General Framework for Opacity Supervision

Nour Elhouda SOUID & Kais Klai

{souid, kais.klai}@lipn.univ-paris13.fr

LIPN Research Lab
University Sorbonne Paris Nord

23 March 2022

Outline

- Introduction
- Background
- Proposed Approach
- Web-Service Use Case
- Developed Tool
- Conclusion & Perspectives

Motivation: Cybersecurity

- Vulnerable systems used daily
- The severity of the damages caused by recent attacks (ransomware[1], Deny of Service[2]).

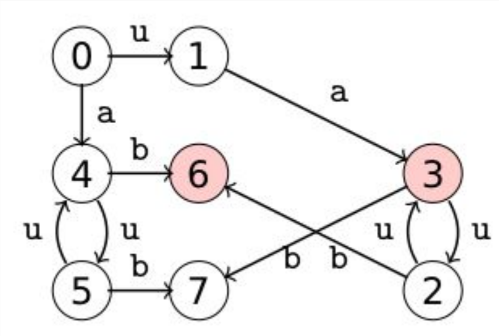
→ In this context, formal methods appear as a reliable technique to model systems and verify their security properties⇒ information flow

- Opacity: a malicious third party is able to deduce that the system is in a secret state?

- 1: (e.g., TeslaCrypt in 2015, WannaCry in 2017)
- 2: (e.g., the MiraiKrebs, OVH DDoS in 2016)

Preliminaries: The Opacity Property

- Defined w.r.t secret predicate (a set of secret states/ runs) & an observer considered as an attacker.
- The predicate φ is opaque if no attacker can ever conclude from its provided interface (observation) that the current run r of the system satisfies φ (r |= φ).
- Formal Definition : $\forall r \in L(T)$ such that $r \models \varphi$, there exists $r' \in L(T)$ such that $(r \sim r') \wedge (r' \not\models \varphi)$



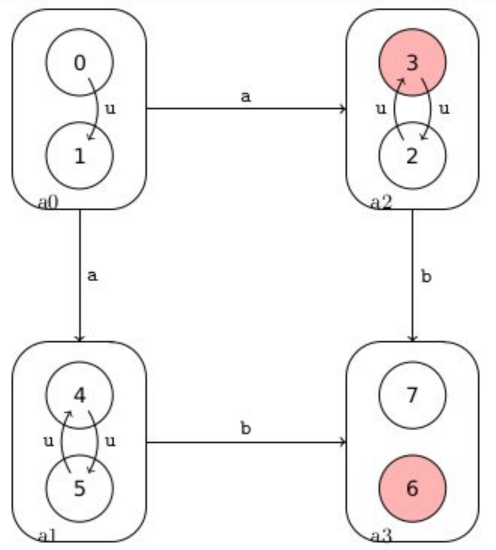
Attacker observation= {a, b}

Opaque system !

Preliminaries: Symbolic Observation Graph (SOG)

Verifying the opacity→ State explosion problem
⇒ regroup states into "aggregates" ⇒ SOG

- Deterministic graph where each node is a set of states linked by unobservable actions and each arc is labeled with an observable action.
- Nodes of the SOG are called aggregates→ managed efficiently using decision diagram techniques
- Complexity?
- SOG opaque ⇔ NONE of its aggregates is included in the secret

