Automated Synthesis of Run-time Monitors to Enforce Authorization Policies in Business Processes

C. Bertolissi[1,4], **D.R. dos Santos**[1,2,3], S. Ranise[1]

[1]Fondazione Bruno Kessler, [2]SAP Labs France,
[3]University of Trento, [4]University of Marseille

SECENTIS

A European Industrial Doctorate on Security and Trust

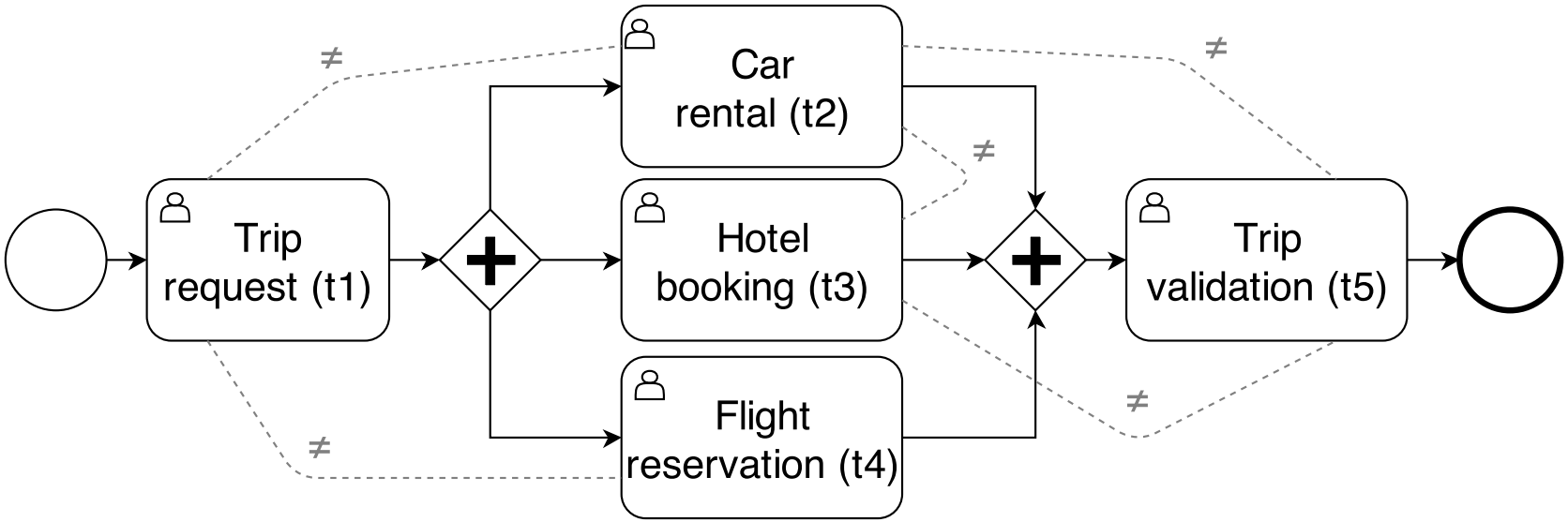
Outline

Outline

Context

- A workflow specifies a collection of tasks and the causal relationships between them
- Authorization policies specify which users can execute which tasks
- Additional constraints, such as Separation/Binding of Duty, further restrict the execution of tasks by users

Example



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

Problem

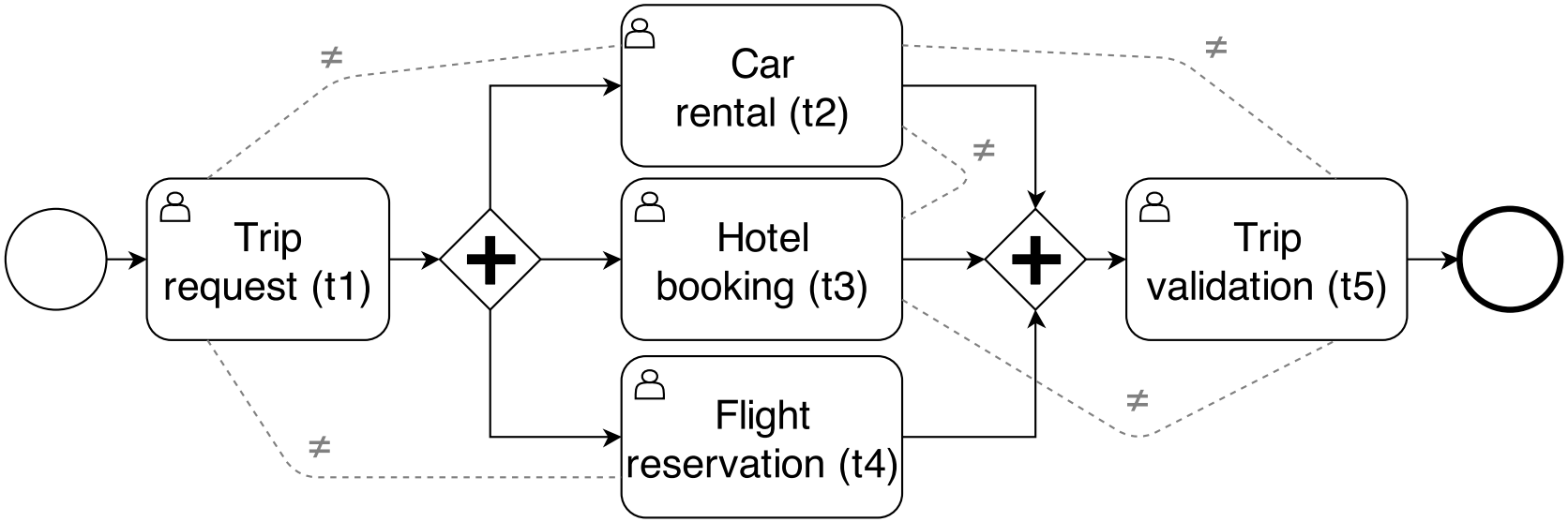
Workflow Satisfiability Problem (WSP)

