Access Nets: Modeling Access to Physical Spaces

Robert Frohardt, Bor-Yuh Evan Chang, and Sriram Sankaranarayanan

University of Colorado, Boulder, Colorado, USA
{frohardt,bec,srirams}@cs.colorado.edu

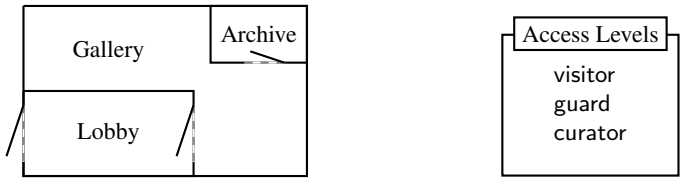
Abstract. Electronic, software-managed mechanisms using, for example, radio-frequency identification (RFID) cards, enable great flexibility in specifying access control policies to physical spaces. For example, access rights may vary based on time of day or could differ in normal versus emergency situations. With such fine-grained control, understanding and reasoning about what a policy permits becomes surprisingly difficult requiring knowledge of permission levels, spatial layout, and time. In this paper, we present a formal modeling framework, called ACCESS NETS, suitable for describing a combination of access permissions, physical spaces, and temporal constraints. Furthermore, we provide evidence that model checking techniques are effective in reasoning about physical access control policies. We describe our results from a tool that uses reachability analysis to validate security policies.

1 Introduction

Access to physical spaces such as buildings, museums, airports, and chemical plants is increasingly mediated by electronic, software-controlled mechanisms. These mechanisms combine traditional human mediation, mechanical lock-and-keys, as well as electronic technologies such as radio-frequency identification (RFID) cards. The use of computerized access control in these systems is on the rise, as they enable highly flexible policies. Computerized access control policies enable administrators to add or remove access to key personnel or specify policies that may vary depending on the time of the day (working hours versus evenings), day of the week (weekdays versus weekends), and months in the year (summer versus fall). These policies can even be automatically changed in response to emergencies such as a fire in the building—in contrast to access policies mediated using only mechanical lock-and-key.

In this paper, we address formal modeling and verification of access control policies for physical spaces. Our approach combines dynamic models of access control policies in physical spaces with an application of model-checking techniques. In particular, we make the following contributions:

- We present a formal framework ACCESS NETS for the modeling of access control in physical spaces, such as offices or buildings (Sect. 3). Our framework models the topology of the physical space, as well as the movement of personnel with various access levels in this space. Our model of access control accommodates rich specifications, including those that depend on time.



P1	The visitors may only be in the museum between 9:00 a.m. and 5:00 p.m.
P2	The visitors may only enter the archive with guard escort during museum hours.
P3	The curators may enter the museum and the archive at any time.

Fig. 1. A floor plan, access control roles (top) and access control policy (bottom) for a museum.

- We demonstrate a new and compelling application of formal verification techniques, like model checking. While software-managed access control systems may be large and complex, we see that well-known state-space reduction techniques are surprisingly effective in reducing the size of the model. Thus, we identify a new domain where model checking techniques are particularly apt (Sect. 4).
- We provide evidence for the applicability of our techniques through an initial case study (Sect. 5). In particular, we observe that our ACCESS NET-specific reduction techniques are quite effective in reducing the state space.

Motivating Example. Figure 1 outlines a simple floor plan and an access control policy for a fictitious museum. The museum has a main entrance leading into a lobby. The lobby in turn leads into a gallery, which is connected to an archive. The main entrance and the entrance to the archive have key card readers. The archive entrance is staffed by a guard during opening hours. The access control policies are also described in Fig. 1.

Given such a policy specification, we wish to verify that the access control mechanisms support it. For example, is it possible for a visitor to be in the archive after hours? Can curators access the archive at any time? In general, it is hard to manually consider all the relevant scenarios, especially for larger buildings with more complex access control policies. Therefore, we desire a formal framework that captures the relevant details of such systems and enables automatic verification.

2 Overview

In this section, we present an overview of the main features in the ACCESS NETS model, using the museum example shown in Fig. 1. Note that we are not interested in details like the precise spatial layout of the building (e.g., coordinates). Thus, we seek a graph-like model that captures connectivity but abstracts spatial layout.

Drawing inspiration from Petri nets [19], we use tokens to model persons and transitions to capture the movement of persons from one place to another. Each transition has at least one incoming and one outgoing arc. Transitions move tokens *one-way* from their input places to their output places. This captures common situations wherein, a key is needed to enter a room but not needed to exit. The ACCESS NET model for the museum example has the graph structure shown in Fig. 2.

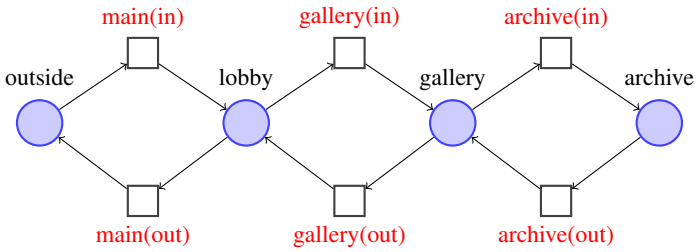


Fig. 2. The graph structure of the ACCESS NETS model for the example in Fig. 1.

