Model Checking Branching Properties on Petri Nets with Transits [*]

Bernd Finkbeiner[1], Manuel Gieseking[2],
Jesko Hecking-Harbusch[1], and Ernst-Rüdiger Olderog[2]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{finkbeiner,jesko.hecking-harbusch}@cispa.saarland
[2] University of Oldenburg, Oldenburg, Germany
{gieseking,olderog}@informatik.uni-oldenburg.de

Abstract. To model check concurrent systems, it is convenient to distinguish between the data flow and the control. Correctness is specified on the level of data flow whereas the system is configured on the level of control. Petri nets with transits and Flow-LTL are a corresponding formalism. In Flow-LTL, both the correctness of the data flow and assumptions on fairness and maximality for the control are expressed in linear time. So far, branching behavior cannot be specified for Petri nets with transits. In this paper, we introduce Flow-CTL* to express the intended branching behavior of the data flow while maintaining LTL for fairness and maximality assumptions on the control. We encode physical access control with policy updates as Petri nets with transits and give standard requirements in Flow-CTL*. For model checking, we reduce the model checking problem of Petri nets with transits against Flow-CTL* via automata constructions to the model checking problem of Petri nets against LTL. Thereby, physical access control with policy updates under fairness assumptions for an unbounded number of people can be verified.

1 Introduction

Petri nets with transits [8] superimpose a transit relation onto the flow relation of Petri nets. The flow relation models the *control* in the form of tokens moving through the net. The transit relation models the *data flow* in the form of flow chains. The configuration of the system takes place on the level of the control whereas correctness is specified on the level of the data flow. Thus, Petri nets with transits allow for an elegant separation of the data flow and the control without the complexity of unbounded colored Petri nets [14]. We use *physical access control* [12,11,13] as an application throughout the paper. It defines and enforces access policies in physical spaces. People are represented as the data flow in the building. The control defines which policy enforcement points like

[*]This work has been supported by the German Research Foundation (DFG) through Grant Petri Games (392735815) and through the Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR 248, 389792660), and by the European Research Council (ERC) through Grant OSARES (683300).

doors are open to which people identified by their RFID cards [19]. Changing access policies is error-prone as closing one door for certain people could be circumvented by an alternative path. Therefore, we need to verify such updates.

Flow-LTL [8] is a logic for Petri nets with transits. It specifies linear time requirements on both the control and the data flow. Fairness and maximality assumptions on the movement of tokens are expressed in the control part. The logic lacks branching requirements for the data flow. In physical access control, branching requirements can specify that a person has the *possibility* to reach a room but not necessarily has to visit it. In this paper, we introduce *Flow-CTL** which maintains LTL to specify the control and adds CTL* to specify the data flow. Fairness and maximality assumptions in the control part dictate which executions, represented by runs, are checked against the data flow part.

This leads to an interesting encoding for physical access control in Petri nets with transits. Places represent rooms to collect the data flow. Transitions represent doors between rooms to continue the data flow. The selection of runs by fairness and maximality assumptions on the control restricts the branching behavior to transitions. Hence, the data flow is split at transitions: Every room has exactly one outgoing transition enabled unless all outgoing doors are closed. This transition splits the data flow into all successor rooms and thereby represents the maximal branching behavior.

We present a reduction of the model checking problem of safe Petri nets with transits against Flow-CTL* to the model checking problem of safe Petri nets against LTL. This enables for the first time the automatic verification of physical access control with *policy updates* under fairness and maximality assumptions for an unbounded number of people. Policy updates occur for example in the evening when every employee is expected to eventually leave the building and therefore access is more restricted. Such a policy update should prevent people from entering the building but should not trap anybody in the building.

Our reduction consists of three steps: First, each data flow subformula of the given Flow-CTL* formula is represented, via an alternating tree automaton, an alternating word automaton, and a nondeterministic Büchi automaton, by a finite Petri net to guess and then to verify a counterexample tree. Second, the original net for the control subformula of the Flow-CTL* formula and the nets for the data flow subformulas are connected in sequence. Third, an LTL formula encodes the control subformula, the acceptance conditions of the nets for the data flow subformulas, and the correct skipping of subnets in the sequential order. This results in a model checking problem of safe Petri nets against LTL.

The remainder of this paper is structured as follows: In Sect. 2, we motivate our approach with an example. In Sect. 3, we recall Petri nets and their extension to Petri nets with transits. In Sect. 4, we introduce Flow-CTL*. In Sect. 5, we express fairness, maximality, and standard properties for physical access control in Flow-CTL*. In Sect. 6, we reduce the model checking problem of Petri nets with transits against Flow-CTL* to the model checking problem of Petri nets against LTL. Section 7 presents related work and Sect. 8 concludes the paper. Further details can be found in the full version of the paper [10].

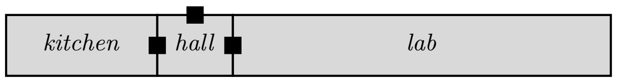


Fig. 1: The layout of a simple building is shown. There are three rooms indicated by gray boxes which are connected by doors indicated by small black boxes.

2 Motivating Example

We motivate our approach with a typical example for physical access control. Consider the very simple building layout in Fig. 1. There are three rooms connected by two doors. An additional door is used to enter the building from the outside. Only employees have access to the building. A typical specification requires that employees can access the *lab* around the clock while allowing access to the *kitchen* only during daytime to discourage too long working hours. Meanwhile, certain safety requirements have to be fulfilled like not trapping anybody in the building. During the day, a correct access policy allows access to all rooms whereas, during the night, it only allows access to the *hall* and to the *lab*.

Figure 2 shows a Petri nets with transits modeling the building layout from Fig. 1. There are corresponding places (represented by circles) with tokens (represented by dots) for the three rooms: *hall*, *lab*, and *kitchen*. These places are connected by transitions (represented by squares) of the form *from*→*to* for *from* and *to* being rooms. The doors from the *kitchen* and *lab* to the *hall* cannot be closed as this could trap people. For all other doors, places of the form $o_{from \to to}$ and $c_{from \to to}$ exist to represent whether the door is open or closed.

In (safe) Petri nets, transitions define the movement of tokens: Firing a transition removes one token from each place with a black arrow leaving to the transition and adds one token to each place with a black arrow coming from the transition. Firing transition *evening* moves one token from place $o_{h \to k}$ to place $c_{h \to k}$ as indicated by the single-headed, black arrows and one token from and to each of the places *hall*, *lab*, and *kitchen* as indicated by the double-headed, black arrows. Firing transitions modeling doors returns all tokens to the same places while the transit relation as indicated by the green, blue, and orange arrows represents employees moving through the building. Dashed and dotted arrows only distinguish them from black arrows in case colors are unavailable.

Firing transition *enterHall* starts a flow chain modeling an employee entering the building as indicated by the single-headed, green (dashed) arrow. Meanwhile, the double-headed, blue (dotted) arrow maintains all flow chains previously in *hall*. All flow chains collectively represent the data flow in the modeled system incorporating all possible control changes. Firing transitions *from*→*to*, which correspond to doors, continues all flow chains from place *from* to place *to* as indicated by the single-headed, green (dashed) arrows and merges them with all flow chains in the place *to* as indicated by the double-headed, blue (dotted) arrows. For example, firing transition *hall*→*lab* lets all employees in the *hall* enter the *lab*. When employees *leave* the *hall*, their flow chain ends because it is not continued as indicated by the lack of colored arrows at transition *leaveHall*.