Is there an assignment of users to tasks such that a workflow terminates while satisfying all authorization constraints?

Run-time WSP

Answering sequences of user requests at execution time ensuring termination with the satisfaction of authorization constraints

Run-time version allows us to divide the problem in two steps



task	roles
t1	r3
t2	r2
t3	r2
t4	r1
t5	r2

roles	users
r1	a
r2	a, b, c
r3	a, b

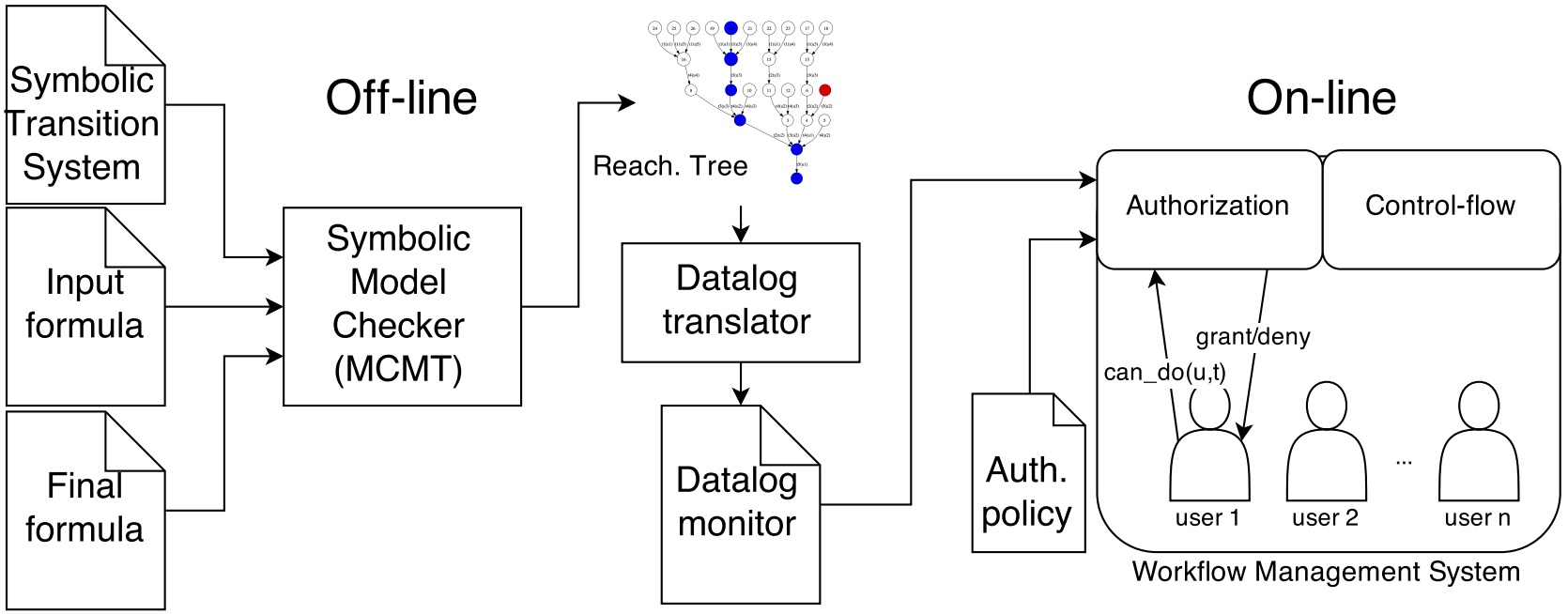
Satisfying trace: *t1(b), t2(a), t3(c), t4(a), t5(b)*

