ALPS: An Action Language for Policy Specification and Automated Safety Analysis

Silvio Ranise Riccardo Traverso

Security & Trust Unit
Fondazione Bruno Kessler
Trento, Italy

Trento, October 21th, 2014

Many Models for Access Control

- Several models, languages, and enforcement mechanisms have been proposed for different scenarios.
    - Discretionary
    - Mandatory
    - Role-Based
    - Attribute-Based
- We can choose the most suited model or even combine multiple ones, depending on the type of authorization conditions needed.
- There are many types of access conditions we might want to express.

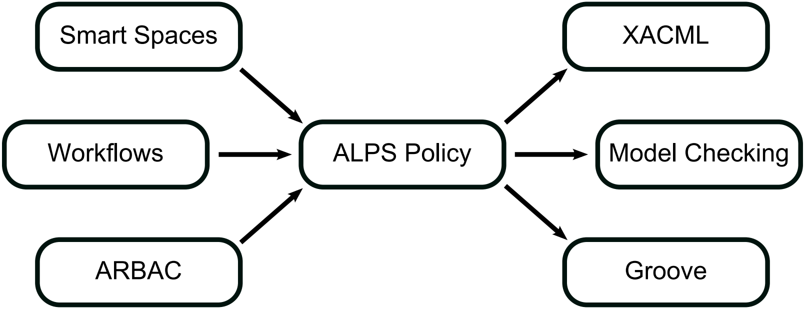
The Problem

- Too expressive models have to be restricted in order to make safety decidable, making it more difficult to use them for realistic scenarios.
- Each model has its own characteristics and algorithms for solving safety.
- Sometimes it is possible to translate a policy into a different model, but the very heterogeneous landscape of models makes it a difficult task.

Action Language for Policy Specification (ALPS)

- It preserves decidability of safety verification.
- It can express many models and policies with contextual constraints.
- Intermediate language:
  - focus on re-using existing, well-engineered verification techniques by means of translations from/to other models;
  - still, there is the possibility to develop new specific decidability results for safety.

ALPS: Intermediate Language



Smart Spaces

Workflows

ARBAC

ALPS Policy

XACML

Model Checking

Groove

ALPS: Features

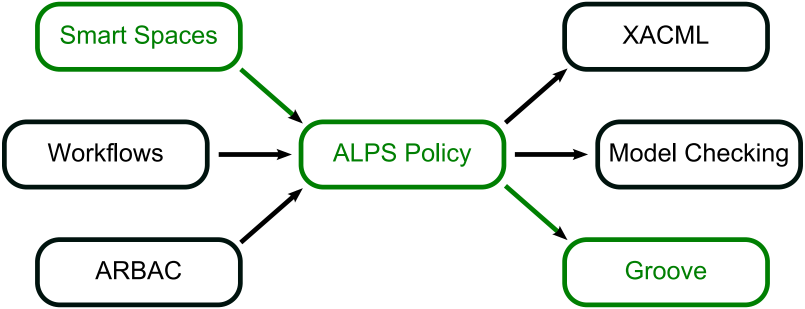
Inspired by the STRIPS action language.

- Pre-conditions: positive/negative guards.
- Post-conditions: add list, delete list.
- Priorities (normal vs mandatory).
- . . .

ALPS: Features (cont'd)

- . . .
- Type predicates with subtyping.
- Immutable predicates.
- Notion of time (modulo some T).

Example: Smart Spaces



Smart Spaces

Workflows

ARBAC

ALPS Policy

XACML

Model Checking

Groove

Doctor Cabinet Example

We model the policy for a cabinet of a group of doctors.

- A person is either a doctor or a patient.

- Several waiting rooms, freely accessible, connected to offices.
- Each office is owned by a doctor.

- Doctors arrive between 7am and 9am.
- Patients enter offices one at a time, between 8am and 7pm.
- Doctors leave anytime, if they are not busy with a patient.