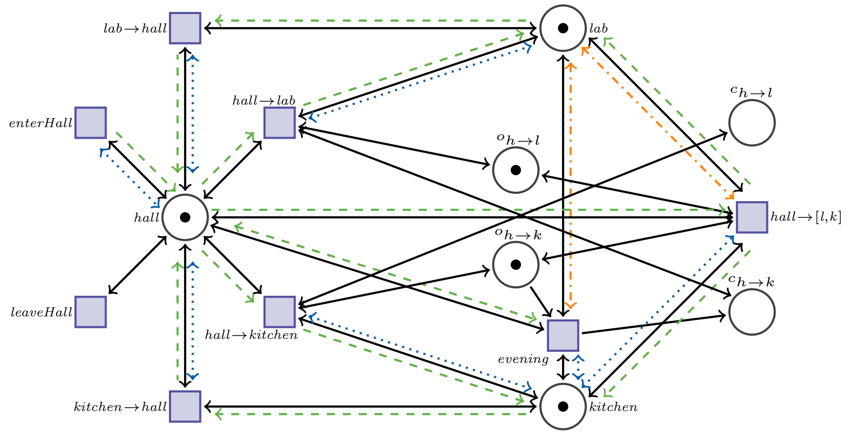


Fig. 2: The Petri net with transits encoding the building from Fig. 1 is depicted. Rooms are modeled by corresponding places, doors by transitions. Tokens in places starting with *o* configure the most permissive access policy during the day. In the *evening*, access to the *kitchen* is restricted. Employees in the building are modeled by the transit relation depicted by green, blue, and orange arrows.

Flow-CTL* allows the splitting of flow chains in transitions. Splitting flow chains corresponds to branching behavior. Thus, when the doors to the *lab* and *kitchen* are open, we represent this situation by *one* transition which splits the flow chains. Transition *hall*→[*l*,*k*] realizes this by the single-headed, green (dashed) arrows from the *hall* to the *lab* and *kitchen*. Branching results in a *flow tree* for the possible behavior of an employee whereas a flow chain represents one explicit path from this flow tree, i.e., each employee has one flow tree with possibly many flow chains. Notice that transition *hall*→[*l*,*k*] can only be fired during the day, because, when firing transition *evening*, access to the *kitchen* is revoked. Then, only transition *hall*→*lab* can be fired for moving flow chains from the *hall*. For simplicity, we restrict the example to only one time change which implies that the transition *hall*→*kitchen* can never be fired. Firing transition *evening* continues all flow chains in the three places *hall*, *lab*, and *kitchen*, respectively, as indicated by the distinctly colored, double-headed arrows. Thus, we can specify requirements for the flow chains after the time change.

We specify the correctness of access policies with formulas of the logic *Flow-CTL**. The formula $\mathbb{A}\,\mathbf{AGEF}\mathit{lab}$ expresses *persistent permission* requiring that all flow chains ($\mathbb{A}$) on all paths globally ($\mathbf{AG}$) have the possibility ($\mathbf{EF}$) to reach the *lab*. The formula $\mathbb{A}\,\mathbf{A}((\mathbf{EF}\mathit{kitchen})\mathbf{U}\mathit{evening})$ expresses *dependent permission* requiring that all flow chains on all paths ($\mathbf{A}$) have the possibility to reach the *kitchen* until ($\mathbf{U}$) *evening*. Both properties require weak or strong fairness for all transitions modeling doors to be satisfied. The second property additionally requires weak or strong fairness for transition *evening* to be satisfied. Flow-CTL* and specifying properties with it are discussed further in Sect. 4 and Sect. 5.

3 Petri Nets with Transits

We recall the formal definition of Petri nets with transits [8] as extension of Petri nets [17]. We refer the reader to the full paper for more details [10]. A safe *Petri net* is a structure $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ with the set of *places* $\mathcal{P}$, the set of *transitions* $\mathcal{T}$, the (*control*) *flow relation* $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$, and the *initial marking* $In \subseteq \mathcal{P}$. In *safe* Petri nets, each reachable marking contains at most one token per place. The elements of the disjoint union $\mathcal{P} \cup \mathcal{T}$ are considered as *nodes*. We define the *preset* (and *postset*) of a node x from Petri net $\mathcal{N}$ as $pre^{\mathcal{N}}(x) = \{y \in \mathcal{P} \cup \mathcal{T} \mid (y, x) \in \mathcal{F}\}$ (and $post^{\mathcal{N}}(x) = \{y \in \mathcal{P} \cup \mathcal{T} \mid (x, y) \in \mathcal{F}\}$). A safe *Petri net with transits* is a structure $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In, \Upsilon)$ which additionally contains a *transit relation* Υ refining the flow relation of the net to define the data flow. For each transition $t \in \mathcal{T}$, $\Upsilon(t)$ is a relation of type $\Upsilon(t) \subseteq (pre^{\mathcal{N}}(t) \cup \{\rhd\}) \times post^{\mathcal{N}}(t)$, where the symbol $\rhd$ denotes a *start*. With $\rhd\ \Upsilon(t)\ q$, we define the start of a new data flow in place q via transition t and with $p\ \Upsilon(t)\ q$ that all data in place p *transits* via transition t to place q. The *postset regarding* Υ of a place $p \in \mathcal{P}$ and a transition $t \in post^{\mathcal{N}}(p)$ is defined by $post^{\Upsilon}(p, t) = \{p' \in \mathcal{P} \mid (p, p') \in \Upsilon(t)\}$.

The graphic representation of $\Upsilon(t)$ in Petri nets with transits uses a *color coding* as can be seen in Fig. 2. Black arrows represent the usual *control flow*. Other matching colors per transition are used to represent the transits of the *data flow*. Transits allow us to specify where the data flow is moved forward, split, and merged, where it ends, and where data is newly created. The data flow can be of infinite length and at any point in time (possibly restricted by the control) new data can enter the system at different locations.

As the data flow is a local property of each distributed component (possibly shared via joint transitions) it is convenient that Petri nets with transits use a true concurrency semantics to define the data flow. Therefore, we recall the notions of unfoldings and runs [5,6] and their application to Petri nets with transits. In the unfolding of a Petri net $\mathcal{N}$, every transition stands for the unique occurrence (instance) of a transition of $\mathcal{N}$ during an execution. To this end, every loop in $\mathcal{N}$ is unrolled and every backward branching place is expanded by multiplying the place. Forward branching, however, is preserved. Formally, an *unfolding* is a branching process $\beta^U = (\mathcal{N}^U, \lambda^U)$ consisting of an occurrence net $\mathcal{N}^U$ and a homomorphism λ^U that labels the places and transitions in $\mathcal{N}^U$ with the corresponding elements of $\mathcal{N}$. The unfolding exhibits concurrency, causality, and nondeterminism (forward branching) of the unique occurrences of the transitions in $\mathcal{N}$ during all possible executions. A *run* of $\mathcal{N}$ is a subprocess $\beta = (\mathcal{N}^R, \rho)$ of β^U, where $\forall p \in \mathcal{P}^R : |post^{\mathcal{N}^R}(p)| \leq 1$ holds, i.e., all nondeterminism has been resolved but concurrency is preserved. Thus, a run formalizes one concurrent execution of $\mathcal{N}$. We lift the transit relation of a Petri net with transits to any branching process and thereby obtain notions of runs and unfoldings for Petri nets with transits. Consider a run $\beta = (\mathcal{N}^R, \rho)$ of $\mathcal{N}$ and a finite or infinite firing sequence $\zeta = M_0[t_0\rangle M_1[t_1\rangle M_2 \cdots$ of $\mathcal{N}^R$ with $M_0 = In^R$. This sequence *covers* β if $(\forall p \in \mathcal{P}^R : \exists i \in \mathbb{N} : p \in M_i) \wedge (\forall t \in \mathcal{T}^R : \exists i \in \mathbb{N} : t = t_i)$, i.e., all places and transitions in $\mathcal{N}^R$ appear in ζ. Several firing sequences may cover β.