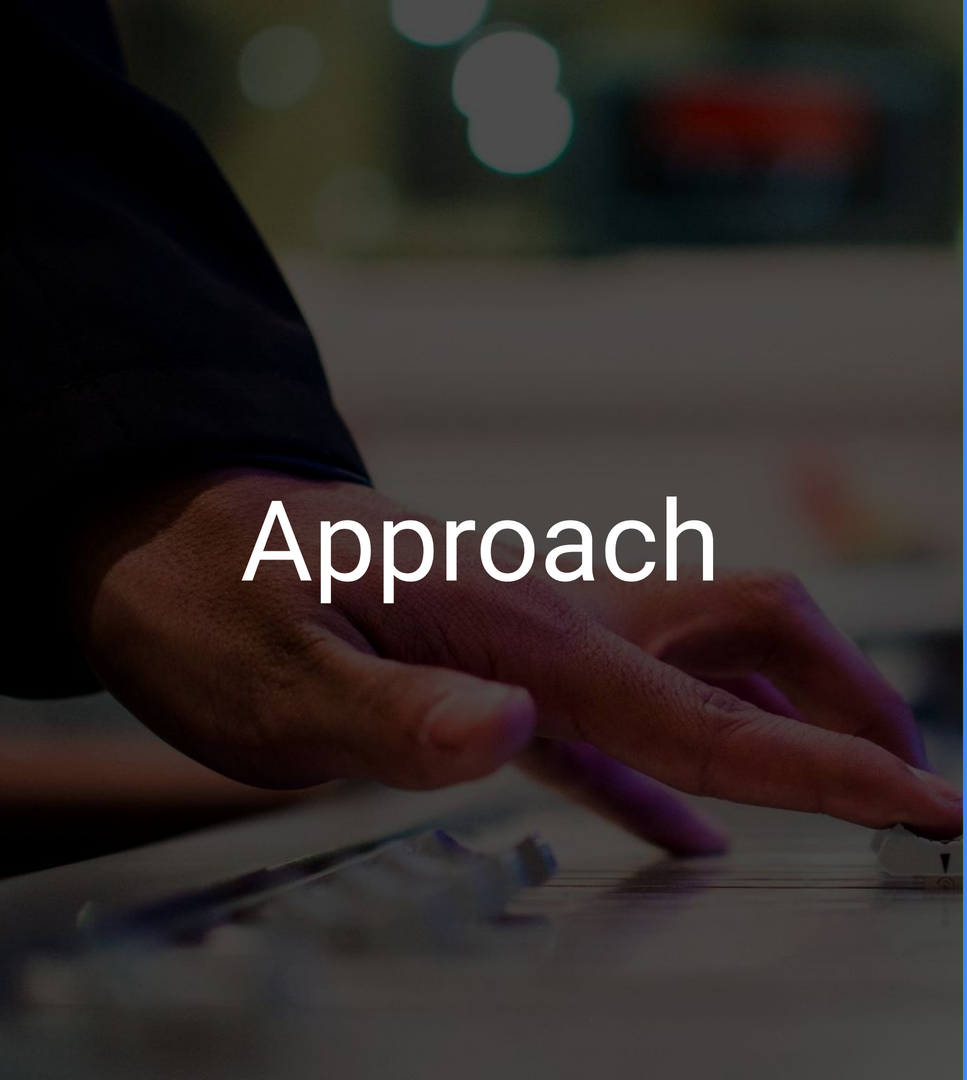


opaque systems

Preliminaries: Supervisory Control Background (SCT)

- A formal framework for modeling and control of Discrete Event Systems (DESs).
- Objective: synthesize a supervisor → can prevent some actions from occurring to enforce security properties.
- Supervisor : Partial observer (Σ_m) & controls only a subset of events (Σ_c).
- The supervisor can be viewed as a function (**γ**) : returns a set of actions to be disabled after the observation of a trace. ⇒ **γ**(tr)={c1, c2}
- **Permissiveness**

Approach



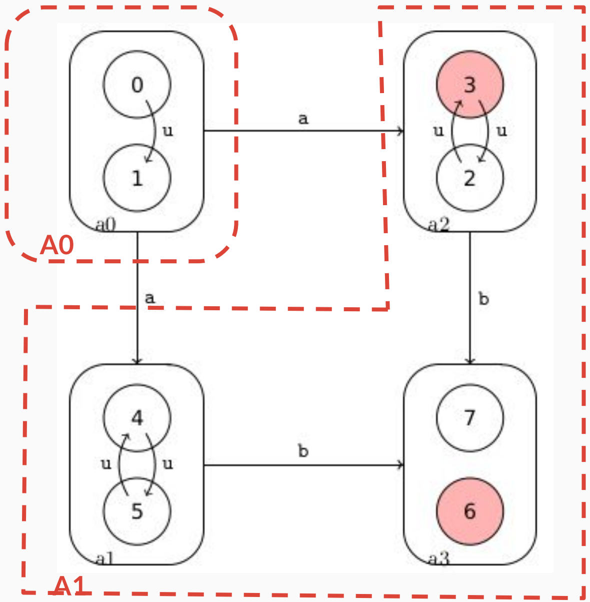
Reinforcing the opacity of a (DES) from the SCT perspective

Suggest a novel methodology to synthesize a maximal supervisor

→ restricts the behavior without any hypothesis on the relationship between the attacker and the supervisor observations.

General Framework for Opacity Reinforcement: HSOG

Notation:
- Attacker Observation Σ_a = { a }
- Supervisor Observation Σ_m = { b }