Declarations

```
maxtime 24;

type Space, Office(Space), WaitingRoom(Space);
type Person, Doctor(Person), Patient(Person);

immutable predicate Owner(Doctor, Office);
immutable predicate Door(Space, Space);

predicate Busy(Doctor), In(Person, Space);
```

docArrives and docLeaves

```
mandatory action docArrives(Doctor d, WaitingRoom w,
    Office o)
{
  7 <= time and time <= 9,
  Owner(d, o), Door(w, o),
  -In(d, w), +In(d, o)
}

action docLeaves(Doctor d, Office o, WaitingRoom w)
{
  not Busy(d), Door(o, w),
  -In(d, o), +In(d, w)
}
```

patEnters and patLeaves

```
action patEnters(Patient p, Doctor d, Office o,
    WaitingRoom w)
{
  8 <= time and time < 19,
  In(d, o), not Busy(d), Door(w, o),
  -In(p, w), +In(p, o), +Busy(d)
}

action patLeaves(Patient p, Doctor d, Office o,
    WaitingRoom w)
{
  In(d, o), Door(o, w),
  -In(p, o), -Busy(d), +In(p, w)
}
```

waitingRoom

```
action waitingRoom(Person p, WaitingRoom w1,
    WaitingRoom w2)
{
  Door(w1, w2),
  -In(p, w1), +In(p, w2)
}
```

Initial Configurations

```
conf start1
{
  Office(o1), Office(o2), WaitingRoom(wr),
  Door(wr,o1), Door(o1,wr), Door(wr,o2), Door(o2,wr),
  Doctor(d1), Doctor(d2), Patient(p),
  Owner(d1, o1), Owner(d2, o2),
  In(p, wr), In(d1, wr), In(d2, wr)
}

conf start2 { ... }

...
```

Verification

Starting from the initial configuration *start*1:

- It is possible for a patient to be in an office outside of the opening hours (i.e. before 7 am or after 7 pm)?

```
Patient(p), Office(o), In(p, o),
(0 <= time and time < 7) or (19 < time
  and time < 24)
```

- Could a doctor leave a patient alone in an office?

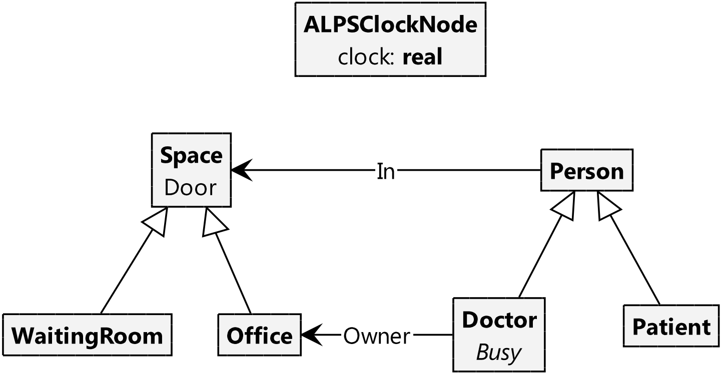
```
Doctor (d), Patient(p), Office(o), Owner(d, o),
In(p, o), not In(d, o)
```

Groove

Groove is a model checker for graph rewriting systems.

- It has a rich input language with numeric fields, typed nodes, and control programs.
- Translating ALPS policies into graph grammars for Groove provides
    - an interactive, graphical representation of the system, and
    - CTL/LTL model checking capabilities.

Type Graph



- Type and predicate declarations are encoded in type graph:
    - types become node types;
    - unary predicates become flags;
    - binary predicates become directed edges;
- Time is represented via a real-valued field in a special node

Rules: Time Handling

Time Transition:

ALPSClockNode

clock := self.clock + 0.5

self.clock < 24.0

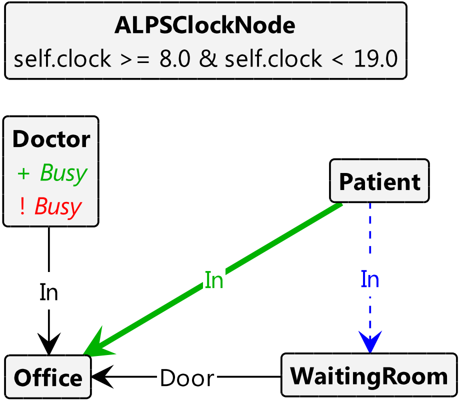
Time Reset:

