TPLP: Page 1–8. © The Author(s), 2021. Published by Cambridge University Press 2021
doi:10.1017/xxxxx

Specifying and Reasoning about CPS through the Lens of the NIST CPS Framework

THANH HAI NGUYEN, MATTHEW BUNDAS, TRAN CAO SON
Department of Computer Science, New Mexico State University, Las Cruces, USA
(*e-mail:* thanhnh@nmsu.edu,bundasma@nmsu.edu,stran@nmsu.edu)

MARCELLO BALDUCCINI, KATHLEEN CAMPBELL GARWOOD
Saint Joseph's University, Philadelphia, USA
(*e-mail:* mbalducc@sju.edu,kcampbel@sju.edu)

EDWARD R. GRIFFOR
National Institute of Standards and Technologies, USA
(*e-mail:* edward.griffor@nist.gov)

submitted 04-Jun-2021; revised 24-Oct-2021; accepted 21-Jan-2022

Abstract

This paper introduces a formal definition of a Cyber-Physical System (CPS) in the spirit of the CPS Framework proposed by the National Institute of Standards and Technology (NIST). It shows that using this definition, various problems related to concerns in a CPS can be precisely formalized and implemented using Answer Set Programming (ASP). These include problems related to the dependency or conflicts between concerns, how to mitigate an issue, and what the most suitable mitigation strategy for a given issue would be. It then shows how ASP can be used to develop an implementation that addresses the aforementioned problems. The paper concludes with a discussion of the potentials of the proposed methodologies.

KEYWORDS: Artificial Intelligence, Knowledge Representation, Automated Reasoning and Planning, Cyber-Physical System, Answer Set Programming, Concern Satisfaction, CPS Ontology

1 Introduction

The utility (potable water, wastewater) distribution systems, the electric power grid, the transportation network, automated driving systems (ADS), hospital robots, and smart-home systems are a few examples of cyber-physical systems (CPS)[1] that are (or soon to be) a part of our daily life. Before any CPS is deployed into the real-world, several concerns need to be investigated and addressed, e.g., why should someone trust that the CPS will perform its functions safely, securely and reliably? How will such a system respond to a certain critical conditions and will that response be acceptable? In other words, evidence must be gathered and argued to be sufficient to conclude that critical properties of a CPS have been assured before its deployment. For financial and practical reasons, the validation and verification of a CPS should be done as early as possible, starting with its design. CPS are complex systems that evolve with use, requiring a principled methodology and tools for developing an assurance case before release to the market.

[1] For brevity, we use CPS to stand for both the plural and the singular cyber-physical system.

Such a methodology and the tools for applying it are two key contributions of this paper. We present here a formalization of a CPS with a clearly defined semantics that enables the assessment of critical system properties. The need for such a foundation for assurance can be seen in the next example.

Example 1
Suppose that we would like to develop an Automated Driving System (ADS). We have two constraints that we would like to enforce: (*a*) packets sent from the wind-sensor, a part of the situational awareness module (`SAM`), to the main processor must be fast and reliable; (*b*) all communication channel must be encrypted. We will refer to (*a*) and (*b*) as an `Integrity` concern and `Encryption` concern, respectively.

Consider a situation in which the ADS has only one possible communication channel, which is fast, reliable when encryption is disabled, but is not when encryption is enabled. In this situation, the two constraints are in conflict with each other. It is impossible to satisfy both of them.

Assume that we also have some preference, called `Verification`, which is related to the verification of received data. Encrypted data would have been preferred to non-encrypted one. If the wind-sensor uses the non-encrypted socket communication, it can satisfy (or *positively* affect) the `Integrity` concern but it does not satisfy (or *negatively* affect) the `Verification` preference.

In this paper, we view a CPS as a dynamic system that consists of several components with various constraints and preferences which will be referred as *concerns* hereafter. Given a concrete state of the system, a concern might or might not be satisfied. We aim at laying the mathematical foundation for the study of CPS' concerns. This foundation must allow CPS developers and practitioners to represent and reason about the concerns and answer questions such as (*i*) will a certain concern or a set of concerns be satisfied? (*ii*) is there any potential conflict between the concerns? and (*iii*) how can we generate the best plan that addresses an issue raised by the lack of satisfaction of a concern? Readers familiar with research in representing and reasoning about dynamic systems might wonder whether well-known formalisms for representing and reasoning about dynamic systems such as automata, action languages, Markov decision process, etc. could be used for this purpose. Indeed, our proposed framework extends these formalisms by adding a layer for modeling the components and concerns in CPS.