We define flow chains by following the transits of a given run. A (*data*) *flow chain* of a run $\beta = (\mathcal{N}^R, \rho)$ of a Petri net with transits $\mathcal{N}$ is a *maximal* sequence $\xi = t_0, p_0, t_1, p_1, t_2 \ldots$ of connected places and transitions of $\mathcal{N}^R$ with

(I) $(\rhd, p_0) \in \Upsilon^R(t_0)$,

(con) $(p_{i-1}, p_i) \in \Upsilon^R(t_i)$ for all $i \in \mathbb{N} \setminus \{0\}$ if ξ is infinite and for all $i \in \{1, \ldots n\}$ if $\xi = t_0, p_0, t_1, \ldots, t_n, p_n$ is finite,

(max) if $\xi = t_0, p_0, t_1, \ldots, t_n, p_n$ is finite there is no transition $t \in \mathcal{T}^R$ and place $q \in \mathcal{P}^R$ such that $(p_n, q) \in \Upsilon^R(t)$.

A *flow chain suffix* $\xi' = t_0, p_0, t_1, p_1, t_2 \ldots$ of a run β requires constraints (con), (max), and in addition to (I) allows that the chain has already started, i.e., $\exists p \in \mathcal{P}^R : (p, p_0) \in \Upsilon^R(t_0)$.

A Σ-labeled *tree* over a set of *directions* $\mathcal{D} \subset \mathbb{N}$ is a tuple (T, v), with a labeling function $v : T \to \Sigma$ and a tree $T \subseteq \mathcal{D}^*$ such that if $x \cdot c \in T$ for $x \in \mathcal{D}^*$ and $c \in \mathcal{D}$, then both $x \in T$ and for all $0 \leq c' < c$ also $x \cdot c' \in T$ holds. A (*data*) *flow tree* of a run $\beta = (\mathcal{N}^R, \rho)$ represents all branching behavior in the transitions of the run w.r.t. the transits. Formally, for each $t_0 \in \mathcal{T}^R$ and place $p_0 \in \mathcal{P}^R$ with $(\rhd, p_0) \in \Upsilon^R(t_0)$, there is a $\mathcal{T}^R \times \mathcal{P}^R$-labeled tree $\tau = (T, v)$ over directions $\mathcal{D} \subseteq \{0, \ldots, \mathtt{max}\{|post^{\Upsilon^R}(p, t)| - 1 \mid p \in \mathcal{P}^R \wedge t \in post^{\mathcal{N}^R}(p)\}\}$ with

1. $v(\epsilon) = (t_0, p_0)$ for the root ϵ, and
2. if $n \in T$ with $v(n) = (t, p)$ then for the only transition $t' \in post^{\mathcal{N}^R}(p)$ (if existent) we have for all $0 \leq i < |post^{\Upsilon^R}(p, t')|$ that $n \cdot i \in T$ with $v(n \cdot i) = (t', q)$ for $q = \langle post^{\Upsilon^R}(p, t') \rangle_i$ where $\langle post^{\Upsilon^R}(p, t') \rangle_i$ is the i-th value of the ordered list $\langle post^{\Upsilon^R}(p, t') \rangle$.

Figure 3 shows a finite run of the example from Fig. 2 with two flow trees. The first tree starts with transition $enterHall_0$, i.e., $v(\epsilon) = (enterHall_0, hall_1)$ and is indicated by the gray shaded area. This tree represents an extract of the possibilities of a person entering the hall during the day ending with the control change to the evening policy. The second tree ($v(\epsilon) = (enterHall_1, hall_3), v(0) = (lab{\to}hall, hall_4), v(00) = (kitchen{\to}hall, hall_5), v(000) = (evening, hall_6)$) shows the possibilities of a person in this run who later enters the hall and can, because of the run, only stay there. Note that the trees only end due to the finiteness of the run. For maximal runs, trees can only end when transition *leaveHall* is fired.

4 Flow-CTL* for Petri Nets with Transits

We define the new logic *Flow-CTL** to reason about the Petri net behavior and the data flow individually. Properties on the selection of runs and the general behavior of the net can be stated in LTL, requirements on the data flow in CTL*.

4.1 LTL on Petri Net Unfoldings

We recall LTL with *atomic propositions* $AP = \mathcal{P} \cup \mathcal{T}$ on a Petri net $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ and define the semantics on runs and their firing sequences. We

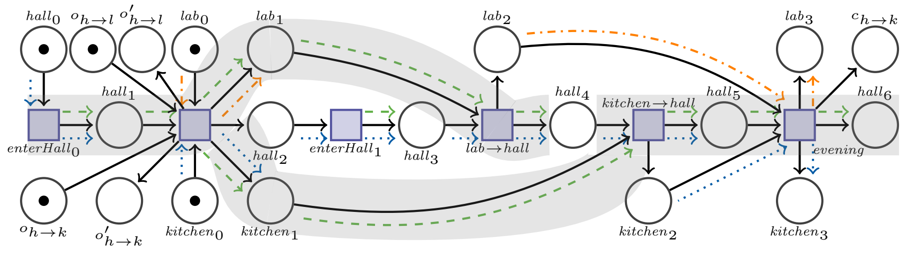


Fig. 3: A finite run of the Petri net with transits from Fig. 2 with two data flow trees is depicted. The first one is indicated by the gray shaded area.

use the *ingoing* semantics, i.e., we consider the marking and the transition used to enter the marking, and *stutter* in the last marking for finite firing sequences.

Syntactically, the set of *linear temporal logic* (LTL) formulas LTL over AP is defined by $\psi ::= true \mid a \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \bigcirc\psi \mid \psi_1 \,\mathcal{U}\, \psi_2$, with $a \in AP$ and $\bigcirc$ being the *next* and $\mathcal{U}$ the *until* operator. As usual, we use the propositional operators $\vee$, $\rightarrow$, and $\leftrightarrow$, the temporal operators $\Diamond\psi = true\,\mathcal{U}\,\psi$ (the *eventually* operator) and $\Box\psi = \neg\Diamond\neg\psi$ (the *always* operator) as abbreviations.

For a Petri net $\mathcal{N}$, we define a *trace* as a mapping $\sigma : \mathbb{N} \to 2^{AP}$. The i-th suffix $\sigma^i : \mathbb{N} \to 2^{AP}$ is a trace defined by $\sigma^i(j) = \sigma(j+i)$ for all $j \in \mathbb{N}$. To a (finite or infinite) covering firing sequence $\zeta = M_0[t_0\rangle M_1[t_1\rangle M_2 \cdots$ of a run $\beta = (\mathcal{N}^R, \rho)$ of $\mathcal{N}$, we associate a trace $\sigma(\zeta) : \mathbb{N} \to 2^{AP}$ with $\sigma(\zeta)(0) = \rho(M_0)$, $\sigma(\zeta)(i) = \{\rho(t_{i-1})\} \cup \rho(M_i)$ for all $i \in \mathbb{N} \setminus \{0\}$ if ζ is infinite and $\sigma(\zeta)(i) = \{\rho(t_{i-1})\} \cup \rho(M_i)$ for all $0 < i \leq n$, and $\sigma(\zeta)(j) = \rho(M_n)$ for all $j > n$ if $\zeta = M_0[t_0\rangle \cdots [t_{n-1}\rangle M_n$ is finite. Hence, a trace of a firing sequence covering a run is an infinite sequence of states collecting the corresponding marking and ingoing transition of $\mathcal{N}$, which stutters on the last marking for finite sequences.