ALPSClockNode

clock := 0.00

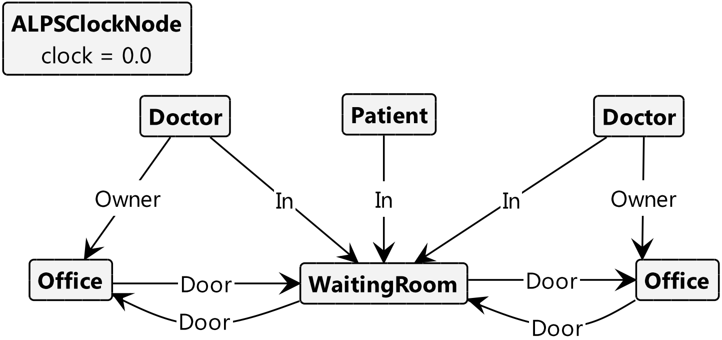
self.clock >= 24.0

Rule: patEnters

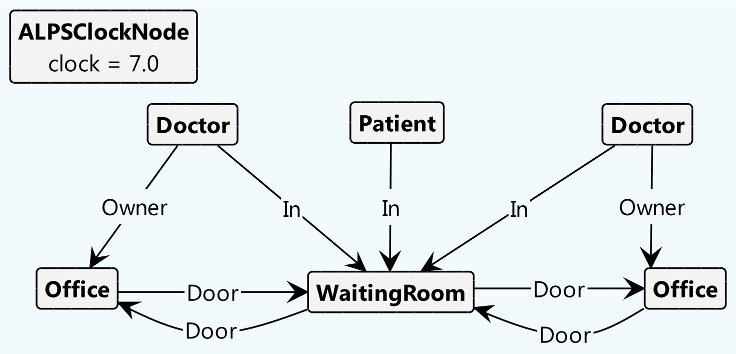


```
action patEnters(Patient p, Doctor d, Office o,
    WaitingRoom w) {
  8 <= time and time < 19, In(d, o), not Busy(d), Door(
     w, o), -In(p, w), +In(p, o), +Busy(d)
}
```