Contribution

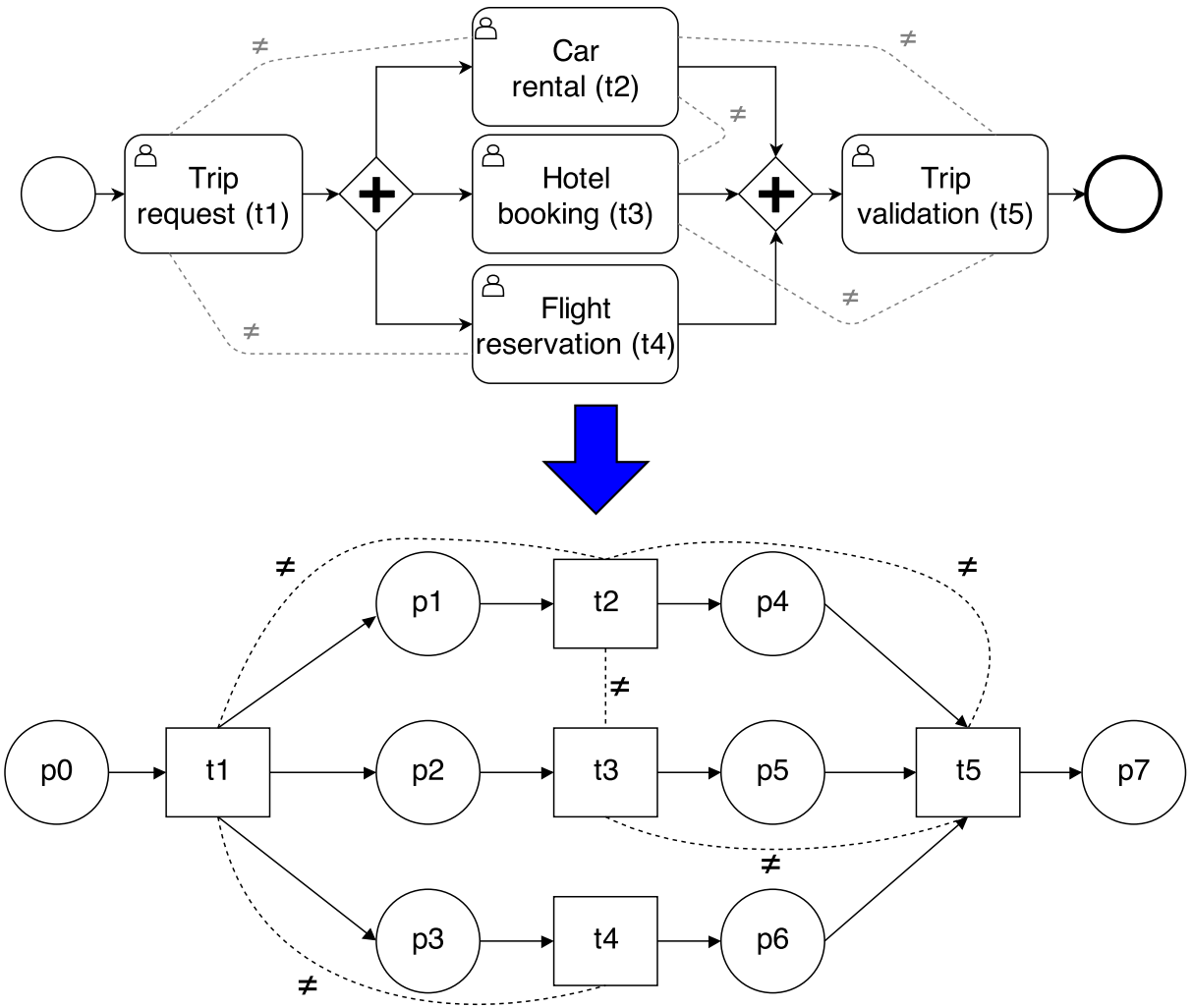
- Automated technique to synthesize run-time monitors solving the run-time WSP
- Divided in off-line (workflow+constraints) and on-line (policy) phases
- Changes in the authorization policies can be accommodated without re-running from scratch the approach
- Main advantage: symbolic representation of users