Token Types. Each token in our model has an associated type that represents its access level (e.g., visitor, guard, curator, administrator, or supervisor). Transitions are enabled based on the number of tokens of each type from each of their input places. For example, the rule that a visitor may only enter the archive under guard escort (rule P2 in Fig. 1) is shown in Fig. 3. Both the incoming and outgoing arcs of the archive(in) transition are annotated with 1 guard and 1 visitor. These labels specify the enabling condition that there must be a guard and a visitor present in the gallery.

Transition Firings. Transitions whose input conditions are satisfied may fire non-deterministically to yield a next state based on the output conditions of the transition. Thus, for example, to capture that a curator may enter the archive herself without guard escort (rule P3), we can simply add a separate transition with arcs from the gallery and to the archive each labeled with 1 curator.

Time. Some transition rules depend on the time of day. For example, anyone may enter the museum between 9:00 a.m. and 5:00 p.m. To model time, we add a global clock to an ACCESS NET state and a set of time intervals to the enabling condition of each transition. For instance, we can associate a set of times $[9, 17]$ with the main(in) transition in Fig. 3, so that it is enabled between 9:00 a.m. and 5:00 p.m.

Mandatory Transitions. Recall that visitors may be in the museum only between 9:00 a.m. and 5:00 p.m. (rule P1), so not only do we allow visitors to enter during those hours, but we must require visitors to leave when the museum closes at 5:00 p.m. To do so, we introduce the notion of a *mandatory transition*. At any state, if any mandatory transition is enabled, one of them must be taken next (see Sect. 3). In this scenario, we add mandatory transitions from the archive, gallery, and lobby to the outside requiring the visitors to leave during the time range $[17, 17.5]$ (i.e., 5:00 p.m. to 5:30 p.m.).

3 Access Nets

In this section, we provide a formalization of ACCESS NETS. The formal model provides a basis for verification techniques (Sect. 4) and the case studies (Sect. 5).

Topology. The topology of a building is modeled using a directed graph, whose nodes include a set of *places* P and a set of *transitions* T. The *arcs*, $F \subseteq (P \times T) \cup (T \times P)$, connect places to transitions, and transitions back to places. The inset

places	$p \in P$
transitions	$t \in T$
arcs	$f \in F$

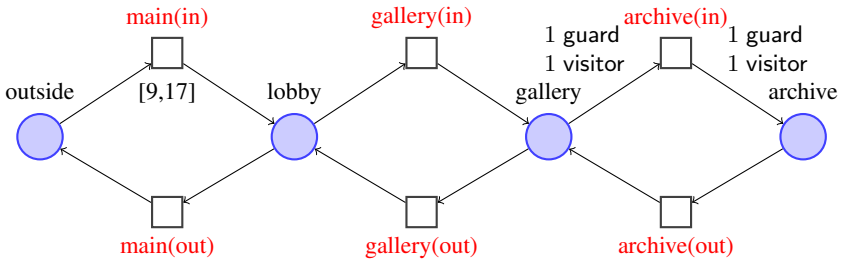


Fig. 3. The ACCESS NET model from Fig. 2 with access controls specified.

box to the right summarizes the notation used. The incoming arcs for a transition t indicate the places from which t removes tokens, and the outgoing arcs indicate the places to which t adds tokens (cf. Definition 3). Pictorially, places are denoted as circles and transitions as rectangles (cf. Fig. 2).

State: Access Types, Markings, and Time. To model various access control roles, each token is annotated with a *type* drawn from a set S. For example, in the museum example discussed previously, the set S is $\{\mathsf{visitor}, \mathsf{guard}, \mathsf{curator}\}$, representing various roles. Persons are represented by *tokens* of particular types (i.e., with particular access roles). As part of the state, we describe where people are with a *marking*.

types	$s \in S$
markings	$m : (P \times S) \to \mathbb{N}$
global time	$\tau \in [\tau_{\mathsf{min}}, \tau_{\mathsf{max}}]$
enabled times	$H : T \to \mathcal{P}([\tau_{\mathsf{min}}, \tau_{\mathsf{max}}])$
state	$\sigma = (m, \tau)$

Definition 1 (Marking). *A* marking m *is a function* $m : (P \times S) \to \mathbb{N}$ *that represents the number of tokens of type* s *in place* p.

Pictorially, a marking is denoted by drawing $m(p, s)$ dots labeled s at place p.

To model temporal access control rules, we introduce a global clock τ that is a value in a fixed range $[\tau_{\mathsf{min}}, \tau_{\mathsf{max}}]$. For example, we may choose $\tau_{\mathsf{min}} = 0$ and $\tau_{\mathsf{max}} = 24$ representing the hours of day. The framework is agnostic to translation of these values to "real time." Therefore, time can be modeled at the appropriate granularity (e.g., seconds, minutes, hours, and days). Time is updated in the ACCESS NET model by using a special *tick* transition. For each transition t, we define the "hours" function $H : T \to \mathcal{P}([\tau_{\mathsf{min}}, \tau_{\mathsf{max}}])$. For simplicity, $H(t)$ is assumed to be the union of finitely many disjoint intervals for each transition t, specifying the time instants during which the transition t can be *enabled* (cf. Definition 2). Diagrammatically, $H(t)$ is denoted by writing a range next to the transition (e.g., $[9, 17]$ in Fig. 3). The absence of such an annotation indicates that the transition is time independent (i.e., $H(t)$ is $[\tau_{\mathsf{min}}, \tau_{\mathsf{max}}]$).