The *semantics* of an LTL formula $\psi \in$ LTL on a Petri net $\mathcal{N}$ is defined over the traces of the covering firing sequences of its runs: $\mathcal{N} \models_{\mathtt{LTL}} \psi$ iff for all runs β of $\mathcal{N}$: $\beta \models_{\mathtt{LTL}} \psi$, $\beta \models_{\mathtt{LTL}} \psi$ iff for all firing sequences ζ covering β : $\sigma(\zeta) \models_{\mathtt{LTL}} \psi$, $\sigma \models_{\mathtt{LTL}} true$, $\sigma \models_{\mathtt{LTL}} a$ iff $a \in \sigma(0)$, $\sigma \models_{\mathtt{LTL}} \neg\psi$ iff not $\sigma \models_{\mathtt{LTL}} \psi$, $\sigma \models_{\mathtt{LTL}} \psi_1 \wedge \psi_2$ iff $\sigma \models_{\mathtt{LTL}} \psi_1$ and $\sigma \models_{\mathtt{LTL}} \psi_2$, $\sigma \models_{\mathtt{LTL}} \bigcirc\psi$ iff $\sigma^1 \models_{\mathtt{LTL}} \psi$, and $\sigma \models_{\mathtt{LTL}} \psi_1 \,\mathcal{U}\, \psi_2$ iff there exists a $j \geq 0$ with $\sigma^j \models_{\mathtt{LTL}} \psi_2$ and $\sigma^i \models_{\mathtt{LTL}} \psi_1$ holds for all $0 \leq i < j$.

4.2 CTL* on Flow Chains

To specify the data flow of a Petri net with transits $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In, \Upsilon)$, we use the complete *computation tree logic (CTL*)*. The set of CTL* formulas CTL* over $AP = \mathcal{P} \cup \mathcal{T}$ is given by the following *syntax* of *state formulas*: $\Phi ::= a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathbf{E}\,\phi$ where $a \in AP$, Φ, Φ_1, Φ_2 are state formulas, and ϕ is a *path formula* with the following *syntax*: $\phi ::= \Phi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathbf{X}\,\phi \mid \phi_1 \mathbf{U}\,\phi_2$ where Φ is a state formula and ϕ, ϕ_1,, ϕ_2 are path formulas. We use the propositional operators $\vee$, $\rightarrow$, $\leftrightarrow$, the path quantifier $\mathbf{A}\phi = \neg\mathbf{E}\neg\phi$, and the temporal operators $\mathbf{F}\phi = true\,\mathbf{U}\,\phi$, $\mathbf{G}\phi = \neg\mathbf{F}\neg\phi$, $\phi_1\mathbf{R}\phi_2 = \neg(\neg\phi_1\mathbf{U}\neg\phi_2)$ as abbreviations.

To a (finite or infinite) flow chain suffix $\xi = t_0, p_0, t_1, p_1, t_2, \dots$ of a run $\beta = (\mathcal{N}^R, \rho)$ of $\mathcal{N}$, we associate a trace $\sigma(\xi) : \mathbb{N} \to \mathcal{S} = \{\{t, p\}, \{p\} \mid p \in \mathcal{P}^R \wedge t \in \mathcal{T}^R\}$ with $\sigma(\xi)(i) = \{t_i, p_i\}$ for all $i \in \mathbb{N}$ if ξ is infinite and $\sigma(\xi)(i) = \{t_i, p_i\}$ for all $i \leq n$, and $\sigma(\xi)(j) = \{p_n\}$ for all $j > n$ if $\xi = t_0, p_0, t_1, p_1, \dots, t_n, p_n$ is finite. Hence, a trace of a flow chain suffix is an infinite sequence of states collecting the current place and ingoing transition of the flow chain, which stutters on the last place p of a finite flow chain suffix. We define $\sigma_{\mathfrak{s}}(\{p\})(i) = \{p\}$ for all $i \in \mathbb{N}$ to stutter on the last place of a finite flow chain suffix.

The *semantics* of a computation tree logic formula $\varphi \in \mathtt{CTL}^*$ is evaluated on a given run $\beta = (\mathcal{N}^R, \rho)$ of the Petri net with transits $\mathcal{N}$ and a state $s \in \mathcal{S}$ of a trace $\sigma(\xi)$ of a flow chain suffix ξ or the trace itself:

$\beta, s \models_{\mathtt{CTL}^*} a$	iff	$a \in \rho(s)$
$\beta, s \models_{\mathtt{CTL}^*} \neg \Phi$	iff	not $\beta, s \models_{\mathtt{CTL}^*} \Phi$
$\beta, s \models_{\mathtt{CTL}^*} \Phi_1 \wedge \Phi_2$	iff	$\beta, s \models_{\mathtt{CTL}^*} \Phi_1$ and $\beta, s \models_{\mathtt{CTL}^*} \Phi_2$
$\beta, s \models_{\mathtt{CTL}^*} \mathbf{E}\, \phi$	iff	there *exists* some flow chain suffix $\xi = t_0, p_0, \dots$ of β with $p_0 \in s$ such that $\beta, \sigma(\xi) \models_{\mathtt{CTL}^*} \phi$ holds for $s \not\subseteq \mathcal{P}$ and $\beta, \sigma_{\mathfrak{s}}(s) \models_{\mathtt{CTL}^*} \phi$ holds for $s \subseteq \mathcal{P}$
$\beta, \sigma \models_{\mathtt{CTL}^*} \Phi$	iff	$\beta, \sigma(0) \models_{\mathtt{CTL}^*} \Phi$
$\beta, \sigma \models_{\mathtt{CTL}^*} \neg \phi$	iff	not $\beta, \sigma \models_{\mathtt{CTL}^*} \phi$
$\beta, \sigma \models_{\mathtt{CTL}^*} \phi_1 \wedge \phi_2$	iff	$\beta, \sigma \models_{\mathtt{CTL}^*} \phi_1$ and $\beta, \sigma \models_{\mathtt{CTL}^*} \phi_2$
$\beta, \sigma \models_{\mathtt{CTL}^*} \mathbf{X}\, \phi$	iff	$\beta, \sigma^1 \models_{\mathtt{CTL}^*} \phi$
$\beta, \sigma \models_{\mathtt{CTL}^*} \phi_1 \mathbf{U}\, \phi_2$	iff	there exists some $j \geq 0$ with $\beta, \sigma^j \models_{\mathtt{CTL}^*} \phi_2$ and for all $0 \leq i < j$ the following holds: $\beta, \sigma^i \models_{\mathtt{CTL}^*} \phi_1$

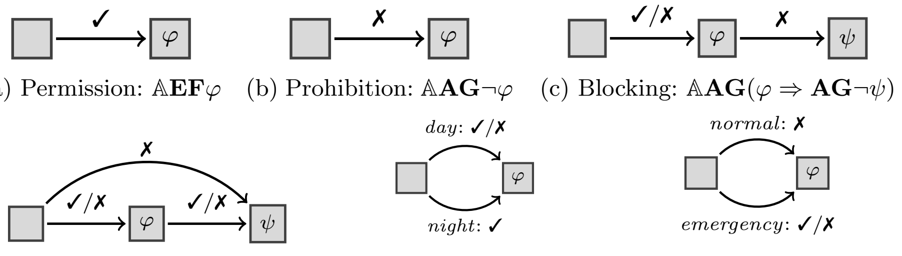
with atomic propositions $a \in AP$, state formulas Φ, Φ_1, and Φ_2, and path formulas ϕ, ϕ_1, and ϕ_2. Note that since the formulas are evaluated on the runs of $\mathcal{N}$, the branching is in the transitions and not in the places of $\mathcal{N}$.