Outline

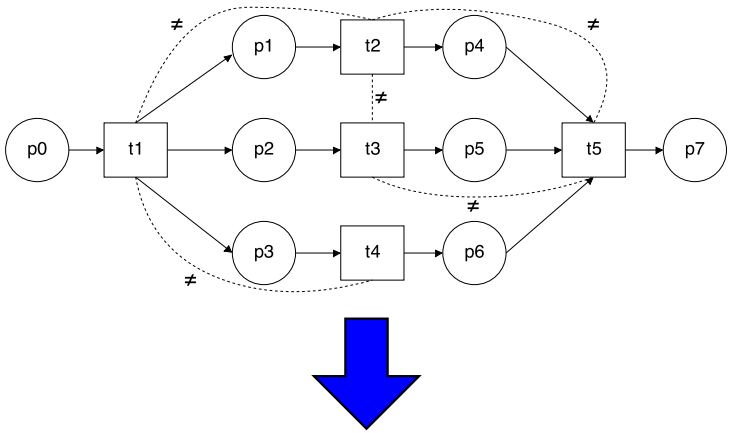
Overview


Symbolic Transition System
Input formula
Final formula
Off-line
Symbolic Model Checker (MCMT)
Reach. Tree
Datalog translator
Datalog monitor
On-line
Authorization
Control-flow
grant/deny
can_do(u,t)
Auth. policy
user 1
user 2
...
user n
Workflow Management System

BPMN to Petri Net



Petri Net to Transition System



t(u): $enabled_{CF} \wedge enabled_{Auth} \longrightarrow act_{CF} || act_{Auth}$

t2(u): $p1 \wedge \neg d_{t2} \wedge a_{t2}(u) \wedge \neg h_{t3}(u) \wedge \neg h_{t1}(u)$
$\longrightarrow p1, p4, d_{t2} := F, T, T || h_{t2}(u) := T$

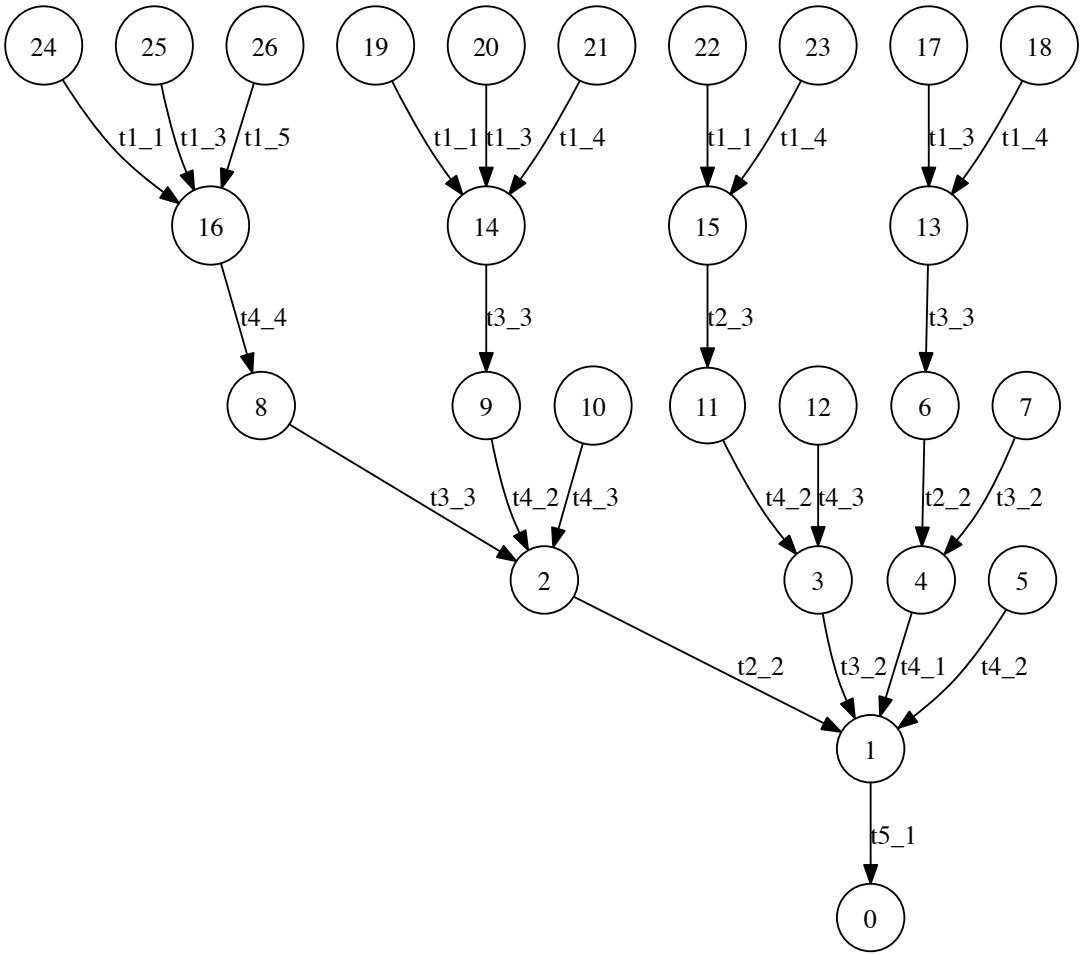
I:$p0 \wedge ... \wedge p7 \neg pi \wedge \neg d_{ti}, ..., d_{t5} \wedge \bigwedge_{i=1,...,5} \forall u. \neg h_{ti}(u)$
F:$p7 \wedge \neg p0 \wedge ... \neg p6 \wedge d_{t1} ... \wedge d_{t5}$