To achieve our goal, we propose a formalism for representing and reasoning about concerns of CPS. We will focus on the properties described in the CPS Framework (CPSF) proposed by the CPS Public Working Group (CPS PWG) organized by the National Institute of Standards and Technology (NIST) Griffor et al. (2017a;b); Wollman et al. (2017). This framework defines several important concepts related to CPS such as *facets* (modes of the system engineering process: conceptualization, realization and assurance), *concerns* (areas of concern), and *aspects* (clusters of concerns: functional, business, human, trustworthiness, timing, data, composition, boundaries, and lifecycle). These concepts are organized in an ontology which is easily extensible and allows us to better manage development and implementation within, and across, multiple application domains. We formally propose the notion of a CPS system that (*i*) considers constraints among concerns; (*ii*) enables the automatic identification of conflicts between concerns; and (*iii*) enables the application of planning techniques in computing mitigation strategies. Building and establishing upon CPSF are important properties of our research, which distinguish it from much of the work done on CPS so far. While most of the prior research is focused on a specific class of CPS or of aspects, e.g., CPS for smart grids or concerns related to cybersecurity Uluagac et al.

(2019), the methodology we provide is intentionally domain-independent and applicable to any class of CPS.

The paper is organized as follows. Section 2 presents a brief overview of the CPS framework, answer set programming, action language, and reasoning with ontologies using answer set programming. Section 3 contains the main contribution of the paper, a formalization of a CPS theory, which includes a specification of CPS domain and the semantics defining when a concern is satisfied. It also formally defines several reasoning tasks related to the satisfaction of concerns such as (*i*) when is a concern satisfied; (*ii*) what are the most/least trustworthy components of a CPS system; (*iii*) is the CPS system compliant; (*iv*) computing a mitigation strategy for a system when some concerns become unsatisfied; (*v*) which mitigation strategy has the best chance to succeed. Section 4 provides an answer set programming implementation of the tasks. The paper concludes with the discussion of the related work. The paper is arranged in a way such that it can be of interest to different groups of readers. Specifically, it separates the formal definitions of a CPS, and the reasoning tasks associated with it, from a concrete implementation of the reasoning tasks. As such, a reader only interested in the formal theories would likely be interested in Section 3. On the other hand, the code in Section 4 would be of interest to readers who would like to experiment with their own CPS.

2 Background

This section reviews the background notions that will be used in the paper, including the CPS ontology, answer set programming, and the use of logic programming in ontology reasoning.

2.1 NIST CPS Framework and the CPS Ontology

One of the major challenges in designing, maintaining and operating CPS is the diversity of areas of expertise involved in these tasks, and in the structure of the CPS itself. For example, developing a "smart ship" Moschopoulos (2001) involves close interaction among, and cooperation of, experts in disciplines ranging from cybersecurity to air conditioning systems and from propulsion to navigation. As demonstrated by, e.g., NASA's Mars Climate Orbiter[2], ensuring a *shared understanding* of a CPS and the interoperability of its components is an essential step towards its success – a goal that is made even more elusive by the fact that the areas of knowledge relevant to a CPS vary greatly depending to the type of CPS considered.

For this purpose, NIST recently hosted a Public Working Group on CPS with the aim of capturing input from those involved in CPS to define a *CPS reference framework* supporting common definitions and facilitating interoperability between such systems, regardless of the type of CPS considered. A key outcome of that work was the CPS Framework (Release 1.0, published as three separate NIST Special Publications Griffor et al. (2017a;b); Wollman et al. (2017)), which proposes a means of describing three *facets* during the life of a CPS: conceptualization, realization, and assurance of CPS; and to facilitate these descriptions through analytical lenses, called *aspects*, which group common concerns addressed by the builders and operators of the CPS. The CPS Framework articulates the artifacts of a CPS in a precise way, including the concerns that motivate important requirements to be considered in conceptualizing, realizing

[2] https://www.simscale.com/blog/2017/12/nasa-mars-climate-orbiter-metric/

(including operating), and assuring CPS. Albeit helpful, being a reference framework the CPS Framework only helps with the specification of a CPS and the discussion among experts. It does not, by itself, reduce the amount of work necessary to analyze the CPS and its evolution of the CPS lifecycle.