4.3 Flow-CTL*

Like in [8], we use Petri nets with transits to enable reasoning about two separate timelines. Properties defined on the run of the system concern the *global* timeline and allow to reason about the global behavior of the system like its general control or fairness. Additionally, we can express requirements about the individual data flow like the access possibilities of people in buildings. These requirements concern the *local* timeline of the specific data flow. In Flow-CTL*, we can reason about these two parts with LTL in the *run* and with CTL* in the *flow* part of the formula. This is reflected in the following *syntax*:

$$\Psi ::= \psi \mid \Psi_1 \wedge \Psi_2 \mid \Psi_1 \vee \Psi_2 \mid \psi \to \Psi \mid \mathbb{A}\, \varphi$$

where Ψ, Ψ_1, Ψ_2 are Flow-CTL* formulas, ψ is an LTL formula, and φ is a CTL* formula. We call $\varphi_{\mathbb{A}} = \mathbb{A}\, \varphi$ *flow formulas* and all other subformulas *run formulas*.



(a) Permission: $\mathbb{A}\mathbf{EF}\varphi$ (b) Prohibition: $\mathbb{A}\mathbf{AG}\neg\varphi$ (c) Blocking: $\mathbb{A}\mathbf{AG}(\varphi \Rightarrow \mathbf{AG}\neg\psi)$

(d) Way-pointing: $\mathbb{A}\mathbf{A}(\varphi\mathbf{R}\neg\psi)$ (e) Policy update: $\mathbb{A}\mathbf{AG}(\mathit{time} \Rightarrow \mathbf{EF}\varphi)$ (f) Emergency situation: $\mathbb{A}\mathbf{A}(\mathbf{AG}\neg\varphi\mathbf{UX}\mathit{emergency})$

Fig. 4: Illustrations for standard properties of physical access control are depicted. Gray boxes represent rooms and arrows represent directions of doors that can be opened (✓), closed (✗), or are not affected by the property (✓/✗).

The *semantics* of a Petri net with transits $\mathscr{N} = (\mathscr{P}, \mathscr{T}, \mathscr{F}, In, \Upsilon)$ satisfying a Flow-CTL* formula Ψ is defined over the covering firing sequences of its runs:

$$
\begin{array}{lll}
\mathscr{N} \models \Psi & \text{iff} & \text{for all runs } \beta \text{ of } \mathscr{N} : \beta \models \Psi \\
\beta \models \Psi & \text{iff} & \text{for all firing sequences } \zeta \text{ covering } \beta : \ \beta, \sigma(\zeta) \models \Psi \\
\beta, \sigma \models \psi & \text{iff} & \sigma \models_{\texttt{LTL}} \psi \\
\beta, \sigma \models \Psi_1 \wedge \Psi_2 & \text{iff} & \beta, \sigma \models \Psi_1 \text{ and } \beta, \sigma \models \Psi_2 \\
\beta, \sigma \models \Psi_1 \vee \Psi_2 & \text{iff} & \beta, \sigma \models \Psi_1 \text{ or } \beta, \sigma \models \Psi_2 \\
\beta, \sigma \models \psi \rightarrow \Psi & \text{iff} & \beta, \sigma \models \psi \text{ implies } \beta, \sigma \models \Psi \\
\beta, \sigma \models \mathbb{A}\,\varphi & \text{iff} & \text{for all flow chains } \xi \text{ of } \beta : \beta, \sigma(\xi) \models_{\texttt{CTL}^*} \varphi
\end{array}
$$

Due to the covering of the firing sequences and the maximality constraint of the flow chain suffixes, every behavior of the run is incorporated. The operator $\mathbb{A}$ chooses flow chains rather than flow trees as our definition is based on the common semantics of CTL* over paths. Though it suffices to find one of the possibly infinitely many flow trees for each flow formula to invalidate the subformula, checking the data flow while the control changes the system complicates the direct expression of the model checking problem within a finite model. In Sect. 6, we introduce a general reduction method for a model with a finite state space.

5 Example Specifications

We illustrate Flow-CTL* with examples from the literature on physical access control [18,13]. Branching properties like permission and way-pointing are given as flow formulas, linear properties like fairness and maximality as run formulas.

5.1 Flow Formulas

Figure 4 illustrates six typical specifications for physical access control [18,13].

Permission. Permission (cf. Fig. 4a) requires that a subformula φ can be reached on one path ($\mathbb{A}\mathbf{EF}\varphi$). In our running example, permission can be required for the *hall* and the *lab*. Permission can be extended as it requires reaching the subformula once. Persistent permission then requires that, on all paths, the subformula φ can be repeatedly reached on a path ($\mathbb{A}\mathbf{AGEF}\varphi$).
Prohibition. Prohibition (cf. Fig. 4b) requires that a subformula φ, for example representing a room, can never be reached on any path ($\mathbb{A}\mathbf{AG}\neg\varphi$). In our running example, closing the door to the *kitchen* would satisfy prohibition for the *kitchen*.
Blocking. Blocking (cf. Fig. 4c) requires for all paths globally that, after reaching subformula φ, the subformula ψ cannot be reached ($\mathbb{A}\mathbf{AG}(\varphi \Rightarrow \mathbf{AG}\neg\psi)$). This can be used to allow a new employee to only enter one of many labs.
Way-pointing. Way-pointing (cf. Fig. 4d) ensures for all paths that subformula ψ can only be reached if φ was reached before ($\mathbb{A}\mathbf{A}(\varphi\mathbf{R}\neg\psi)$). This can be used to enforce a mandatory security check when entering a building.
Policy update. A policy update (cf. Fig. 4e) allows access to subformula φ according to a time schedule ($\mathbb{A}\mathbf{AG}(\mathit{time} \Rightarrow \mathbf{EF}\varphi)$) with *time* being a transition. This can be used to restrict access during the night.
Emergency. An emergency situation (cf. Fig. 4f) can revoke the prohibition of subformula φ at an arbitrary time ($\mathbb{A}\mathbf{A}(\mathbf{AG}\neg\varphi\mathbf{UX}\mathit{emergency})$) with *emergency* being a transition. An otherwise closed door could be opened to evacuate people. The next operator $\mathbf{X}$ is necessary because of the ingoing semantics of Flow-CTL*.

5.2 Run Formulas

Flow formulas require behavior on the maximal flow of people in the building. Doors are assumed to allow passthrough in a fair manner. Both types of assumptions are expressed in Flow-CTL* as run formulas.
Maximality. A run β is *interleaving-maximal* if, whenever some transition is enabled, some transition will be taken: $\beta \models \Box(\bigvee_{t\in\mathscr{T}} \mathit{pre}(t) \rightarrow \bigvee_{t\in\mathscr{T}} \bigcirc t)$. A run β is *concurrency-maximal* if, when a transition t is from a moment on always enabled, infinitely often a transition t' (including t itself) sharing a precondition with t is taken: $\beta \models \bigwedge_{t\in\mathscr{T}}(\Diamond\Box\ \mathit{pre}(t) \rightarrow \Box\Diamond \bigvee_{p\,\in\,\mathit{pre}(t),t'\,\in\,\mathit{post}(p)}\ t')$.
Fairness. A run β is *weakly fair* w.r.t. a transition t if, whenever t is always enabled after some point, t is taken infinitely often: $\beta \models \Diamond\Box\ \mathit{pre}(t) \rightarrow \Box\Diamond\, t$.
A run β is *strongly fair* w.r.t. t if, whenever t is enabled infinitely often, t is taken infinitely often: $\beta \models \Box\Diamond\ \mathit{pre}(t) \rightarrow \Box\Diamond\, t$.

6 Model Checking Flow-CTL* on Petri Nets with Transits

We solve the model checking problem for a given Flow-CTL* formula Ψ and a safe Petri net with transits $\mathscr{N}$ in four steps:

1. For each flow subformula $\mathbb{A}\,\varphi_i$ of Ψ, a subnet $\mathscr{N}_i^{>}$ is created via a sequence of automata constructions which allows to guess a counterexample, i.e., a flow tree not satisfying φ_i, and to check for its correctness.