Computing the Reachability Graph

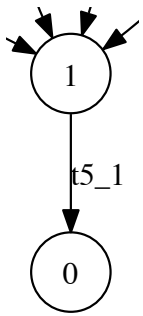
Require: $S = (V_{CF} \cup V_{Auth} \cup V_{User}, Ev_S)$ and F
Ensure: $RG = (N, \lambda, E)$

```
1:  #start with a node for the final state
2:  i ← new(); N ← {i}; E ← ∅; λ[i] ← F; TBV ← {i};
3:  while TBV ≠ ∅ do #until a fix-point is reached
4:      if subsumed(i,N,N′) then
5:          connect(N′,i); TBV ← TBV − {i};
6:      end if
7:      for all ev ∈ Ev_S do #then for every transition
8:          P ← wlp(ev, λ[i]);
9:          if P is satisfiable then #add a new node if applicable
10:             j ← new(); N ← N ∪ {j}; E ← E ∪ {(i, ev̄, j)};
11:             λ[j] ← P; TBV ← TBV ∪ {j};
12:         end if
13:     end for
14:     i ← pickOne(TBV); TBV ← TBV − {i};
15: end while
16: return (N, λ, E);
```

Reachability Graph



Reachability Graph to Datalog



$$\neg p0 \wedge \neg p1 \wedge \neg p2 \wedge \neg p3 \wedge p4 \wedge p5 \wedge p6 \wedge d_{t1} \wedge d_{t2} \wedge d_{t3} \wedge d_{t4} \wedge \neg d_{t5} \wedge (a_{t5}(u1) \wedge \neg h_{t2}(u1) \wedge \neg h_{t3}(u1))$$

$$can\_do(u1, t5) \leftarrow \neg p0 \wedge \neg p1 \wedge \neg p2 \wedge \neg p3 \wedge p4 \wedge p5 \wedge p6 \wedge d_{t1} \wedge d_{t2} \wedge d_{t3} \wedge d_{t4} \wedge \neg d_{t5} \wedge a_{t5}(u1) \wedge \neg h_{t2}(u1) \wedge \neg h_{t3}(u1)$$

Policy

RBAC Policy

$U = \{a, b, c\}$ $R = \{r_1, r_2, r_3\}$
$UA = \{(a, r1), (a, r2), (a, r3), (b, r2), (b, r3), (c, r2)\}$
$TA = \{(r_3, t1), (r_2, t2), (r_2, t3), (r_1, t4), (r_2, t5)\}$

Policy in Datalog

$ua(a, r1)$ $ua(a, r2)$ $ua(a, r3)$ $ua(b, r2)$ $ua(b, r3)$ $ua(c, r2)$
$pa(r_3, t1)$ $pa(r_2, t2)$ $pa(r_2, t3)$ $pa(r_1, t4)$ $pa(r_2, t5)$
$a_t(u) \leftarrow ua(u, r) \wedge pa(r, t)$ for each $t \in \{t1, ..., t5\}$