A state σ of an ACCESS NET is then the pair (m, τ) consisting of its current marking and its current time.

State Transitions. The execution of an ACCESS NET models the movement of people throughout the building and the progression of time. Recall from Sect. 2 that our model contains *mandatory transitions* $M \subseteq T$ that are taken whenever enabled. Definition 2 describes the enforcement of mandatory transitions.

State transitions $\sigma \longrightarrow \sigma'$ denote a move from current state σ to a next state σ'. There are two main types of state transitions: (1) *token transitions* model the movement of people, and (2) *tick transitions* model the progression of time.

Token Transitions. An ACCESS NET has a *weight* function $W : (F \times S) \to \mathbb{N}$ that gives the number of tokens of a type s that move along each arc f during a transition.

Definition 2 (Enabled Transition). *Transition t is* enabled *in state $\sigma = (m, \tau)$ iff*

1. *The current time belongs to the permissible range: $\tau \in H(t)$.*
2. *There are sufficiently many tokens in the input places: $W(f, s) \leq m(p, s)$ for all $f : (p, t) \in \mathsf{in}(t)$ and for all $s \in S$.*
3. *If $t \notin M$, then every mandatory transition $t_m \in M$ is not enabled.*

where $\mathsf{in}(t) \stackrel{\text{def}}{=} \{(p, t) \in F \mid p \in P\}$ *and* $\mathsf{out}(t) \stackrel{\text{def}}{=} \{(t, p) \in F \mid p \in P\}$ *(i.e., the incoming and outgoing arcs of transition t, respectively).*

An enabled transition can move tokens from its input places to its output places.

Definition 3 (Token Transition). *Given state $\sigma = (m, \tau)$ and enabled transition t, a* token transition *results in a new marking m', such that*

$$m'(p, s) = m(p, s) - W(f, s), \forall\, s \in S,\ f : (p, t) \in \mathsf{in}(t).$$
$$m'(p, s) = m(p, s) + W(f, s), \forall\, s \in S,\ f : (t, p) \in \mathsf{out}(t).$$

For simplicity in presentation, $\mathsf{in}(t)$ and $\mathsf{out}(t)$ are assumed disjoint, that is, there are no self-loops. Self-loops can be eliminated by the introduction of dummy transitions and places [19]. We write such a token transition as $(m, \tau) \xrightarrow{t} (m', \tau)$.

Tick Transitions. Tick transitions model the elapse of time. For any state $\sigma = (m, \tau)$ such that $\tau \in [\tau_{\mathsf{min}}, \tau_{\mathsf{max}})$, and *no mandatory transitions are enabled*, the global time may progress to any time in $(\tau, \tau']$ where $\tau' = \mathsf{min}(\tau_M, \tau_{\mathsf{max}})$ where $\tau_M > \tau$ is the next time when some mandatory transition could be enabled. We write a time transition from τ to τ' as follows: $(m, \tau) \xrightarrow{\text{tick}} (m, \tau')$. When checking the model, not all times values τ need to be considered. Instead, time is abstracted using a region construction along the lines of Alur and Dill [1]. To simplify some of formalization, we also define idling transitions that do not change the state, which we write as $\sigma \xrightarrow{\varepsilon} \sigma$.

Execution. An *execution* of an ACCESS NET consists of a finite sequence of states $\sigma_0, \sigma_1, \ldots, \sigma_n$ wherein each state σ_{i+1} is obtained from the previous state σ_i by a legal state transition as described above. For example, we write a sample execution as follows: $\sigma_0 \xrightarrow{t_1} \sigma_1 \xrightarrow{\varepsilon} \cdots \xrightarrow{\text{tick}} \sigma_{n-1} \xrightarrow{t_2} \sigma_n$. We consider finite sequences of states since we are interested in executions in which time remains within $[\tau_{\mathsf{min}}, \tau_{\mathsf{max}}]$. However, temporal logics are interpreted over infinite state sequences (or trees) [5]. We extend our finite sequences to infinite ones by adding infinitely many *idling transitions*.

Conservation. Since tokens represent people, there is a physical constraint that all state transitions $\sigma \longrightarrow \sigma'$ conserve the number and type of tokens. We enforce this by requiring that all transitions t are *conservative*. Conservative transitions is not an inherent limitation to our approach but rather a check for more faithful models.

Definition 4 (Conservative Transitions). *A transition t is* conservative *iff for every access type, the sum of tokens of that type on incoming edges to t is equal to the sum of that type on outgoing edges, that is, for all $s \in S$,*

$$\sum_{f \in \mathsf{in}(t)} W(f, s) = \sum_{f \in \mathsf{out}(t)} W(f, s) \ .$$

If all transitions are conservative, then any execution is also conservative capturing the desired physical constraint (see our companion technical report [10] for a proof).

For reference, we gather all of the pieces of an ACCESS NET as described above in our companion technical report [10]. There are related Petri net models, e.g., with typed tokens [16] and with predicates on transitions [11]. Here, we have incorporated the aspects that are critically necessary to capture access control policies.

4 Verification of Access Properties

In this section, we consider verifying properties of ACCESS NETS. Our primary goal is to check whether tokens of a certain type can be present in a certain room in a certain time range; for example, a property of interest could say, "There is never a visitor in the archive before 9:00 a.m. or after 5:30 p.m." This restricted class of reachability properties enables us to perform aggressive state-space reduction. It is possible to extend our reductions to verify ACTL* properties, following Clarke et al. [6].