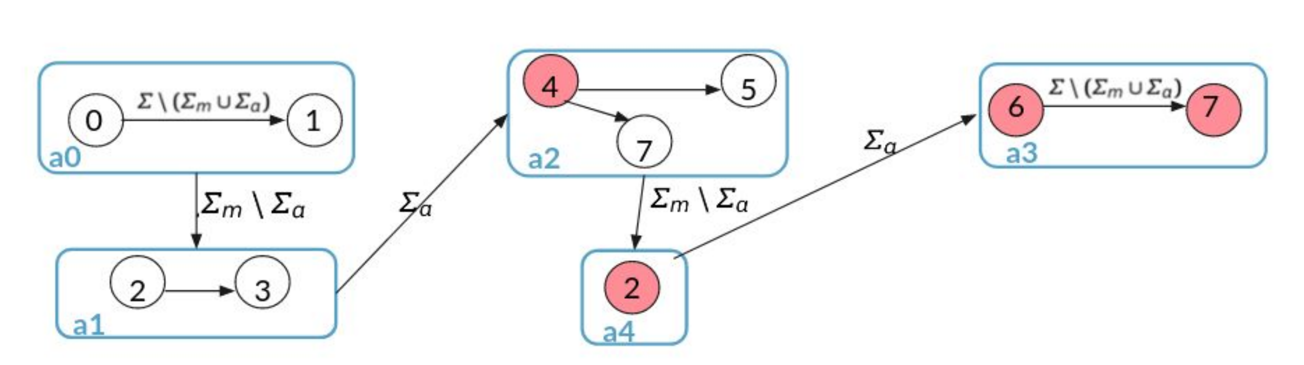


- Hyper Symbolic Observation Graph
  - Nodes **[super aggregates]**: sets of aggregates (**not single states**)
  - **Actions** in $\Sigma_m \setminus \Sigma_a$ and
  - **Arcs** are labeled with actions in Σ_a

⇒ Representing state space in a condensed manner
⇒ Alleviate the explosion state problem

How to obtain an HSOG?

1. Build the SOG of the system based on $\Sigma_a \cup \Sigma_m$

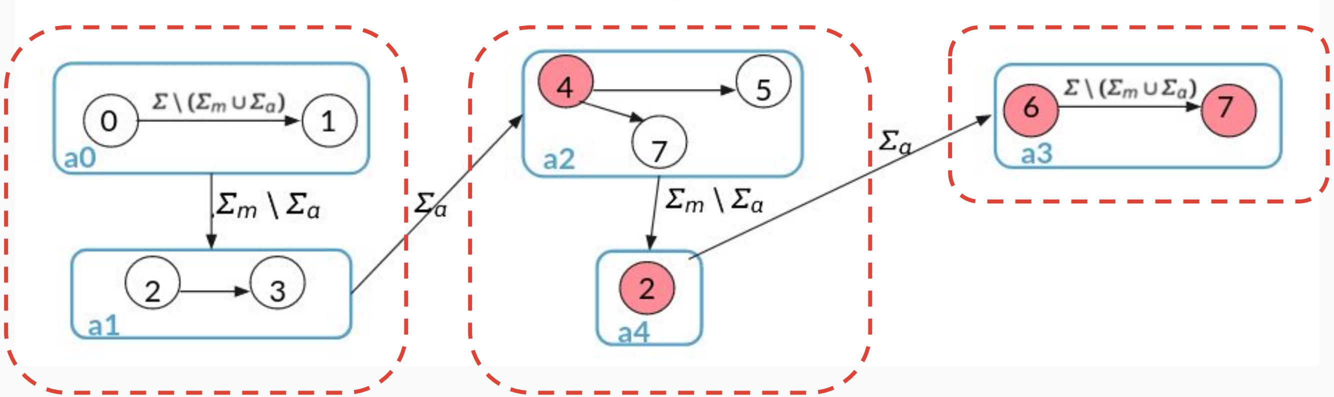


Notation:
- Attacker Observation Σ_a
- Supervisor Observation Σ_m

How to obtain an HSOG?

2. Consider the obtained SOG as a LTS

3. Build the corresponding SOG based on Σ_a only.



HSOG opaque ⇔ NONE of its aggregates is included in the secret

Notation:
- Attacker Observation Σ_a
- Supervisor Observation Σ_m

Approach: How it works

Step 1

On-the-fly HSOG Construction

→ Abstraction of the state space according to the attacker's observation

Step 2

Check the condition of opacity violation

→ A super-aggregate [node] is totally included in the secret?