This realization gave impulse to the investigation that ultimately resulted in the *CPS Ontology* Balduccini et al. (2018); Nguyen et al. (2020a), which provides a CPS analysis methodology based on the *CPS Framework* featuring a vocabulary that describes and supports the understanding and development of new and existing CPS, including those designed to interact with other CPS and function in multiple interconnected infrastructure environments.

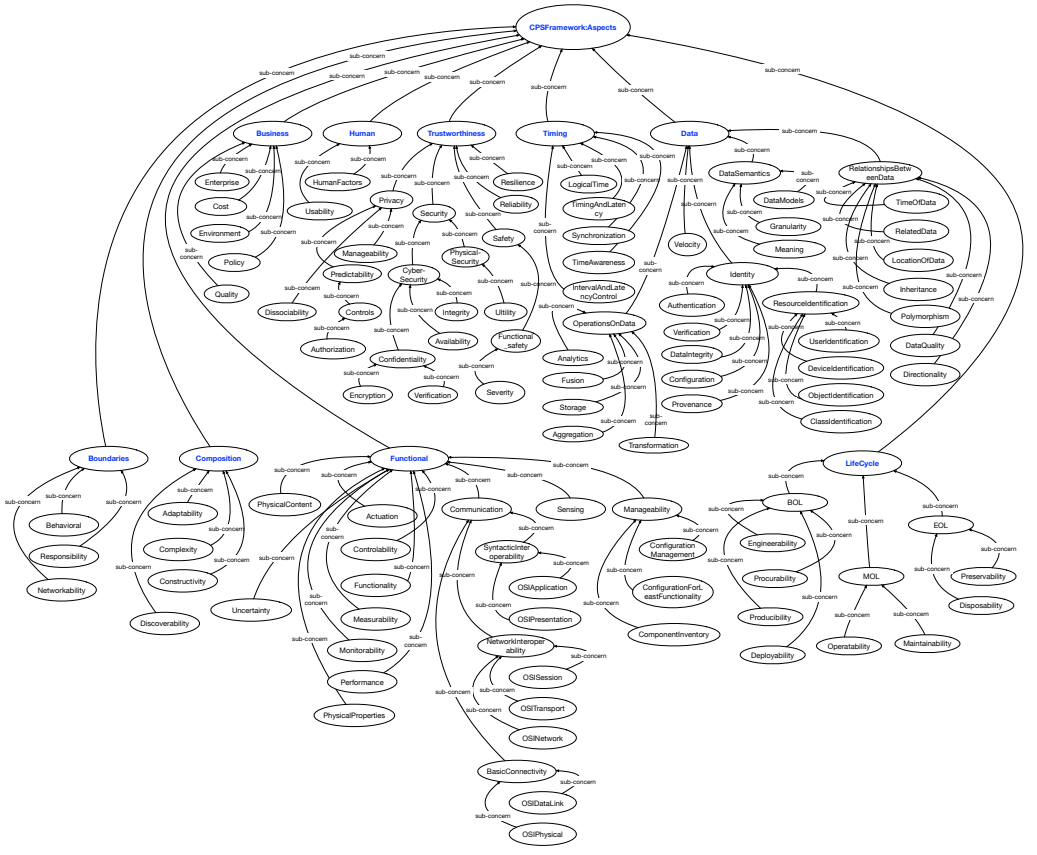

Fig. 1: NIST CPS Ontology

At the core of the CPS Framework and of the CPS Ontology are the notions of domains, facets (conceptualization, realization and assurance), aspects and concerns, and a cyber-physical functional decomposition. The product of the conceptualization facet is a model of the CPS (requirements added to address prioritized concerns), the product of the realization facet is a CPS satisfying the model and the product of the assurance facet is assurance case for the prioritized set of concerns. *Domains* represent the different application areas of CPS such as automated driving systems, electrical grid, etc. *Concerns* are characteristics of a system that one or more of its stakeholders are concerned about. They are addressed throughout the lifecycle of a CPS, including development, maintenance, operation and disposal. *Requirements* are assertions about

the state variables of a CPS aimed at addressing the concerns. The reader should note that, in line with the current CPSF specification, we consider the term *property* to be a synonym of requirement, and we use the two terms interchangeably in the rest of this paper. *Artifacts* are the elements of products of the facets for a CPS and include requirements, design elements, tests, and judgments. *Aspects* are the ten high-level concerns of the CPS Framework: functional, business, human, trustworthiness, timing, data, communication, boundaries, composition, and lifecycle.

