# Exploring current problems in CPS Theory $(\mathcal{S}, I)$

Thanh H. Nguyen

New Mexico State University

*tnguyen@cs.nmsu.edu*

January 27, 2020

# Overview

1. Problem 1: Complicated Relations between Properties and Concerns
2. Problem 2: The Conflicts between Properties
3. Problem 3: Evaluate the performance and effectiveness of current CPS Configuration
4. Problem 4: Timing Constraints
5. Problem 4: More Advanced Trustworthiness Queries

# Problem 1: Problem Description

- In CPS theory, there are 2 most important relations between *concern* and *property*: $addBy(C, P)$ and $subconcern(C, C_1)$
- A concern C is satisfied iff **all sub-concerns of C** are satisfied AND **all properties that address for C** are satisfied.
- However, there exists a case that: $\exists p_1, p_2 \in P, c \in C$ and $addBy(c, p_1) \wedge addBy(c, p_2)$ (Assume that $c$ has no sub-concerns), $sat(c)$ holds if $sat(p_1) \vee sat(p_2)$ holds.
- Example, properties $\{two\_factos\_auth, finger\_printing\_auth\}$ address concern *Authorization*. concern *Authorization* is satisfied if a physical device uses *two_factos_auth* OR *finger_printing_auth*.
- This idea is not appropriate with current CPS Theory.

# Problem 1: Solution – Extend CPS Ontology

- We propose new terminology: *Supplementary Property* (SP)
- We propose new relation between *Supplementary Property* and *Property*: $supportFor(SP, P)$ denotes that supplementary property $SP$ supports for property $P$.
- A property $P$ is satisfied IFF the truth value of $P$ is *true* OR one of supplementary properties of $P$ is True.
  `holds(sat(P),S):- 1{holds(sat(SP),S):supportFor(SP,P)}.`
- In CPS theory $(\mathcal{S}, I)$. The relation $r \in R$ denotes the relation between a component $c$ and a set of supplementary properties $sp$. The predicate $relation(c, sp)$ denotes that component $c$ is related with supplementary property $sp$.

# Problem 1: Changes in Planning Engine

- The extension of CPS Ontology will support to improve the reasoning and the mitigation strategies generation.
- The CPS action is not only able to turn ON/OFF the supplementary property (make the truth values of these properties True or False), but also is able to switch component to use authentication function between $\{two\_factors\_auth$ and $finger\_printing\_auth\}$.
- Generate the more powerful mitigation strategies (multiple types of actions which changes truth value of supplementary property AND changes the relation between component and supplementary properties).

# Problem 2: The Conflicts between Properties

- Assuming that, in CPS Theory $\exists p_1, p_2 \in P$, $\exists c_1, c_2 \in C$, $relation(c_1, p_1), relation(c_2, p_2) \in R$, $obs(p_1, true)$ and $obs(p_2, true)$.
- There exists a case that $p_1$ and $p_2$ are not able to hold at the same CPS state.
  `CONFLICT :- holds(sat(`$p_1$`),S), holds(sat(`$p_2$`),S).`
  etc.
- This situation causes the conflicts between properties in a state of CPS evolution.
- **Example 1**: In autonomous car, a sensor uses socket connection to transfer data — assume that socket is unique connection to be able to ensure *Integrity* concern. But socket connection is not secure and does not ensure the *Encryption* concern.
  `CONFLICT :- holds(use(sensor,socket_conn),S),`
  `holds(use(sensor,protocol_encrypted),S).`

# Problem 2: Solution

- I am still working on the solution for this issue

# Problem 3: Evaluate the performance and effectiveness of current CPS Configuration

- There exists a problem to evaluate the effectiveness and performance of different CPS configurations. Assuming that, these configurations make all related concerns are satisfied. But which configuration is better or worse than the others ?
- **Example 2**: LKAS system with components $C = \{SAM, CAM, Battery\}$. LKAS works perfectly if *SAM* and *CAM* in *advanced_mode*, but it causes *Battery* working on *low_mode* (not working normally – negative) because component in *advanced_mode* consumes a lot of energy. If one of *SAM*, *CAM* works on *basic_mode* then *Battery* will works in *normal_mode*. If all components in *basic_mode*, *Battery* will be *power_mode*. At least 3 configurations.
  ```
  holds(use(battery,low_mode),S) :-
  2{holds(use(X,advanced_mode), S) : component(X)}.
  ```
  (static causal laws)

# Problem 3: Solution – Likelihood of Satisfaction (Different from Dr Marcello's idea)

- In this issue, we need a value to measure how much satisfaction of related concerns. For example, in 3 above configurations, *Integrity* concern is satisfied but how much satisfy of *Integrity* for each configuration.
- Likelihood of Concern Satisfaction can solve this problem (Different from Dr Marcello's idea).
- We assign each property in CPS Theory a value called *sat_value*.

| | |
|---|---|
| advanced_mode | 0.8 |
| basic_mode | 0.6 |
| power_mode | 0.9 |
| normal_mode | 0.5 |
| low_mode | 0.2 |

- We calculate the likelihood of concern satisfaction based on the likelihood of concern satisfaction of sub-concerns AND *sat_value* of properties which address this concern.

# Problem 3: Solution – Likelihood of Satisfaction (Different from Dr Marcello's idea)

- For example in above configurations, assume that all properties address *Integrity* concern and the current configuration is that: *SAM*, *CAM* are in *advanced_mode*, *Battery* is *low_mode*, then $likelihood\_sat(Integrity) = 0.8 * 0.8 * 0.2 = 0.128$ (Assume that *Integrity* hasn't had any sub-concerns)
- *Integrity* and *Confidentiality* are sub-concerns of *Cyber_security* then $likelihood\_sat(Cyber\_security) =$ $likelihood\_sat(Integrity) * likelihood\_sat(Conf.)$
- By default, all satisfied likelihoods of concerns are 1.
- We keep recursively calculate *likelihood_sat* to the root of concern tree $likelihood\_sat(trustworthiness)$. Comparing the performance and effectiveness of different CPS configurations based on $likelihood\_sat(trustworthiness)$.

# Problem 4: Timing Constraints

- There are a lot of use cases related to Timing Constraints especially on Autonomous System.
- For example, in above configuration, if *SAM*, *CAM* are in *advanced_mode* causes *Battery* in *low_mode*. If *Battery* in *low_mode* is over limited time $\delta_t$ (seconds) then the system will be down.
  `SYSTEM_DOWN :- holds(use(battery,low_mode),S),`
  `starting_time(use(battery,low_mode),`$@t_0$`),`
  `current_time(@t), limit_in_low_mode(`$\delta_t$`),` $@t - @t_0 > \delta_t$.
- In order to prevent SYSTEM_DOWN, starting from time $@t_0$, a mitigation strategy has to be generated to fix the problem. The constraint is that the total effect time of plan execution CANNOT be over $\delta_t$. Total effect time is calculated from start executing the first action to getting the effects of last action in plan.
- In addition, an action taken at time point $@t$ can have its effects at $@t$ or a later time points after $@t$. (Temporal Planning)

# Problem 4: Formalization and Implementation

- I am still working on this part.

# More Advanced Trustworthiness Queries

- What are the robustness requirements/properties ? (Preventing a fault)
- What are the resiliency requirements/properties ? (Recovering from a fault or sub-fault)
- What happens if information within the system leaks?
  In this case, property $p$ is still True to make the addressed concerns satisfiable. However, the information still leaks. Need to change to another property $p'$ which *higher* than $p$. We need to define and reason about *higher* property.