Given an ACCESS NET A with a place p and a token type s, we say p is *token reachable* for s at time τ if and only if $\sigma_0 \longrightarrow^* \sigma_n$ where $\rightarrow^*$ is the transitive closure of the transition relation, state σ_0 is the initial state, and if $\sigma_n = (m_n, \tau_n)$ then $m_n(p, s) > 0$ and $\tau_n = \tau$. As expected, we can verify such properties using model checking.

The state space of an ACCESS NET blows up quickly as we increase the number of places, transitions, and token types, as we see in our case study (Sect. 5). Fortunately, there are several natural reductions that can be performed that respect the token reachability property of interest. Our reductions generate a new ACCESS NET that *abstracts* the original in the sense that it is sound with respect to token reachability. Stated more precisely, let A' be a reduced ACCESS NET of A, and let π be the function mapping each place of A to its corresponding place in A'. Then, the reduction is *sound* with respect to token reachability if whenever $\pi(p)$ is not reachable for s at time τ in A', then p is not reachable for s at time τ in A. In other words, reduction preserves safety. Furthermore, two of our three reductions, namely the unlocked doors and redundant transition reductions, are complete with respect to token reachability.

The two most interesting reductions use the following procedure (cf. Clarke et al. [6]): (1) we define an equivalence relation $\overset{\text{pl}}{\sim}$ over places; (2) we define a new ACCESS NET A' as the quotient of A with respect to $\overset{\text{pl}}{\sim}$.

Definition 5 (Access Net Reduction). *Let A be an* ACCESS NET *and let $\overset{\text{pl}}{\sim}$ be an equivalence relation over places. This equivalence relation induces the following equivalence relation over arcs:*

$$\begin{aligned} f_1 \overset{\text{ar}}{\sim} f_2 \quad \textit{iff} \quad & p_1 \overset{\text{pl}}{\sim} p_2 \textit{ and } t_1 = t_2, \textit{ and} \\ & \textit{either } f_1 = (p_1, t_1) \textit{ and } f_2 = (p_2, t_2) \\ & \quad \textit{or } f_1 = (t_1, p_1) \textit{ and } f_2 = (t_2, p_2) \end{aligned}$$

Let π map a place or arc in $A = (P, T, \ldots)$ to its respective equivalence class (under $\overset{\text{pl}}{\sim}$ or $\overset{\text{ar}}{\sim}$). We also write π^{-1} for the pre-image of this mapping. Then, $A' = (P', T', \ldots)$ is a reduced ACCESS NET *under equivalence relation $\overset{\text{pl}}{\sim}$:*

$$P' = \pi(P) \qquad F' = \pi(F)$$
$$T' = T \qquad M' = M \qquad S' = S$$
$$\tau'_0 = \tau_0 \qquad \tau'_{\mathsf{min}} = \tau_{\mathsf{min}} \qquad \tau'_{\mathsf{max}} = \tau_{\mathsf{max}} \qquad H' = H$$
$$W' \colon (f', s) \mapsto \sum_{f \in \pi^{-1}(\{f'\})} W(f, s) \qquad m'_0 \colon (p', s) \mapsto \sum_{p \in \pi^{-1}(\{p'\})} m_0(p, s)$$

where $\pi(P)$ is the image of P under π and $\pi(F)$ is the image of F under π. That is, we map all places and arcs to their equivalence classes (first line); transitions, mandatory transitions, token types, time constraints and the clock stay the same; weights on arcs and the initial marking are combined by summing the number of tokens of each type.

It remains to be shown that A' as defined above is actually an ACCESS NET. In particular, the main property that must be checked is that conservation is preserved, which is shown in our companion technical report [10]. We now define several reductions on ACCESS NETS that use this idea of defining equivalence relations over places.

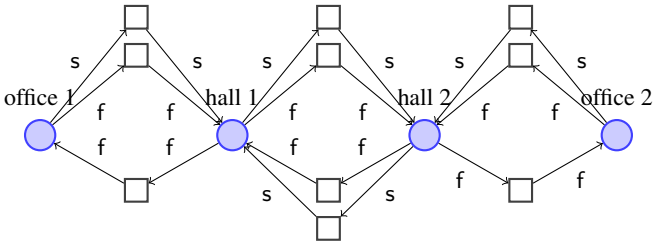
Unlocked Doors Reduction. If the only barrier between two rooms is an unlocked door, then for the purpose of checking reachability, the two rooms can be merged into a single room.