Step 3

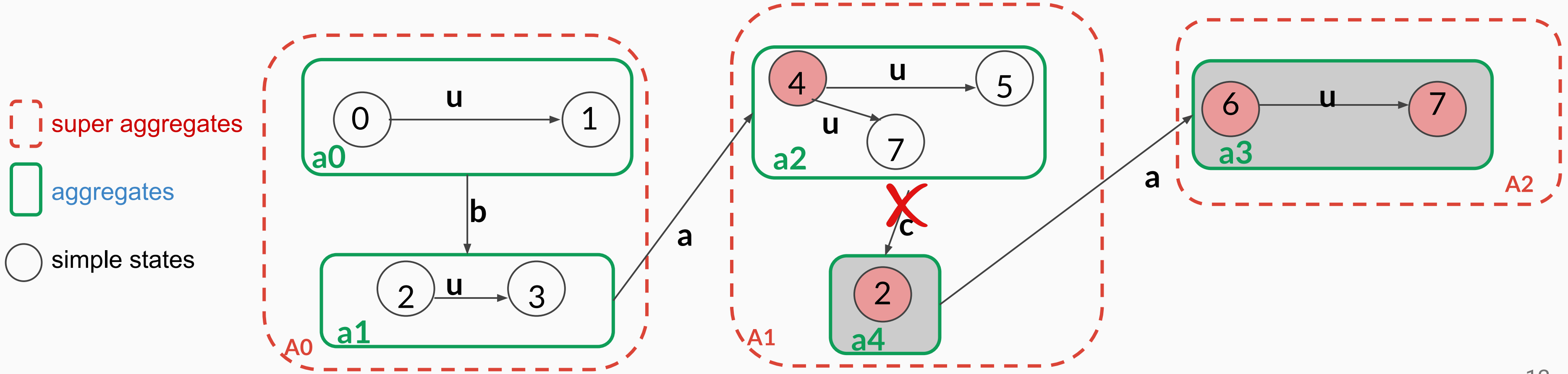
Enforce the opacity

→ Backtracking + disable the last controllable event

General Framework for Opacity Reinforcement: HSOG Example

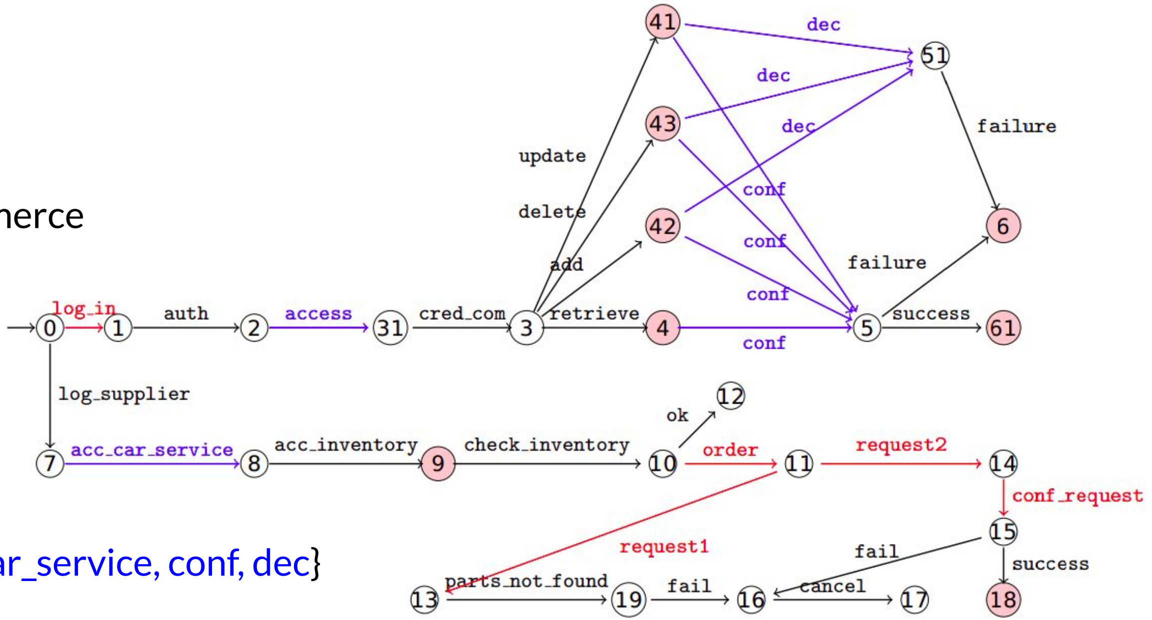
Notation:
- Attacker Observation Σ_a ={ **a** }
- Supervisor Observation Σ_m ={ **b**, **c** }
- Supervisor controls $\Sigma_c \subseteq \Sigma_m$ ={ **c** }
- $\Sigma \setminus (\Sigma_m \cup \Sigma_a)$ ={ **u** }

