W, I)$ and initial marking M_0 on N, that consists of the following set of linear equations:

$$M_0(p) + \sum_{t \in T}(W(t,p) - W(p,t))x_t \geq 0 \quad \text{for all } p \in P.$$

Which ensures that no solution to the linear program, can leave a place with a negative amount of tokens.

Algorithm 3: Simplify $\varphi_1 \vee \varphi_2$

Function $\mathit{simplify}(\varphi_1 \vee \varphi_2)$
  $(\varphi'_1, LPS_1) \leftarrow \mathit{simplify}(\varphi_1)$
  if $\varphi'_1 = \mathit{true}$ **then**
    return $(\mathit{true}, \{\{0 \leq 1\}\})$
  $(\varphi'_2, LPS_2) \leftarrow \mathit{simplify}(\varphi_2)$
  if $\varphi'_2 = \mathit{true}$ **then**
    return $(\mathit{true}, \{\{0 \leq 1\}\})$
  if $\varphi'_1 = \mathit{false}$ **then**
    return (φ'_2, LPS_2)
  if $\varphi'_2 = \mathit{false}$ **then**
    return (φ'_1, LPS_1)
  $(\varphi''_1, LPS'_1) \leftarrow \mathit{simplify}(\neg\varphi_1)$
  $(\varphi''_2, LPS'_2) \leftarrow \mathit{simplify}(\neg\varphi_2)$
  $LPS \leftarrow \mathit{merge}(LPS'_1, LPS'_2)$
  if $\{LP \cup BASE \mid LP \in LPS\}$ *has no solution* **then**
    return $(\mathit{true}, \{\{0 \leq 1\}\})$
  return $(\varphi'_1 \vee \varphi'_2, LPS_1 \cup LPS_2)$

Algorithm 4: Simplify $\neg\varphi$

Function $\mathit{simplify}(\neg\varphi)$
  $(\varphi', LPS) \leftarrow \mathit{simplify}(\varphi)$
  if $\varphi' = \mathit{true}$ **then**
    return $(\mathit{false}, \emptyset)$
  else if $\varphi'_2 = \mathit{false}$ **then**
    return $(\mathit{true}, \{\{0 \leq 1\}\})$
  else
    return $(\neg\varphi', \{\{0 \leq 1\}\})$

For the comparison operator $e_1 \bowtie e_2$ we introduce the function *const* which takes as input an expression e and returns one side of a linear equation.

$$\begin{aligned}
\mathit{const}(c) &= c \\
\mathit{const}(p) &= M_0(p) + \sum_{t \in T} (W(t,p) - W(p,t)) x_t \\
\mathit{const}(e_1 + e_2) &= \mathit{const}(e_1) + \mathit{const}(e_2) \\
\mathit{const}(e_1 - e_2) &= \mathit{const}(e_1) - \mathit{const}(e_2) \\
\mathit{const}(e_1 \cdot e_2) &= \mathit{const}(e_1) \cdot \mathit{const}(e_2)
\end{aligned}$$

Algorithm 5: Simplify $e_1 \bowtie e_2$

Function *simplify*$(e_1 \bowtie e_2)$
    if e_1 *is not linear or* e_2 *is not linear* **then**
        return $(e_1 \bowtie e_2, \{\{0 \leq 1\}\})$
    $LPS_1 \leftarrow \{\{const(e_1) \bowtie const(e_2)\}\}$
    $LPS_2 \leftarrow \{\{const(e_1) \overline{\bowtie} const(e_2)\}\}$
    if $\{LP \cup BASE \mid LP \in LPS_1\}$ *have no solution* **then**
        return $(\mathit{false}, \emptyset)$
    else if $\{LP \cup BASE \mid LP \in LPS_2\}$ *have no solution* **then**
        return $(\mathit{true}, \{\{0 \leq 1\}\})$
    else
        return $(e_1 \bowtie e_2, LPS_1)$

Algorithm 6: Simplify $AX\varphi$

Function *simplify*$(AX\varphi)$
    $(\varphi', LPS) \leftarrow \mathit{simplify}(\varphi)$
    if $\varphi' = \mathit{true}$ **then**
        return $(\mathit{true}, \{\{0 \leq 1\}\})$
    else if $\varphi' = \mathit{false}$ **then**
        return $(\mathit{deadlock}, \{\{0 \leq 1\}\})$
    else
        return $(AX\varphi', \{\{0 \leq 1\}\})$

Lemma 6 (Formula Simplification Correctness). *Let $N = (P, T, W, I)$ be a Petri net, M_0 an initial marking on N, and $\varphi \in \Phi_{CTL}$ a CTL formula. If simplify$(\varphi) = (\varphi', LPS)$ then for all $M \in \mathcal{M}(N)$ such that $M_0 \xrightarrow{w} M$ we have:*

1. *$M \models \varphi$ iff $M \models \varphi'$, and*
2. *if $M \models \varphi$ then there exists $LP \in LPS$ such that $\wp(w)$ is a solution to LP.*

Algorithm 7: Simplify $EX\varphi$

```
Function simplify(EXφ)
    (φ', LPS) ← simplify(φ)
    if φ' = true then
        return (¬deadlock, {{0 ≤ 1}})
    else if φ' = false then
        return (false, ∅)
    else
        return (EXφ', {{0 ≤ 1}})
```

Algorithm 8: Simplify $QF\varphi$ where $Q \in \{A, E\}$

```
Function simplify(QFφ)
    (φ', LPS) ← simplify(φ)
    if φ' = true then
        return (true, {{0 ≤ 1}})
    else if φ' = false then
        return (false, ∅)
    else
        return (QFφ', {{0 ≤ 1}})
```

Algorithm 9: Simplify $QG\varphi$ where $Q \in \{A, E\}$

```
Function simplify(QGφ)
    (φ', LPS) ← simplify(φ)
    if φ' = true then
        return (true, {{0 ≤ 1}})
    else if φ' = false then
        return (false, ∅)
    else
        return (QGφ', {{0 ≤ 1}})
```

Algorithm 10: Simplify $Q(\varphi_1 U \varphi_2)$ where $Q \in \{A, E\}$

Function *simplify*$(Q(\varphi_1 U \varphi_2))$
    $(\varphi'_2, LPS_2) \leftarrow$ *simplify*(φ_2)
    if $\varphi'_2 = true$ **then**
        return $(true, \{\{0 \leq 1\}\})$
    else if $\varphi'_2 = false$ **then**
        return $(false, \emptyset)$
    $(\varphi'_1, LPS_1) \leftarrow$ *simplify*(φ_1)
    if $\varphi'_1 = true$ **then**
        return $(QF\varphi'_2, \{\{0 \leq 1\}\})$
    else if $\varphi'_1 = false$ **then**
        return (φ'_2, LPS_2)
    else
        return $(Q(\varphi'_1 U \varphi'_2), \{\{0 \leq 1\}\})$