Example trace

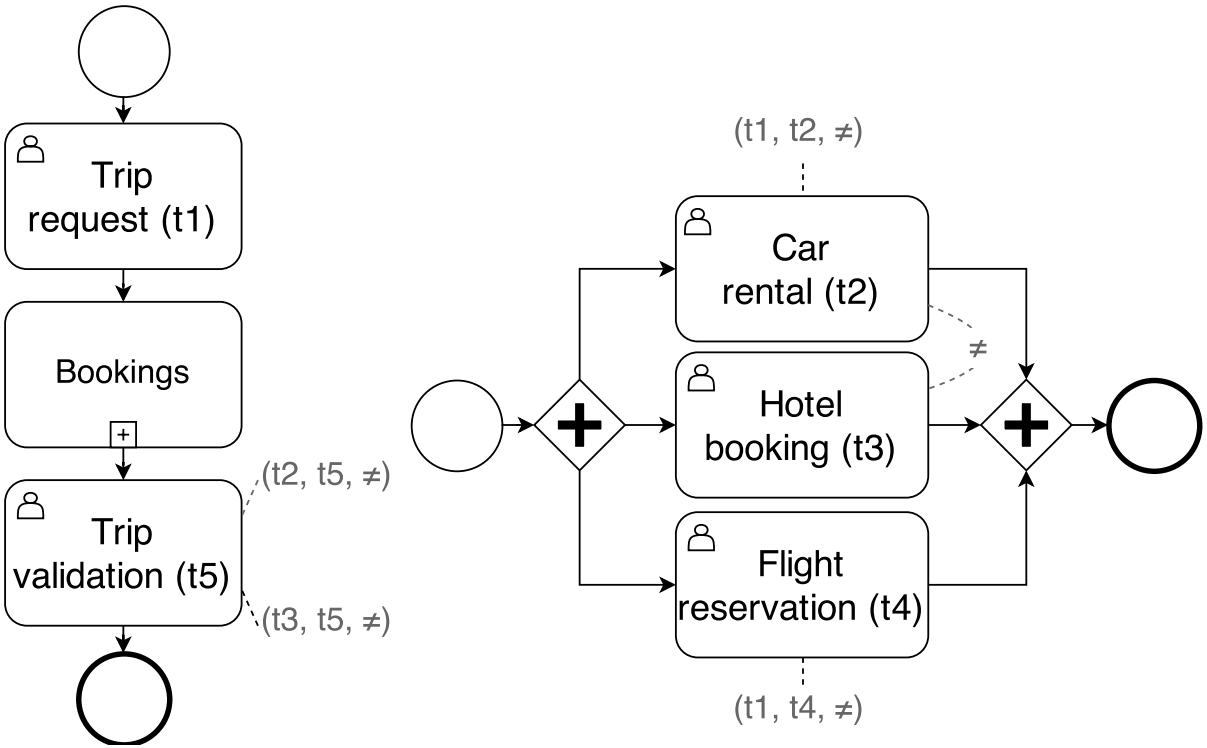
	CF	Auth					*can_do*	
#	Token in	h_{t1}	h_{t2}	h_{t3}	h_{t4}	h_{t5}	(u, t)	Resp.
0	$p0$	-	-	-	-	-	$(a, t1)$	deny
1	$p0$	-	-	-	-	-	$(b, t1)$	grant
2	$p1, p2, p3$	b	-	-	-	-	$(b, t2)$	deny
3	$p1, p2, p3$	b	-	-	-	-	$(a, t2)$	grant
4	$p4, p2, p3$	b	a	-	-	-	$(c, t3)$	grant
5	$p4, p5, p3$	b	a	c	-	-	$(a, t4)$	grant
6	$p4, p5, p6$	b	a	c	a	-	$(b, t5)$	grant
7	$p7$	b	a	c	a	b	-	-