Supervisor: γ(u b u a u) = { c }



Application to a Web Service Use Case: Scenario Description

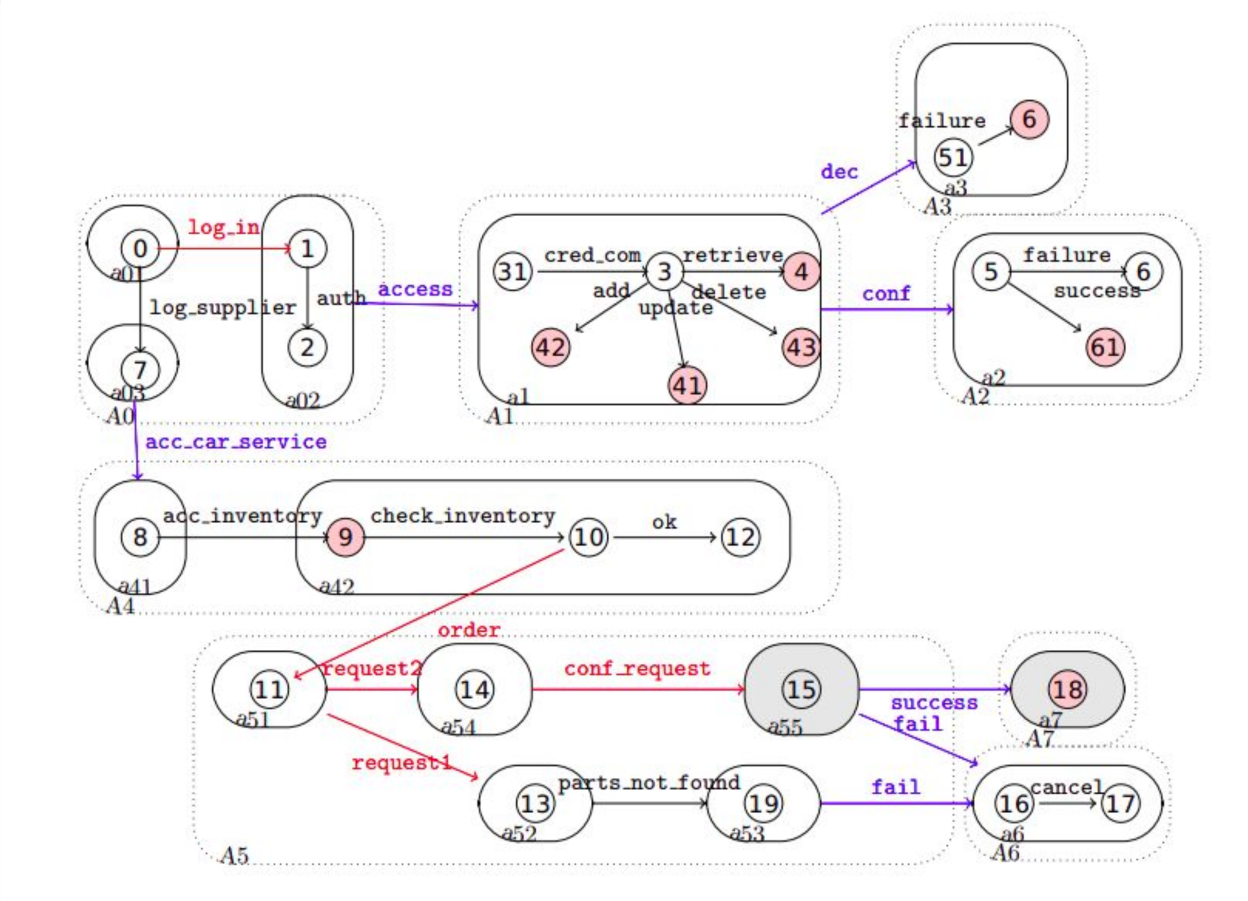
- B2B (business-to-business) e-commerce
- Supply chain relationship between:
  - a **car dealer**
  - a **manufacturer**
  - a **part supplier**



- Attacker's observation={access, acc_car_service, conf, dec}
- Secret states={4,42,43,41,6,61,918,}
- Supervisor's observation={log_supplier, acc_inventory, parts_not_found}
- Supervisor's control={log_in, order, request1, request2, conf_request}

Labelled Transition System representing the case study

Application to a Web Service Use Case: Supervision



Super aggregate ⊆secret??

- Supervisor:

γ(ε)={ conf_request }

Developed Tool:

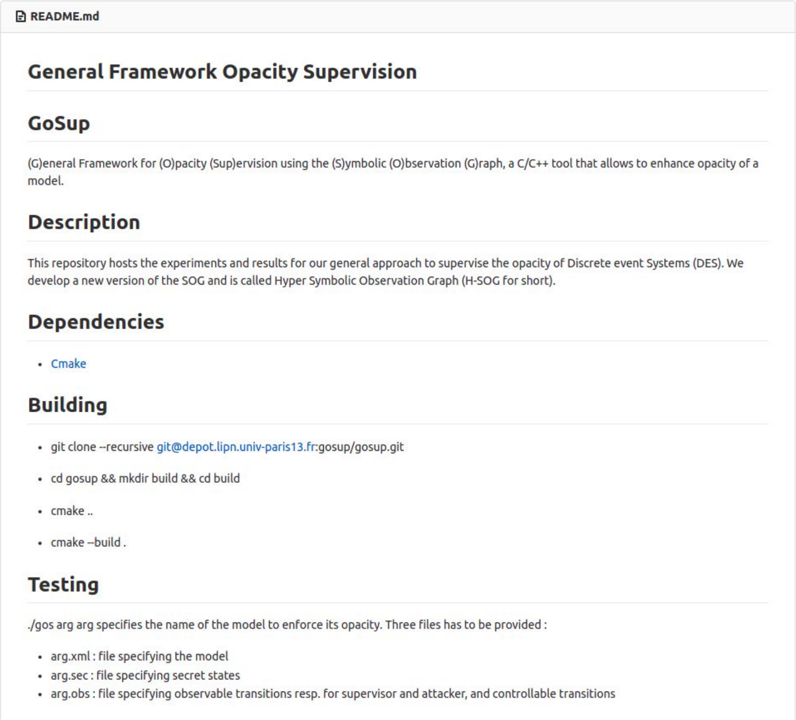
- C++ language based tool
- A tool to reinforce the opacity of DESs.
- Open source.