Definition 6 (Equivalent up to Unlocked Doors). *A room p_2 can be reached through one unlocked door from a room $p_1 \neq p_2$, written* $\mathsf{unlocked}(p_1, p_2)$*, if and only if for every security role s, there is some transition t such that*

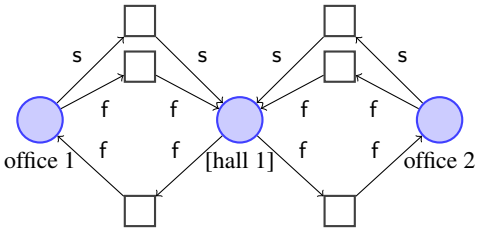
1. *We have $H(t) = [\tau_{\mathsf{min}}, \tau_{\mathsf{max}}]$, that is, the transition is enabled at all times.*
2. *We have* $\mathsf{pred}(t) = \{p_1\}$ *and* $\mathsf{succ}(t) = \{p_2\}$ *where* pred *and* succ *are the functions mapping a place to its sets of predecessor and successor places, respectively. In other words, t is a transition from p_1 to p_2 and does not take tokens from or send tokens to any places other than p_1 and p_2.*

Two rooms p_1 and p_2 are equivalent up to unlocked doors in one-step*, written $p_1 \overset{\text{pl}}{\sim}_1 p_2$, if and only if* $\mathsf{unlocked}(p_1, p_2)$ *and* $\mathsf{unlocked}(p_2, p_1)$*. The equivalence relation for unlocked doors is simply the reflexive-transitive closure of $\overset{\text{pl}}{\sim}_1$.*

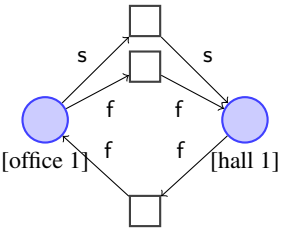
Figure 4(a) shows a simplified ACCESS NET of the office building (ECOT) used in our case study (see Sect. 5) with two token types `s` and `f` (to represent students and faculty, respectively). Figure 4(b) shows the result of applying the unlocked doors reduction to Fig. 4(a). We see that the two places hall 1 and hall 2 have been merged into a place [hall 1], as the two places allow free passage of both `s` and `f` in both directions. After the unlocked doors reduction, the reduced model technically would have unnecessary self-loop transitions between each new representative and itself. These transitions can be deleted from the model.



(a) Simplified ACCESS NET before any reductions.



(b) After the unlocked doors reduction.



(c) After equivalent rooms and redundant transitions reductions.

Fig. 4. Applying reductions to a simplified ACCESS NET for an office building.

Redundant Transitions Reduction. If two transitions represent identical access rules and move tokens from the same sources to the same destinations, then one of the two transitions can be deleted. This reduction does not follow the pattern of defining an equivalence relation on places for the sake of merging places. Instead if two transitions are equivalent according to the following definition, then one of the transitions can be arbitrarily deleted.

Definition 7 (Equivalent up to Redundant Transitions). *Two transitions t_1 and t_2 are* equivalent up to redundant transitions*, written $\overset{\text{tr}}{\sim}$, iff:*

1. *$H(t_1) = H(t_2)$, that is, the transitions are enabled at exactly the same times.*
2. *There exists a bijective mapping $\mu : \mathsf{in}(t_1) \to \mathsf{in}(t_2)$ such that for every place p_1 where $(p_1, t_1) \in \mathsf{in}(t_1)$, then whenever $(p_2, t_2) = \mu(p_1, t_1)$, we have:*

$$p_1 = p_2 \qquad \textit{and} \qquad W((p_1, t_1), s) = W((p_2, t_2), s) \textit{ for every } s \in S \ .$$

3. *There exists a bijective mapping $\nu : \mathsf{out}(t_1) \to \mathsf{out}(t_2)$ such that for every place p_1 where $(t_1, p_1) \in \mathsf{out}(t_1)$, then whenever $(t_2, p_2) = \nu(t_1, p_1)$, we have:*

$$p_1 = p_2 \qquad \textit{and} \qquad W((t_1, p_1), s) = W((t_2, p_2), s) \textit{ for every } s \in S \ .$$

This definition says that two transitions are equivalent if they are enabled at exactly the same times and if their incoming and outgoing edges can be put into a bijective correspondence of equivalent edges. It is straightforward to show that $\overset{\text{tr}}{\sim}$ is an equivalence relation. For each equivalence class of transitions, all of the transitions in that class can be deleted except for one arbitrary representative.

Equivalent Rooms Reduction. If two rooms are equivalent in the sense that they are only reachable from the same rooms according to the same access control rules, then the two rooms can be merged into a single room.

Definition 8 (Equivalent Rooms). *First, we define two transitions as being* equivalent up to $q_1 = q_2$*, written* $\overset{q_1=q_2}{\sim}$ *in the same way as* $\overset{\text{tr}}{\sim}$ *from the redundant transitions reduction with one change. Instead of requiring of* μ *and* ν *that* $p_1 = p_2$*, we require only that*

$$p_1 = p_2 \qquad \textit{or} \qquad p_1 = q_1 \textit{ and } p_2 = q_2 \qquad \textit{or} \qquad p_1 = q_2 \textit{ and } p_2 = q_1$$

That is, the transitions are redundant under an assumption that $q_1 = q_2$*. Then two rooms* q_1 *and* q_2 *are* equivalent rooms*, denoted* $q_1 \overset{\text{pl}}{\sim} q_2$ *if and only if*

1. *There exists a bijective mapping* $\mu : \mathsf{pred}(q_1) \rightarrow \mathsf{pred}(q_2)$ *such that for every transition* $t \in \mathsf{pred}(q_1)$*, we have* $t \overset{q_1=q_2}{\sim} \mu(t)$*.*
2. *There exists a bijective mapping* $\nu : \mathsf{succ}(q_1) \rightarrow \mathsf{succ}(q_2)$ *such that for every transition* $t \in \mathsf{succ}(q_1)$*, we have* $t \overset{q_1=q_2}{\sim} \nu(t)$*.*

It is straightforward to show that $\overset{\text{pl}}{\sim}$ is an equivalence relation. Figure 4(c) shows the result of applying the equivalent rooms and redundant doors reductions to Fig. 4(b). We see that office 1 and office 2 have been merged into a single place [office 1]. These reductions correspond naturally to our intuition, as from the perspective of token reachability, all of the offices and all of the halls look the same as long as they are connected to each other though "unlocked areas."