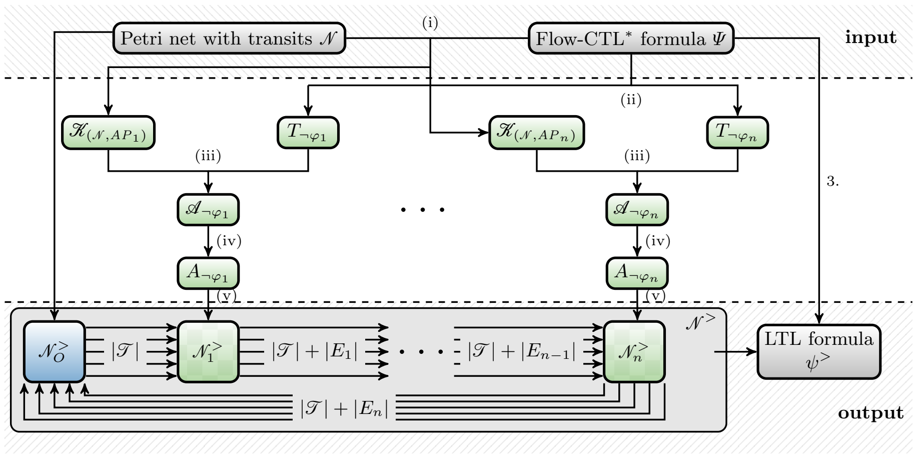


Fig. 5: Overview of the model checking procedure: For a given safe Petri net with transits $\mathcal{N}$ and a Flow-CTL* formula Ψ, a standard Petri net $\mathcal{N}^>$ and an LTL formula $\psi^>$ are created: For each flow subformula $\mathbb{A}\,\varphi_i$, create (i) a labeled Kripke structure $\mathcal{K}_{(\mathcal{N},AP_i)}$ and (ii) the alternating tree automaton $T_{\neg\varphi_i}$, construct (iii) the alternating word automaton $\mathcal{A}_{\neg\varphi_i} = T_{\neg\varphi_i} \times \mathcal{K}_{(\mathcal{N},AP_i)}$, and from that (iv) the Büchi automaton $A_{\neg\varphi_i}$ with edges E_i, which then (v) is transformed into a Petri net $\mathcal{N}_i^>$. These subnets are composed to a Petri net $\mathcal{N}^>$ such that they get subsequently triggered for every transition fired by the original net. The constructed formula $\psi^>$ skips for the run part of Ψ these subsequent steps and checks the acceptance of the guessed tree for each automaton. The problem is then solved by checking $\mathcal{N}^> \models_{\texttt{LTL}} \psi^>$.

2. The Petri net $\mathcal{N}^>$ is created by composing the subnets $\mathcal{N}_i^>$ to a copy of $\mathcal{N}$ such that every firing of a transition subsequently triggers each subnet.
3. The formula $\Psi^>$ is created such that the subnets $\mathcal{N}_i^>$ are adequately skipped for the run part of Ψ, and the flow parts are replaced by LTL formulas checking the acceptance of a run of the corresponding automaton.
4. $\mathcal{N}^> \models_{\texttt{LTL}} \Psi^>$ is checked to answer $\mathcal{N} \models \Psi$.

The construction from a given safe Petri net with transits $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In, \Upsilon)$ and a Flow-CTL* formula Ψ with $n \in \mathbb{N}$ flow subformulas $\varphi_{\mathbb{A}_i} = \mathbb{A}\,\varphi_i$ with atomic propositions AP_i to a Petri net $\mathcal{N}^> = (\mathcal{P}^>, \mathcal{T}^>, \mathcal{F}^>, \mathcal{F}_I^>, In^>)$ with inhibitor arcs (denoted by $\mathcal{F}_I^>$) and an LTL formula $\Psi^>$ is defined in the following sections. More details and proofs can be found in the full paper [10]. An *inhibitor arc* connects a place p and a transition t of a Petri net such that t is only enabled when p is empty. Figure 5 gives a schematic overview of the procedure.

6.1 Automaton Construction for Flow Formulas

In Step 1, we create for each flow subformula $\mathbb{A}\,\varphi_i$ of Ψ with atomic propositions AP_i a nondeterministic Büchi automaton $A_{\neg\varphi_i}$ which accepts a sequence of transitions of a given run if the corresponding flow tree satisfies $\neg\varphi_i$. This construction has four steps:

(i) Create the labeled Kripke structure $\mathscr{K}_{(\mathscr{N},AP_i)}$ which, triggered by transitions $t \in \mathscr{T}$, tracks every flow chain of $\mathscr{N}$. Each path corresponds to a flow chain.
(ii) Create the alternating tree automaton $T_{\neg\varphi_i}$ for the negation of the CTL* formula φ_i and the set of directions $\mathscr{D} \subseteq \{0,\dots,\mathtt{max}\{|post^{\Upsilon}(p,t)|-1 \mid p \in \mathscr{P} \wedge t \in post^{\mathscr{N}}(p)\}\}$ which accepts all 2^{AP_i}-labeled trees with nodes of degree in $\mathscr{D}$ satisfying $\neg\varphi_i$ [15].
(iii) Create the alternating word automaton $\mathscr{A}_{\neg\varphi_i} = T_{\neg\varphi_i} \times \mathscr{K}_{(\mathscr{N},AP_i)}$ like in [15].
(iv) Alternation elimination for $\mathscr{A}_{\neg\varphi_i}$ yields the nondeterministic Büchi automaton $A_{\neg\varphi_i}$ [16,4].

Step (ii) and Step (iv) are well-established constructions. For Step (iii), we modify the construction of [15] by applying the algorithm for the groups of equally labeled edges. By this, we obtain an alternating word automaton with the alphabet $A = \mathscr{T} \cup \{\mathfrak{s}\}$ of the labeled Kripke structure rather than an alternating word automaton over a 1-letter alphabet. This allows us to check whether the, by the input transition dynamically created, system satisfies the CTL* subformula φ_i.

Step (i) of the construction creates the labeled Kripke structure $\mathscr{K}_{(\mathscr{N},AP_i)} = (AP, S, S_0, L, A, R)$ with a set of *atomic propositions* $AP = AP_i$, a finite set of *states* $S = ((\mathscr{T} \cap AP) \times \mathscr{P}) \cup \mathscr{P}$, the *initial states* $S_0 \subseteq S$, the *labeling function* $L : S \to 2^{AP}$, the *alphabet* $A = \mathscr{T} \cup \{\mathfrak{s}\}$, and the *labeled transition relation* $R \subseteq S \times A \times S$. The Kripke structure serves (in combination with the tree automaton) for checking the satisfaction of a flow tree of a given run. Hence, the states track the current place of the considered chain of the tree and additionally, when the transition extending the chain into the place occurs in the formula, also this ingoing transition. The initial states S_0 are either the tuples of transitions t_j and places p_j which start a flow chain, i.e., all $(t_j, p_j) \in \mathscr{T} \times \mathscr{P}$ with $(\triangleright, p_j) \in \Upsilon(t_j)$ when $t_j \in AP$ or only the place p_j otherwise. The labeling function L labels the states with its components. The transition relation R connects the states with respect to the transits, connects each state $(t,p) \in S$ with $\mathfrak{s}$-labeled edges to the state $p \in S$, and loops with $\mathfrak{s}$-labeled edges in states $s \in \mathscr{P}$ to allow for the stuttering of finite chains.

Lemma 1 (Size of the Kripke Structure). *The constructed Kripke structure $\mathscr{K}_{(\mathscr{N},AP_i)}$ has $O(|AP_i \cap \mathscr{T}| \cdot |\mathscr{N}| + |\mathscr{N}|)$ states and $O(|\mathscr{N}^3|)$ edges.*

Note that the number of edges stems from the number of transits $(p,t,q) \in \mathscr{P} \times \mathscr{T} \times \mathscr{P}$ used in the Petri net with transits $\mathscr{N}$.