- *Functional* aspect is a set of concerns related to the sensing, computational, control, communications and actuation functions of the CPS.
- *Business* aspect includes the concerns about enterprise, time to market, environment, regulation, cost, etc.
- *Human* aspect is a set of concerns related to how a CPS is used by humans or interacts with them.
- *Trustworthiness* aspect is a set of concerns related to the trustworthiness of CPS including security, privacy, safety, reliability, and resilience. In this paper we adopt the definition of trustworthiness from the NIST CPS Framework, where the term is taken to denote the demonstrable likelihood that the system performs according to designed behavior under any set of conditions as evidenced by its characteristics.[3]
- *Timing* aspect: Concerns about time and frequency in CPS, including the generation and transport of time and frequency signals, time-stamping, managing latency, timing composability, etc.
- *Data* aspect includes the concerns about data interoperability including data semantics, identify operations on data, relationships between data, and velocity of data.
- *Communications* aspect includes the concerns about the exchange of information between components of a CPS.
- *Boundaries* aspect is set of concerns about the interdependence among behavioral domains. Concerns related to the ability to successfully operate a CPS in multiple application area.
- *Composition* aspect includes the concerns about the ability to compute selected properties of a component assembly from the properties of its components. Compositionality requires components that are composable: they do not change their properties in an assembly. Timing composability is particularly difficult.
- *Lifecycle* aspect: Concerns about the lifecycle of CPS including its components.

The *CPS Ontology* defines concepts and individuals related to concepts (with focus on *Trustworthiness*) and the relationships between them (e.g., has-subconcern). Figure 2, excluding the nodes labeled `CAM`, `SAM` and `BAT` and links labeled "relates" and "active", shows a fragment of the CPS ontology where circle nodes represent specific concerns and grey rectangle nodes represent properties. To facilitate information sharing, the CPS Ontology leverages standards such as the Resource Description Framework (RDF[4]) and the Web Ontology Language (OWL[5]) for describing the data, representing the entities and their relationships, formats for encoding the data and related metadata for sharing and fusing. An entity or relationship is defined in the ontology

[3] This is a pragmatic choice dictated by our intent to provide a formal account of the NIST CPS Framework. The debate on a universally accepted definition of trustworthiness is on-going and is beyond the scope of this paper.
[4] https://www.w3.org/TR/rdf-concepts/
[5] https://www.w3.org/TR/owl-features/

by an RDF-triple (*subject*, *predicate*, *object*). Below are the main classes and relationships in the CPS ontology.

Aspects and Concerns. The ontology defines the highest-level concept of *Concern* with its refinement of *Aspect*. In the concern tree in Figure 1, the circle nodes of a concern tree represent specific concerns which are individuals of class *Concern*. The root nodes of the concern tree is a particular kind of concern that is an instance of class *Aspect* (subclass of *Concern*). Specific concerns are represented as individuals: `Trustworthiness` as an individual of class *Aspect*, `Security` and `Cybersecurity` of class *Concern*. Edges linking aspects and concerns are represented by the relation `has-subconcern`. A relation `has-subconcern` is used to associate a concern with its sub-concerns. Thus, `Trustworthiness` aspect `has-subconcern Security`, which in turn `has-subconcern Cybersecurity`.

Properties. Properties of a CPS are represented by individuals of class *Property*. In the CPS Framework, a concern can be addressed by a combination of properties. An edge that links a property p with an aspect or concern c is represented by the relation `addressed-by`, which says that concern c is addressed by property p. For example in Figure 2 (LKAS domain), concern `Integrity` has been addressed by some properties: `Secure-Boot`, `Advanced-Mode`, `Powerful-Mode`, `Normal-Mode` and `Saving-Mode`.