- Input:
  - The system [PNML file]
  - The confidential information [set of states]
  - The observable behaviour of the system [set of states]
  - The desired supervisor :
    - What to control
    - What to observe
- Output:
  - Supervision function → what actions to enable/disable

GoSup

General Opacity Supervision

https://depot.lipn.univ-paris13.fr/gosup/gosup



README.md

General Framework Opacity Supervision

GoSup

(G)eneral Framework for (O)pacity (Sup)ervision using the (S)ymbolic (O)bservation (G)raph, a C/C++ tool that allows to enhance opacity of a model.

Description

This repository hosts the experiments and results for our general approach to supervise the opacity of Discrete event Systems (DES). We develop a new version of the SOG and is called Hyper Symbolic Observation Graph (H-SOG for short).

Dependencies

- Cmake

Building

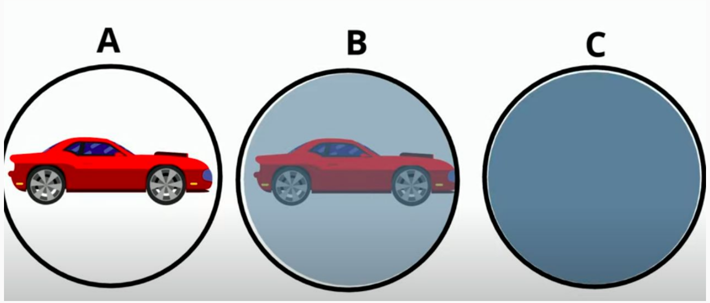
- git clone --recursive git@depot.lipn.univ-paris13.fr:gosup/gosup.git
- cd gosup && mkdir build && cd build
- cmake ..
- cmake --build .

Testing

./gos arg arg specifies the name of the model to enforce its opacity. Three files has to be provided :

- arg.xml : file specifying the model
- arg.sec : file specifying secret states
- arg.obs : file specifying observable transitions resp. for supervisor and attacker, and controllable transitions

Conclusion

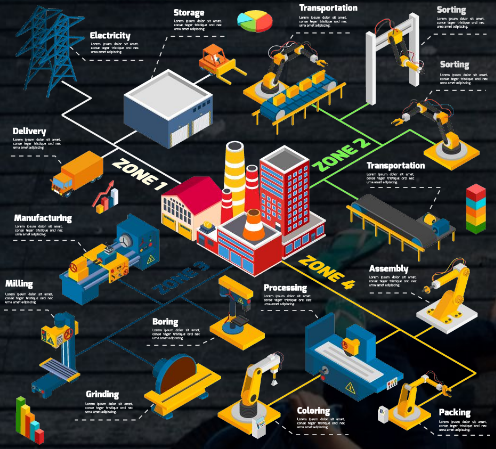


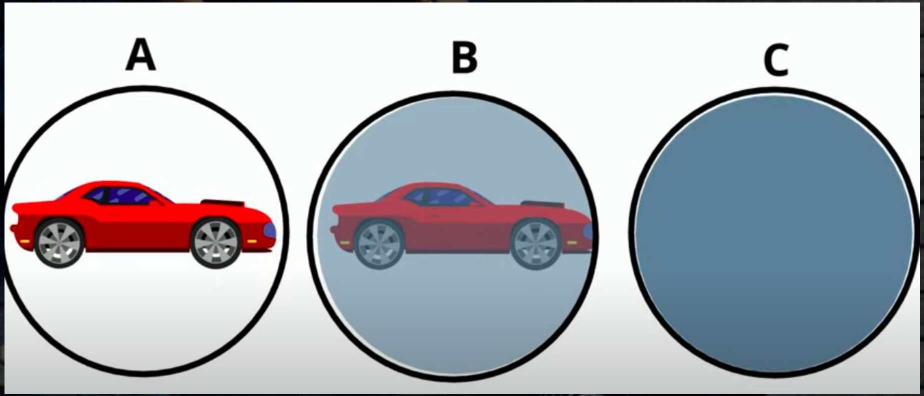
Why's next?