All three reductions, unlocked doors, redundant transitions, and equivalent rooms, are sound with respect to token reachability. Furthermore, unlocked doors and redundant transitions are complete with respect to token reachability (though equivalent rooms is not). Proofs of these facts are given in our companion technical report [10]. At a high-level, the reductions merge "equivalent" places that satisfy the same set of properties, in order to construct an abstraction. This abstraction is similar in ways to canonical abstraction in the TVLA program analysis framework [22].

Untiming. Regarding tick transitions, the definition for state transitions allows arbitrary time steps and describes an infinite state space. However, only the the initial time and the boundaries of time intervals referenced by time-dependent transitions need to be considered during verification. To this end, we apply a standard untiming construction as described in Alur and Dill [1]. The untiming construction is especially simplified in the case of ACCESS NETS since the model has a single timer.

5 Case Study: Office Security

To validate the feasibility of our approach, we modeled a part of the Engineering Center Office Tower (ECOT) at the University of Colorado, Boulder and a set of synthetic access control rules. Furthermore, we have completely modeled an actual, large office building with multiple floors, occupied by many businesses using a real access control

Table 1. Dependence of explicit-state model checking using Spin on the size of the building for verifying a valid property. In each test case run, we show the size of the ACCESS NET (number of rooms, transitions, and persons) along with the number states observed by Spin, the total memory used by Spin, and the total time for the model checking to run.

Model	Rooms	Transitions	Persons	States	Memory (MB)	Time (s)
ECOT 8	17	38	3	1603	13.9	0.1
ECOT 7,8	50	120	3	20429	22.7	2.7
ECOT 6,7,8	92	222	3	93578	103.6	22.8

policy. With these models, we applied explicit-state model checking using Spin [15] and bounded model checking [4] using our implementation based on the Yices SMT solver [8] (other verification techniques could apply). With this study, we are interested in how the building and access control policy is encoded as an ACCESS NET model and how feasible is model checking. We also look at how much the state space can be reduced using the techniques from Sect. 4.

To create our ACCESS NET model of ECOT, we examined CAD drawings of the sixth, seventh, and eighth floors of the building (which is where the Computer Science Department is located). The access control policy involved three access types, student, faculty, and maintenance and consisted of the following rules:

1. Any faculty can enter any office. Anybody in an office can exit it.
2. Any maintenance can enter a mechanical room or janitorial room. Anybody in one of these rooms can exit it.
3. Any student can only enter a conference room accompanied by a faculty and only between 9:00 a.m. and 5:00 p.m. Conference rooms can be exited freely.

As perhaps expected, the state space grows quickly even for our relatively small models by either increasing the number of places and transitions or the number of tokens. Fortunately, the topology and access policies that we work with are amenable to reduction, which apply regardless of model checking technique. We first consider verification of a valid property using Spin on unreduced models and look at the growth in verification resources as a function of model size. Then, we look at the cost of discovering property violations. We consider not only explicit-state model checking but also bounded model checking, which is insensitive to number of tokens. Finally, we look at the effectiveness of reduction.

Verifying Valid Properties. Table 1 shows the relationship between the resources required for explicit-state model checking and number of rooms and transitions in an ACCESS NET, while Table 2 considers the dependence on number of persons. All of our tests were executed on a Linux server with 32 GB memory and sixteen 2.93GHz Intel Xeon X7350 CPUs. In the test runs in these two tables, we checked that "a student can never be in a particular office (826) and a faculty member can never be in a specific mechanical room (805A)," which is valid in these models. There were no uses of time in any of these models (i.e., we did not use rule 3 regarding the conference rooms here).

In Table 1, we see that Spin works with reasonable memory and time constraints, but the state space blows up quite quickly as we add additional rooms (by successively

Table 2. Dependence of explicit-state model checking using Spin on the number of persons in the model for verifying a valid property. We started with one person of each type and then successively added one faculty at a time.

Model	Rooms	Transitions	Persons	States	Memory (MB)	Time (s)
ECOT 8	17	38	3	1603	13.9	0.1
ECOT 8	17	38	4	12523	15.8	0.7
ECOT 8	17	38	5	70763	26.1	4.3
ECOT 8	17	38	6	318283	89.1	22.4
ECOT 8	17	38	7	1209355	405.3	97.9

adding the rooms on the 7th and 6th floors). In these tests, we started one token of each type in a public room on the 8th floor.

For increasing number of persons, we see the number of states considered by Spin also grows rapidly, as does the increase in memory and time consumption, as shown in Table 2. In these tests, we started with one token of each type and then successively added one token of type faculty at a time. We chose the faculty type, as it leads to the largest state space.

Discovering Property Violations. Both of the previous examples checked properties that could not be violated (and thus required an exhaustive exploration of the state space). Here, we consider violated properties. First, we consider the same set of ACCESS NET models from Table 1 that do not use timed transitions. The violated property was "a faculty member cannot be in a particular office on the 8th floor (826)." The test runs are shown in the top half of Table 3.