The size of the Büchi automaton is dominated by the tree automaton construction and the removal of the alternation. Each construction adds one exponent for CTL*.

Lemma 2 (Size of the Büchi Automaton). *The size of the Büchi automaton $A_{\neg\varphi_i}$ is in $O(2^{2^{|\varphi|\cdot|\mathscr{N}|^3}})$ for specifications φ_i in CTL* and in $O(2^{|\varphi|\cdot|\mathscr{N}|^3})$ for specifications in CTL.*

6.2 From Petri Nets with Transits to Petri Nets

In Step 2, we construct for the Petri net with transits $\mathcal{N}$ and the Büchi automata $A_{\neg\varphi_i}$ for each flow subformula $\varphi_{\mathbb{A}i} = \mathbb{A}\,\varphi_i$ of Ψ, a Petri net $\mathcal{N}^>$ by composing a copy of $\mathcal{N}$ (without transits), denoted by $\mathcal{N}_O^>$, to subnets $\mathcal{N}_i^>$ corresponding to $A_{\neg\varphi_i}$ such that each copy is sequentially triggered when a transition of $\mathcal{N}_O^>$ fires. The subnet $\mathcal{N}_i^>$, when triggered by transitions $t \in \mathcal{T}$, guesses nondeterministically the violating flow tree of the operator $\mathbb{A}$ and simulates $A_{\neg\varphi_i}$. Thus, a token from the initially marked place $[\iota]_i$ is moved via a transition for each transition $t \in \mathcal{T}$ starting a flow chain to the place corresponding to the initial state of $A_{\neg\varphi_i}$. For each state s of $A_{\neg\varphi_i}$, we have a place $[s]_i$, and, for each edge (s, l, s'), a transition labeled by l which moves the token from $[s]_i$ to $[s']_i$.

There are two kinds of stutterings: *global* stuttering for finite runs and *local* stuttering for finite flow chains. To guess the starting time of both stutterings, there is an initially marked place $\boldsymbol{N}$, a place $\boldsymbol{S}$, and a transition which can switch from *normal* to *stuttering* mode for the global stuttering in $\mathcal{N}_O^>$ and for the local stutterings in each subnet $\mathcal{N}_i^>$ (denoted by $[\boldsymbol{N}]$,$[\boldsymbol{S}]$). The original transitions of $\mathcal{N}_O^>$ and the transitions of a subnet $\mathcal{N}_i^>$ corresponding to a transition $t \in \mathcal{T}$ depend on the normal mode. The $\mathfrak{s}$-labeled transitions (used for global stuttering) of the subnet depend on the stuttering mode. To enable local stuttering, we add, for each edge $e = (s, \mathfrak{s}, s')$ of $A_{\neg\varphi_i}$, a transition $t^>$ for each transition $t \in \mathcal{T}$ for which no edge (s, t, s'') exists in $A_{\neg\varphi_i}$. These transitions depend on the stuttering mode and move the token according to their corresponding edge e.

The original part $\mathcal{N}_O^>$ and the subnets $\mathcal{N}_i^>$ are connected in a sequential manner. The net $\mathcal{N}_O^>$ has an initially marked activation place $\rightarrow_o$ in the preset of each transition, the subnets have one activation place $[\rightarrow_t]$ in the preset of every transition $t^>$ corresponding to a transition $t \in \mathcal{T}$ (normal as well as stuttering). The transitions move the activation token to the corresponding places of the next subnet (or back to $\mathcal{N}_O^>$). To ensure the continuation even though the triggering transition does not extend the current flow tree (e.g., because it is a concurrent transition of the run), there is a skipping transition for each transition $t \in \mathcal{T}$ which moves the activation token when none of the states having a successor edge labeled with t are active. For the global stuttering, each subnet has an activation place $[\rightarrow_{\mathfrak{s}}]_i$, in which an additional transition $t_{\mathfrak{s}}$ in $\mathcal{N}_O^>$ puts the active token if the stuttering mode of $\mathcal{N}_O^>$ is active. Each $\mathfrak{s}$-labeled transition of the subnets moves this token to the next subnet (or back to $\mathcal{N}_O^>$).

By that, we can check the acceptance of each $A_{\neg\varphi_i}$ by checking if the subnet infinitely often reaches any places corresponding to a Büchi state of $A_{\neg\varphi_i}$. This and only allowing to correctly guess the time point of the stutterings is achieved with the formula described in Sect. 6.3. A formal definition is given in in the full paper [10]. The size of the constructed Petri net is dominated by the respective single- or double-exponential size of the nondeterministic Büchi automata.

Lemma 3 (Size of the Constructed Net). *The constructed Petri net with inhibitor arcs $\mathcal{N}^>$ for a Petri net with transits $\mathcal{N}$ and n nondeterministic Büchi automata $A_{\neg\varphi_i} = (\mathcal{T} \cup \{\mathfrak{s}\}, Q_i, I_i, E_i, F_i)$ has $\boldsymbol{O}(|\mathcal{N}| \cdot n + |\mathcal{N}| + \sum_{i=1}^{n} |Q_i|)$ places and $\boldsymbol{O}(|\mathcal{N}|^2 \cdot n + |\mathcal{N}| + \sum_{i=1}^{n} |E_i| + |\mathcal{N}| \cdot \sum_{i=1}^{n} |Q_i|)$ transitions.*

6.3 From Flow-CTL* Formulas to LTL Formulas

The formula transformation from a given Flow-CTL* formula Ψ and a Petri net with transits $\mathcal{N}$ into an LTL formula (Step 3) consists of three parts:

First, we substitute the flow formulas $\varphi_{\mathbb{A}i} = \mathbb{A}\,\varphi_i$ with the acceptance check of the corresponding automaton $A_{\neg\varphi_i}$, i.e., we substitute $\varphi_{\mathbb{A}i}$ with $\neg\Box\Diamond\bigvee_{b\in F_i}[b]_i$ for the Büchi states F_i of $A_{\neg\varphi_i}$.

Second, the sequential manner of the constructed net $\mathcal{N}^>$ requires an adaptation of the run part of Ψ. For a subformula $\psi_1\,\mathcal{U}\,\psi_2$ with transitions $t\in\mathcal{T}$ as atomic propositions or a subformula $\bigcirc\,\psi$ in the run part of Ψ, the sequential steps of the subnets have to be skipped. Let $\mathcal{T}_O^>$ be the transition of the original copy $\mathcal{N}_O^>$, $\mathcal{T}_i^>$ the transitions of the subnet $\mathcal{N}_i^>$, $\mathcal{T}_{\not\Rightarrow_i}$ the transitions of the subnet $\mathcal{N}_i^>$ which skip the triggering of the automaton in the normal mode, and $t_{\mathcal{N}\to\mathcal{S}}$ the transition switching $\mathcal{N}_O^>$ from normal to stuttering mode. Then, because of the ingoing semantics, we can can select all states corresponding to the run part with $\mathtt{M}=\bigvee_{t\in\mathcal{T}_O^>\setminus\{t_{\mathcal{N}\to\mathcal{S}}\}} t$ together with the initial state $\mathtt{i}=\neg\bigvee_{t\in\mathcal{T}^>} t$. Hence, we replace each subformula $\psi_1\,\mathcal{U}\,\psi_2$ containing transitions $t\in\mathcal{T}$ as atomic propositions with $((\mathtt{M}\vee\mathtt{i})\to\psi_1)\,\mathcal{U}((\mathtt{M}\vee\mathtt{i})\to\psi_2)$ from the inner- to the outermost occurrence. For the next operator, the second state is already the correct next state of the initial state also in the sense of the global timeline of $\psi^>$. For all other states belonging to the run part (selected by the until construction above), we have to get the next state and then skip all transitions of the subnet. Thus, we replace each subformula $\bigcirc\,\psi$ with $\mathtt{i}\to\bigcirc\,\psi\wedge\neg\mathtt{i}\to\bigcirc(\bigvee_{t\in\mathcal{T}^>\setminus\mathcal{T}_O^>} t\,\mathcal{U}\bigvee_{t'\in\mathcal{T}_O^>} t'\wedge\psi)$ from the inner- to the outermost occurrence.