- Quantifying the opacity property
    - Modular systems
    - More attackers

- Proposed a **GENERAL** and **REDUCED-COST** algorithm → reinforce the opacity based on a novel graph called **HSOG.**
- **ON-THE-FLY** computation of the supervisor [performed while abstracting the system].
- Prove that the obtained supervisor language K is controllable, observable, supremal, ensures the opacity.
- Use case sample: security of a B2B e-commerce application.

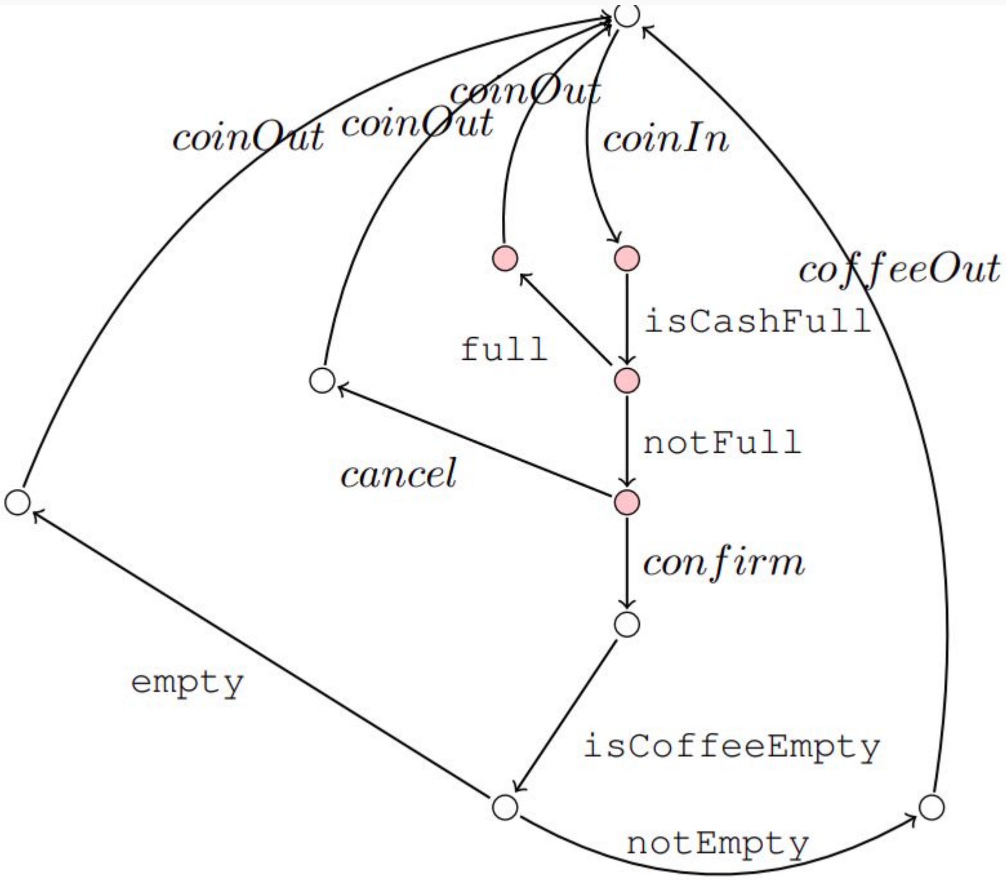
Thank you for your attention

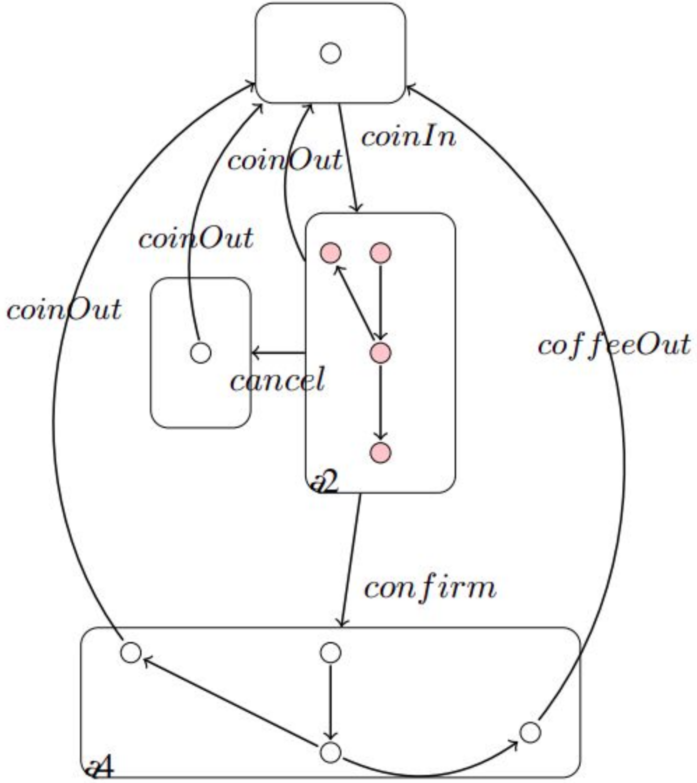

Electricity
Storage
Transportation
Sorting
Sorting
Delivery
ZONE 1
ZONE 2
Transportation
Manufacturing
Milling
Processing
ZONE 4
Assembly
Boring
Grinding
Coloring
Packing