We then looked at a set of timed models that adds the conference room rule (rule 3), that is, that a student can only enter a conference room with a faculty member between 9:00 a.m. and 5:00 p.m. In this case, the initial state was a student and a faculty on the 7th floor at 9:00 a.m. The violated property was "a student cannot be in the conference room (831) at 6:00 p.m." This property can be violated by the faculty letting the student into the conference room between 9:00 a.m. and 5:00 p.m. and then the student remaining in the room after 5:00 p.m. until 6:00 p.m.

Table 2 shows rapid explosion in the number of states considered by Spin as we increase the number of tokens. A potential advantage of bounded model checking is its insensitivity to number of tokens, and thus, we applied it to the property violation cases of Table 3. Table 4 presents these results. We confirm that the Yices-based bounded model checker finds the same witnesses as Spin in a reasonable amount of time and space. Note that our BMC implementation is currently a prototype.

Reductions. The redundant structure of ECOT is particularly well-suited to the reductions described in Sect. 4. Even after adding in the 5th floor, ECOT reduces to just four rooms! In the reduced model, there is one public room, one representative office, one representative maintenance room, and one representative conference room. Table 5 shows the results from the reductions; the memory and time measurements show the cost of computing the reduced model. In all cases, after applying all reductions, the models are so small that the model checking times are negligible (and thus not shown

Table 3. Discovering a property violation using Spin in breadth-first search mode. We consider the set of ACCESS NET models without timed transitions from Table 1 and a new set with timed transitions.

Model	Rooms	Transitions	Persons	States	Memory (MB)	Depth	Time (s)
Without Timed Transitions							
ECOT 8	17	38	3	556	4.8	2	0.0
ECOT 7,8	50	120	3	2961	10.5	5	0.3
ECOT 6,7,8	92	222	3	10730	46.0	6	2.4
With Timed Transitions							
ECOT 8	17	42	3	6897	7.1	6	0.1
ECOT 7,8	50	124	3	64323	68.5	9	6.0

Table 4. Discovering a property violation using the Yices-based BMC (untimed models). Note that BMC does not require a bound on the number of persons.

Model	Memory (MB)	Time (s)	Depth
ECOT 8	6.4	0.2	2
ECOT 7,8	38.4	5.7	5
ECOT 6,7,8	104.9	21.7	6

in the table). The rows labeled 'some reductions' show the effect of performing only the unlocked doors and redundant transitions reductions (Definitions 6 and 7). The rows labeled 'all reductions' add the equivalent rooms reduction (Definition 8), which is all reductions described in Sect. 4. In our model, all faculty can access all offices, but in a slight variant, we may have a unique access policy for each office. In this case, the equivalent rooms reduction would have no effect giving reduced models analogous to the cases with 'some reductions.'

Real-World Example. We also obtained the complete access control specification for an actual four-story, multi-tenant office building. The building houses roughly 200 employees during working hours. Our model of the building had about 200 rooms and 230 doors. The operators of the building can assign up to 24 different access types. Due to the exponential dependence of our model on the number of people in the model, we could not simulate all access types at once. We selected two access types that were more interesting and ran a simple slicing reduction (not described in Sect. 4) that removes transitions for the excluded access types.

Without reductions, this model is too large for the explicit-state model checker, but the reductions are very effective (see Table 6). After reductions, we can prove safety properties very efficiently. Note that the restrictions in the number of persons does not apply to BMC that was run for an unbounded number of persons. This makes the BMC approach especially appealing because the number of people in the model does not directly affect the encoding size. The BMC implementation ran for an hour on the full

Table 5. Size of reduced models and resource requirements to calculate the reduction. The marking (<) indicates something below the granularity of our measurements, while (*) indicates a case where we fail to run Spin possibly because of the model size.

					Model Reduction	
Model	Rooms	Transitions	Persons	States	Memory (MB)	Time (s)
ECOT 8						
no reductions	17	38	3	1603	N/A	N/A
some reductions	6	16	3	37	<	<
all reductions	4	6	3	20	<	<
ECOT 7, 8						
no reductions	50	120	3	20429	N/A	N/A
some reductions	23	62	3	167	<	0.1
all reductions	4	6	3	20	4.1	0.6
ECOT 6, 7, 8						
no reductions	92	222	3	93578	N/A	N/A
some reductions	47	124	3	423	5.1	0.2
all reductions	4	6	3	20	5.1	4.5
ECOT 5, 6, 7, 8						
no reductions	138	336	3	*	N/A	N/A
some reductions	74	194	3	859	5.1	0.4
all reductions	4	6	3	20	5.1	18.1

Table 6. Results of explicit-state model checking for real office building.

Model	Rooms	Transitions	Persons	States	Memory (MB)	Time (s)
before reductions	207	460	2	N/A	N/A	N/A
after reductions	10	15	2	25	13.9	1.7

model without reduction, searching up to depth 7, but was unable to find a violation. The running time on the reduced model was significantly smaller (30s) for a depth 30.

6 Related Work

Sampemane et al.present a specification formalism for role-based access control to physical spaces that allow novel uses of physical spaces, while ensuring that resources in these spaces are not misused [23]. Similarly, Bauer et al.present a framework for modeling and reasoning about personnel credentials and their delegation for physical as well as cyber access control using theorem proving [3]. The previously cited works present formalizations that support the addition, deletion, and modification of access control policies. Our work is complementary: we focus on modeling the physical topology of the building and reasoning about its interplay with access control mechanisms.