Third, we have to ensure the correct switching into the stuttering mode. By $\mathtt{skip_i}=\neg\Diamond\Box((\bigvee_{t\in\mathcal{T}_i^>} t)\to(\bigvee_{t'\in\mathcal{T}_{\not\Rightarrow_i}} t'))$ a subnet is enforced to switch into its stuttering mode if necessary. If it wrongly selects the time point of the global stuttering, the run stops. Hence, we obtain the formula $\psi^>=((\Box\Diamond\to_o)\wedge\bigwedge_{i\in\{1,\ldots,n\}}\mathtt{skip_i})\to\psi$ by only selecting the runs where the original part is infinitely often activated and each subnet chooses its stuttering mode correctly.

Since the size of the formula depends on the size of the constructed Petri net $\mathcal{N}^>$, it is also dominated by the Büchi automaton construction.

Lemma 4 (Size of the Constructed Formula). *The size of the constructed formula $\psi^>$ is double-exponential for specifications given in CTL* and single-exponential for specifications in CTL.*

We can show that the construction of the net and the formula adequately fit together such that the additional sequential steps of the subnets are skipped in the formula and the triggering of the subnets simulating the Büchi automata as well as the stuttering is handled properly.

Lemma 5 (Correctness of the Transformation). *For a Petri net with transits $\mathcal{N}$ and a Flow-CTL* formula Ψ, there exists a safe Petri net $\mathcal{N}^>$ with inhibitor arcs and an LTL formula $\Psi^>$ such that $\mathcal{N}\models\Psi$ iff $\mathcal{N}^>\models_{\mathtt{LTL}}\Psi^>$.*

The complexity of the model checking problem of Flow-CTL* is dominated by the automata constructions for the CTL* subformulas. The need of the alternation removal (Step (iv) of the construction) is due to the checking of branching properties on structures chosen by linear properties. In contrast to standard CTL* model checking on a static Kripke structure, we check on Kripke structures dynamically created for specific runs.

Theorem 1. *A safe Petri net with transits $\mathcal{N}$ can be checked against a Flow-CTL* formula Ψ in triple-exponential time in the size of $\mathcal{N}$ and Ψ. For a Flow-CTL formula Ψ', the model checking algorithm runs in double-exponential time in the size of $\mathcal{N}$ and Ψ'.*

Note that a single-exponential time algorithm for Flow-LTL is presented in [8].

7 Related Work

There is a large body of work on physical access control: Closest to our work are *access nets* [12] which extend Petri nets with mandatory transitions to make people leave a room at a policy update. Branching properties can be model checked for a fixed number of people in the building. Fixing the number of people enables explicit interaction between people. In logic-based access-control frameworks, credentials are collected from distributed components to open policy enforcement points according to the current policy [2,3]. Techniques from networking can be applied to physical access control to detect redundancy, shadowing, and spuriousness in policies [11]. Our model prevents such situations by definition as a door can be either open or closed for people with the same access rights.

A user study has been carried out to identify the limitations of physical access control for real-life professionals [1]. Here, it was identified that policies are made by multiple people which is a problem our approach of global control solves. Types of access patterns are also studied [7,13,18]: Access policies according to time schedules and emergencies, access policies for people without RFID cards, and dependent access are of great importance. The first and the third problem are solvable by our approach and the second one seems like an intrinsic problem to physical access control. Policies for physical access control can be synthesized if no policy updates are necessary [18]. It is an interesting open question whether policy updates can be included in the synthesis of access policies.

8 Conclusion

We present the first model checking approach for the verification of physical access control with policy updates under fairness assumptions and with an unbounded number of people. Our approach builds on Petri nets with transits which superimpose a transit relation onto the flow relation of Petri nets to differentiate between data flow and control. We introduce Flow-CTL* to specify branching properties on the data flow and linear properties on the control in Petri nets with transits. We outline how Petri nets with transits can model physical access control with policy updates and how Flow-CTL* can specify properties on the behavior before, during, and after updates including fairness and maximality. To solve the model checking problem, we reduce the model checking problem of Petri nets with transits against Flow-CTL* via automata constructions to the model checking problem of Petri nets against LTL. In the future, we plan to evaluate our approach in a tool implementation and a corresponding case study. We can build on our tool AdamMC [9] for Petri nets with transits and Flow-LTL.

References

1. Bauer, L., Cranor, L.F., Reeder, R.W., Reiter, M.K., Vaniea, K.: Real life challenges in access-control management. In: Proc. of CHI (2009)
2. Bauer, L., Garriss, S., Reiter, M.K.: Distributed proving in access-control systems. In: Proc. of S&P (2005)
3. Bauer, L., Garriss, S., Reiter, M.K.: Efficient proving for practical distributed access-control systems. In: Proc. of ESORICS (2007)
4. Dax, C., Klaedtke, F.: Alternation elimination by complementation (extended abstract). In: Proc. of LPAR (2008)
5. Engelfriet, J.: Branching processes of Petri nets. Acta Inf. **28**(6) (1991)
6. Esparza, J., Heljanko, K.: Unfoldings – A Partial-Order Approach to Model Checking. Springer (2008)
7. Fernández, E.B., Ballesteros, J., Desouza-Doucet, A.C., Larrondo-Petrie, M.M.: Security patterns for physical access control systems. In: Proc. of Data and Applications Security XXI (2007)
8. Finkbeiner, B., Gieseking, M., Hecking-Harbusch, J., Olderog, E.: Model checking data flows in concurrent network updates. In: Proc. of ATVA (2019)
9. Finkbeiner, B., Gieseking, M., Hecking-Harbusch, J., Olderog, E.: AdamMC: A model checker for Petri nets with transits against Flow-LTL. In: Proc. of CAV (2020)
10. Finkbeiner, B., Gieseking, M., Hecking-Harbusch, J., Olderog, E.: Model checking branching properties on Petri nets with transits (full version). arXiv preprint arXiv:2007.07235 (2020)
11. Fitzgerald, W.M., Turkmen, F., Foley, S.N., O'Sullivan, B.: Anomaly analysis for physical access control security configuration. In: Proc. of CRiSIS (2012)
12. Frohardt, R., Chang, B.E., Sankaranarayanan, S.: Access nets: Modeling access to physical spaces. In: Proc. of VMCAI (2011)
13. Geepalla, E., Bordbar, B., Du, X.: Spatio-temporal role based access control for physical access control systems. In: Proc. of EST (2013)
14. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume 1. Springer (1992)
15. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. ACM **47**(2) (2000)
16. Miyano, S., Hayashi, T.: Alternating finite automata on omega-words. Theor. Comput. Sci. **32** (1984)
17. Reisig, W.: Petri Nets: An Introduction. Springer (1985)
18. Tsankov, P., Dashti, M.T., Basin, D.A.: Access control synthesis for physical spaces. In: Proc. of CSF (2016)
19. Welbourne, E., Battle, L., Cole, G., Gould, K., Rector, K., Raymer, S., Balazinska, M., Borriello, G.: Building the internet of things using RFID: the RFID ecosystem experience. IEEE Internet Comput. **13**(3) (2009)