Outline

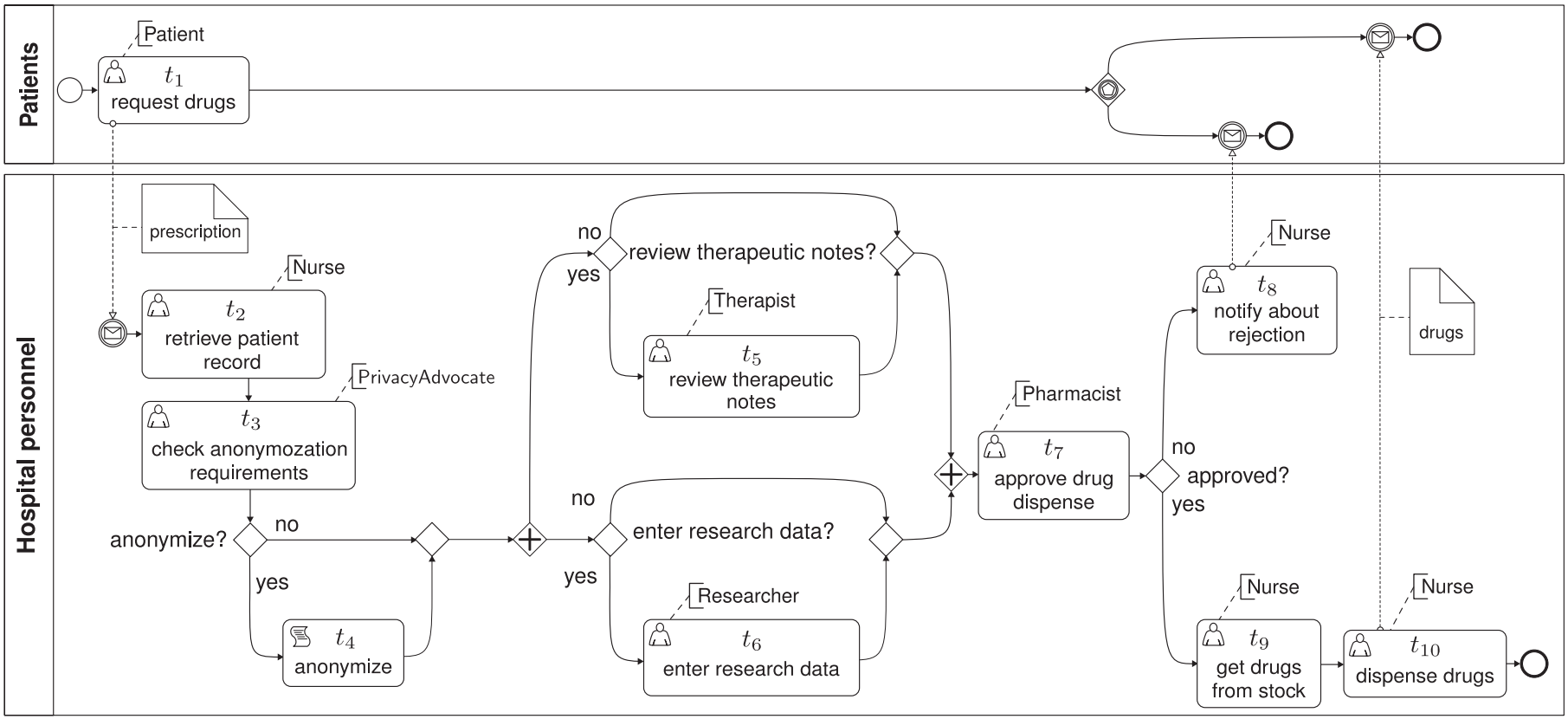
Description

- Real-world for validity
  - Moderate number of tasks
  - Complex control-flow
- Synthetic for scalability
  - Huge number of tasks
  - Simple control-flow
- State space explosion $\rightarrow$ need for heuristics