A
B
C

[1] Serge Haddad, Jean-Michel Ilié, and Kais Klai. Design and evaluation of a symbolic and abstraction-based model checker. In Automated Technology for Verification and Analysis ATVA, volume 3299 of Lecture Notes in Computer Science, pages
196–210. Springer, 2004.

Preliminaries: Verifying the opacity using the SOG





General Framework for Opacity Reinforcement: Approach

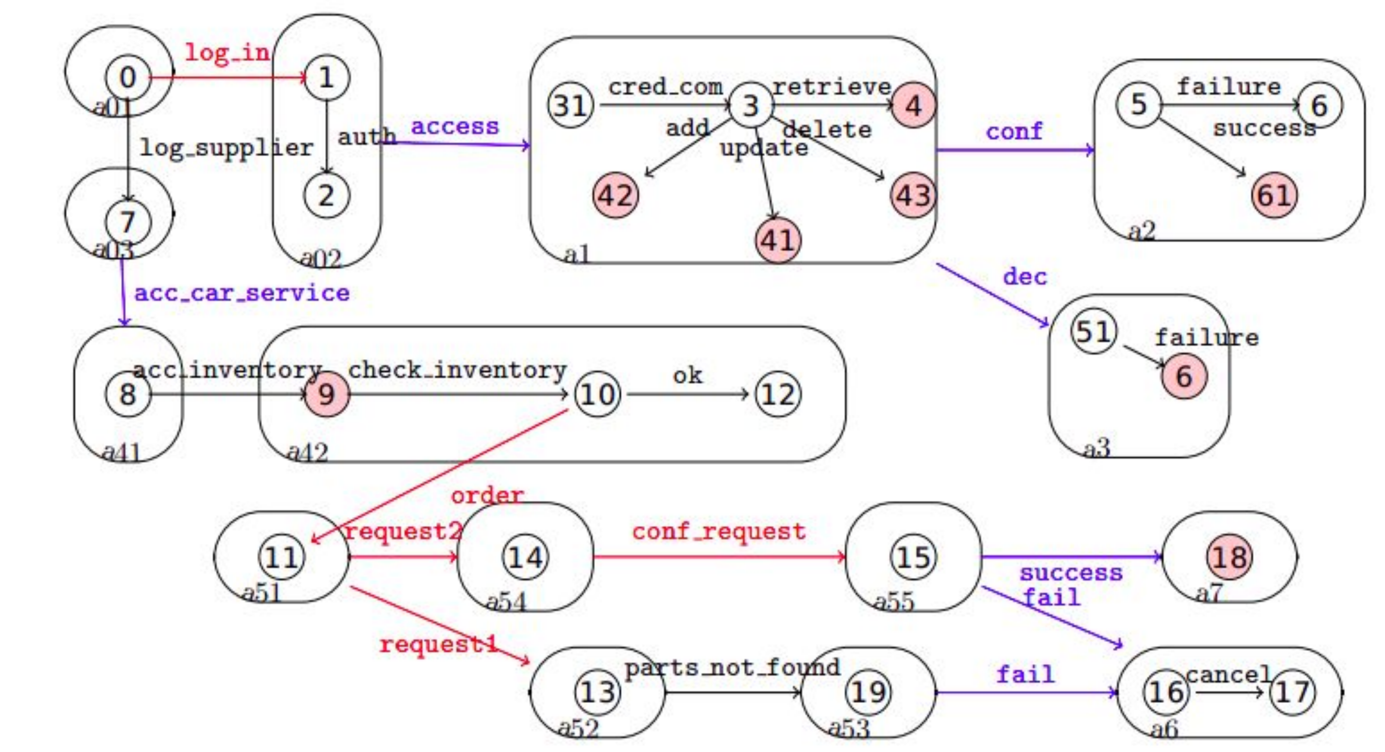
- Define the supervisor's behavior through a supervision function **γ.**
- Prove that the obtained supervisor language K is
  - controllable
  - observable
  - supremal
  - ensures the opacity.
- Propose an algorithm based on an on-the-fly construction of a new version of the SOG[1] called **Hyper Symbolic Observation Graph** (HSOG)

Developed Tool: GoSup

General Opacity Supervision

https://depot.lipn.univ-paris13.fr/gosup/gosup

Application to a Web Service Use Case: SOG of the use case



SOG of the use case