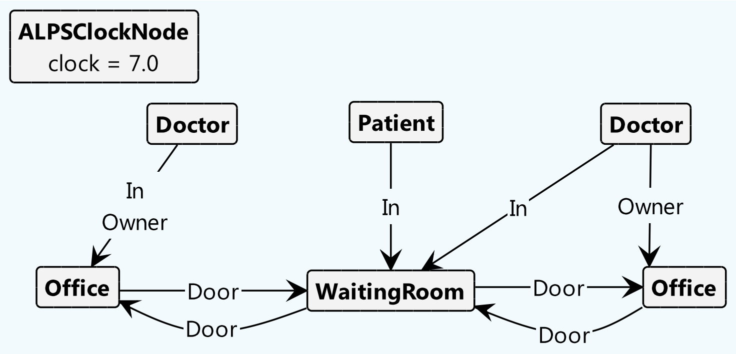
Example execution from start1



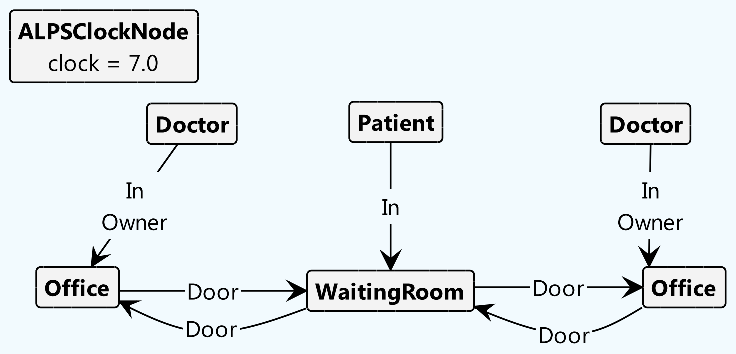
Example execution from start1



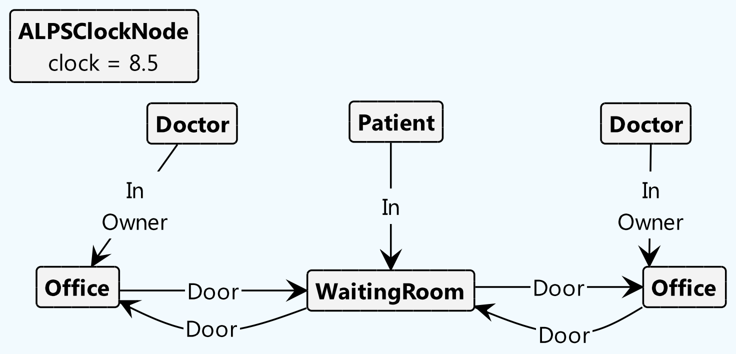
Example execution from start1



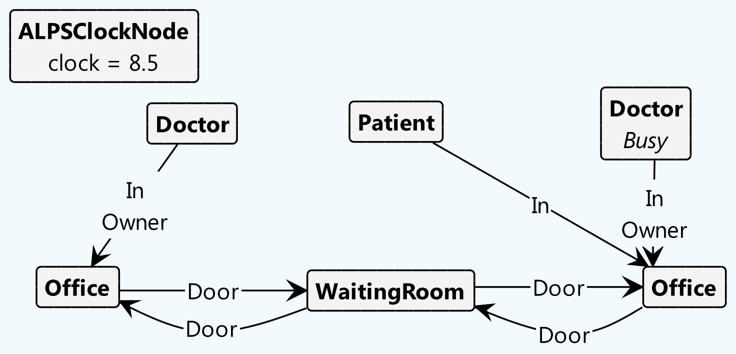
Example execution from start1



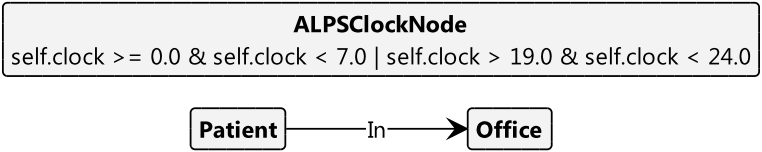
Example execution from start1



Example execution from start1



Verification



ALPSClockNode

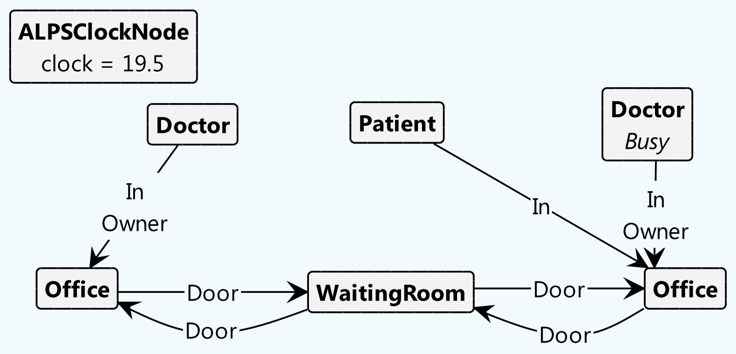
self.clock >= 0.0 & self.clock < 7.0 | self.clock > 19.0 & self.clock < 24.0

Patient In **Office**

```
Patient(p), Office(o), In(p, o),
(0 <= time and time < 7) or (19 < time and time < 24)
```

- Target rule for reachability: does it eventually become fireable?
- Groove proves the system not to be safe, by providing an example with a full execution trace.

Verification



Reachability Problem (Safety)

Given:

- An ALPS policy.
- An initial configuration θ_0.
- A goal $\mathcal{G}$, i.e. conjunction of action pre-conditions (like an action without post-conditions).

Problem:

- Is it possible to reach a configuration satisfying $\mathcal{G}$ by starting from θ_0?

Complexity Results

Without mandatory actions:

- 2-EXPSPACE in the general case;
- EXPSPACE without time;
- NEXPTIME without time and delete lists;
- EXPTIME without time, delete lists, and negative pre-conditions.

With (unbounded) creation of values:

- undecidable already with positive pre-conditions, delete lists, add lists;
- might be decidable by adding restrictions to the usage of variables.

Known Results

Particular cases of ALPS policies:

- Checking safety is PSPACE-complete for ARBAC policies.
- The workflow satisfiability problem is NP-complete.

Advantages

- The same ALPS policy can be:
  - verified for safety with existing techniques from different models;
  - enforced via a translation to XACML.
- It makes possible to provide to different verification techniques a common, more objective and easily comparable set of benchmarks in ALPS.

Future/Ongoing Work

- Implementation of XACML translation.
- Extensions to the language:
  - reflexive, symmetric, and transitive predicates;
  - goals.
- Complete characterization of the complexity for checking safety.
- Study relationships with existing models and techniques.

Thanks for the attention!