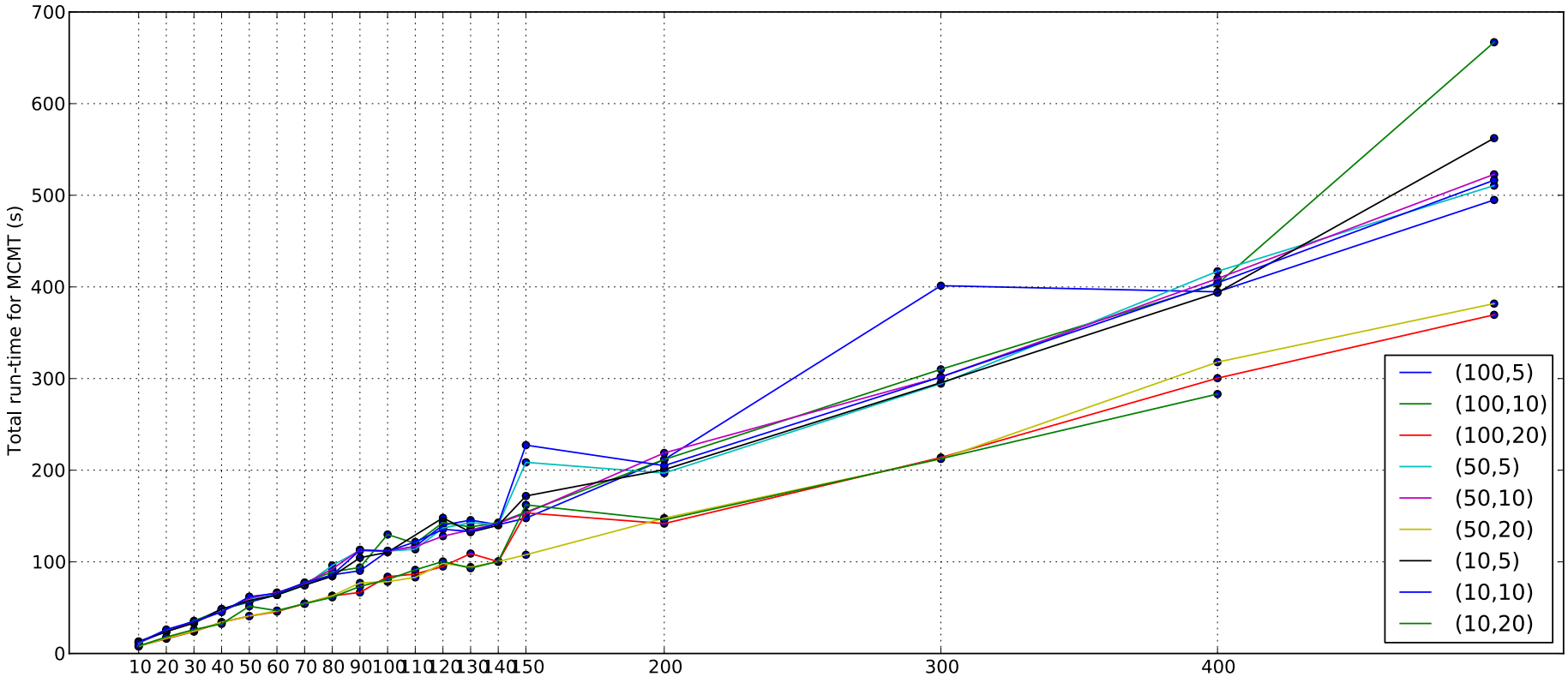
Hierarchical descriptions



Drug dispensation process



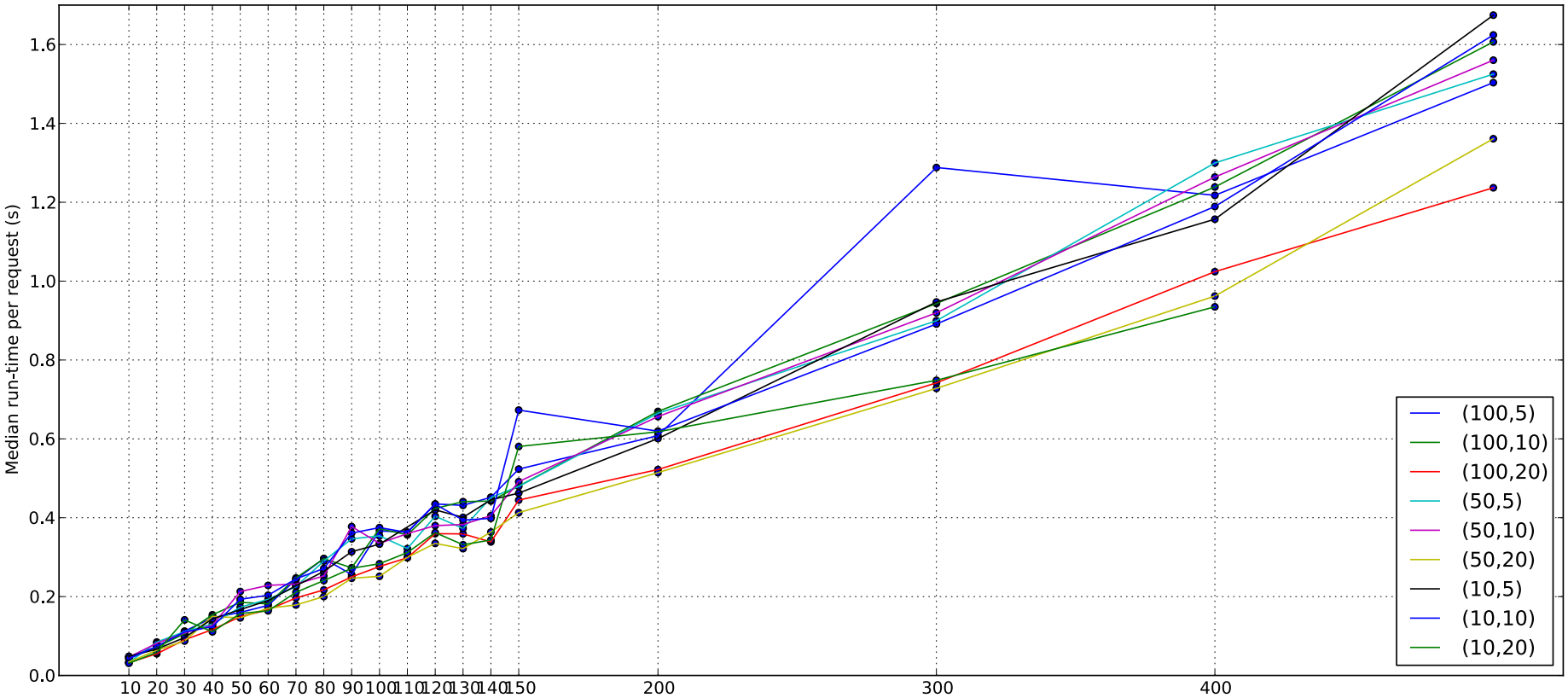
Off-line



Number of tasks

(pa,pc) = (authorization density, constraint density)

On-line



Number of tasks

(pa,pc) = (authorization density, constraint density)

Outline

Conclusions

- Synthesize run-time monitors to ensure the termination of workflows with authorization constraints
  - Off-line: compute a symbolic representation of all behaviors
  - On-line: add the authorization policy and derive the monitor
- Parametric wrt the number of users
- Scalable with the use of hierarchical representations
- Future work: integrate the monitor in a real workflow engine to collect data about performance

Thank you!

www.secentis.eu

bertolissi@fbk.eu
dossantos@fbk.eu
ranise@fbk.eu

UNIVERSITY
OF TRENTO