Dynamical models of buildings have been investigated both at the *macroscopic levels*, wherein, pedestrian flows are often modeled as continuous, without distinguishing the behavior of each individual pedestrian [14] and at *microscopic levels* with an agent-based model of individual actions. Applications of these simulations have included techniques to predict the time to evacuate large and complex buildings [18,13,24,21]. These models inevitably use a graph-based representation to capture the building topology. Our work offers a systematic model that also takes into account the different access levels and the complex software-controlled access policies that are virtually standard in modern buildings.

Model checking has also been applied in the past to check access control policy for computer network systems. The model proposed here is similar to the role-based access control (RBAC) model used for mediating access to electronic resources in an organization [9]. The verification of access control policies for organizational systems has been considered in the past. For instance, Jha et al. [17] present a formalization of various RBAC models and characterize the computational complexity of some analysis problems. Guelev et al.present a model-checking approach for verifying both the permissivity as well as the security of access control policies [12].

Our work on physical spaces bears many similarities to role-based access control policies. For instance, our model assumes that permissions are provided based on certain well-defined organizational roles, which can be finitely many and well-known *a priori*. However, the verification problem is inherently different. Unlike network topologies, buildings have a non-trivial spatial layout, whose modeling at the appropriate level of detail is critical. Furthermore, buildings tend to be larger with more rooms, doors, passageways with a rich variety of access enforcement mechanisms. Building access control rules vary with time unlike network access control rules. Finally, the need for mandatory transitions is also quite unique. Nevertheless, as witnessed by the success of our abstraction-based approaches, buildings also present large amount of regularity that can be exploited through simple reduction schemes to significantly reduce the complexity of property verification.

Our work makes use of a translation to existing model checking tools including Spin for explicit state model checking [5,15], as well as a bounded model checker [4] implemented using the SAT-modulo theory solver Yices [20,8]. Other fast SMT solvers include solvers such as Z3 [7,2].

7 Conclusion

Although we have focused on reachability properties, we can consider ACCESS NETS that model and verify other aspects of physical spaces. For example, other potential applications include checking for *detectability* of violations (e.g., by adding observability to the semantics) or modeling *evacuation* plans for buildings. In summary, we have presented a formal model, ACCESS NETS, for analyzing access control policies for physical spaces. The model can express many aspects that are relevant such as physical topology, role-based access policies, and time-dependent access rules. Formal verification techniques can be used on these models, thereby making computer-aided validation of access control policies possible. Furthermore, we have demonstrated that although the

state-space does explode, domain-specific state-space reduction techniques are quite effective in reducing the complexity of the verification problem.

Acknowledgments. We thank the anonymous reviewers for their helpful comments.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2), 1994.
2. C. Barrett, M. Deters, A. Oliveras, and A. Stump. Design and results of the third annual satisfiability modulo theories competition (SMT-Comp 2007). *International Journal on Artificial Intelligence Tools*, 17(4), 2008.
3. L. Bauer, S. Garriss, and M. K. Reiter. Efficient proving for practical distributed access-control systems. In *Computer Security (ESORICS)*, 2007.
4. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 1999.
5. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. 1999.
6. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5), 1994.
7. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008.
8. B. Dutertre and L. de Moura. The YICES SMT solver. `http://yices.csl.sri.com/tool-paper.pdf`.
9. D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-based Access Control*. 2003.
10. R. Frohardt, B.-Y. E. Chang, and S. Sankaranarayanan. Access Nets: Modeling access to physical spaces (extended version). Technical Report CU-CS-1076-10, Department of Computer Science, University of Colorado, Boulder, 2010.
11. H. J. Genrich and K. Lautenbach. System modelling with high-level Petri nets. *Theor. Comput. Sci.*, 13(1), 1981.
12. D. P. Guelev, M. Ryan, and P.-Y. Schobbens. Model-checking access control policies. In *Information Security (ISC)*, 2004.
13. D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803), 2000.
14. L. Henderson. The statistics of crowd fluids. *Nature*, 229, 1971.
15. G. Holzmann. *The SPIN Model Checker*. 2003.
16. K. Jensen. Coloured Petri nets and the invariant-method. *Theor. Comput. Sci.*, 14(3), 1981.
17. S. Jha, N. Li, M. V. Tripunitara, Q. Wang, and W. H. Winsborough. Towards formal verification of role-based access control policies. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 5(4), 2008.
18. G. Lovas. Modeling and simulation of pedestrian traffic flow. *Transportation Research B*, 28(6), 1994.
19. T. Murata. Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4), 1989.
20. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract DPLL procedure to DPLL(T). *J. ACM*, 53(6), 2006.
21. N. Pelechano and A. Malkawi. Evacuation simulation models: Challenges in modeling high rise building evacuation with cellular automata approaches. *Automation in Construction*, 17(4), 2008.
22. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.*, 24(3), 2002.
23. G. Sampemane, P. Naldurg, and R. H. Campbell. Access control for active spaces. In *Computer Security Applications (ACSAC)*, 2002.
24. T. Shen. ESM: A building evacuation simulation model. *Building and Environment*, 40(5), 2005.