To ease the reading, we provide a summary of the main classes and relationships in the CPS ontology in Table 1.

2.2 Answer Set Programming

Answer Set Programming (ASP) Marek and Truszczyński (1999); Niemelä (1999) is a declarative programming paradigm based on logic programming under the answer set semantics. A logic program Π is a set of rules of the form:

$$c \leftarrow a_1, \ldots, a_m, not\ b_1, \ldots, not\ b_n$$

where c, a_i's, and b_i's are literals of a propositional language[6] and *not* represents (default) negation. c can be absent. Intuitively, a rule states that if a_i's are believed to be true and none of the b_i's is believed to be true then c must be true. For a rule r, r^+ and r^-, referred to as the *positive* and *negative* body, respectively, denote the sets $\{a_1, \ldots, a_m\}$ and $\{b_1, \ldots, b_n\}$, respectively.

Let Π be a program. An interpretation I of Π is a set of ground atoms occurring in Π. The body of a rule r is satisfied by I if $r^+ \subseteq I$ and $r^- \cap I = \emptyset$. A rule r is satisfied by I if the body of r is satisfied by I implies $I \models c$. When c is absent, r is a constraint and is satisfied by I if its body is not satisfied by I. I is a model of Π if it satisfies all rules in Π.

For an interpretation I and a program Π, the *reduct* of Π w.r.t. I (denoted by Π^I) is the program obtained from Π by deleting *(i)* each rule r such that $r^- \cap I \neq \emptyset$, and *(ii)* all atoms of the form *not a* in the bodies of the remaining rules. Given an interpretation I, observe that the program Π^I is a program with no occurrence of *not a*. An interpretation I is an *answer set* Gelfond and Lifschitz (1990) of Π if I is the least model (wrt. $\subseteq$) of Π^I.

A program Π can have several answer sets, one answer set, or no answer set. Π is said to be consistent if it has at least one answer set; it is inconsistent otherwise. Several extensions (e.g., *choice atoms*, *aggregates*, etc.) have been introduced to simplify the use of ASP. We will use and

[6] For convenience, we often use first order logic literals under the assumption that they represent all suitable ground instantiations.

Class	**Meaning**
Concern	Concerns that stakeholders have w.r.t. to a system, such as *security*, *integrity*, etc. They are represented in the ontology as individuals. The link between a concern and its sub-concerns is represented by the `has-subconcern` relation.
Aspect	High-level grouping of conceptually equivalent or related cross-cutting concerns (i.e., *human*, *trustworthiness*, etc). In the ontology, *Aspect* is subclass of class *Concern*.
Property	Class of the properties relevant to a given CPS. The fact that a property addresses a concern is formalized by relation `addressed-by`.
Configuration	Features of a CPS that characterize its state, e.g., if a component is on or off. When property satisfaction can change at run-time, corresponding individuals will be included in this class.
Action and Constraint	Actions are those within the control of an agent (e.g., an operator) and those that occur spontaneously. Constraints capture dependencies among properties (e.g., mutual exclusion).

Object Property	**Meaning**
cpsf:hasSubCon	The object property represents the `has-subconcern` relationship between the concerns.
cpsf:addrConcern	The object property represents the `addressed-by` relation between a concern and a property.
cpsf:impactPositively	The object property represents positive impact relation between a *property* and a *concern*.

Table 1: Main components of the CPS Ontology

explain them when needed. Given a program Π and an atom a, we write $\Pi \models a$ to say that a belongs to every answer set of Π. $\Pi \mathrel{|\!\sim} a$ to say that a belongs to at least one answer set of Π.

We illustrate the concepts of answer set programming by showing how the 3-coloring problem of a bi-directed graph G can be solved using logic programming under the answer set semantics. Let the three colors be red (r), blue (b), and green (g) and the vertex set of G be $\{0,1,\ldots,n\}$. Let $\Pi(G)$ be the program consisting of

- the set of atoms $edge(u,v)$ for every edge (u,v) of G,
- for each vertex u of G, the rule stating that u must be assigned one of the colors red, blue, or green:

$$1\{color(u,g);color(u,r);color(u,b)\}1 \leftarrow$$

This rule uses the choice atom, introduced in Niemelä et al. (1999), to simplify the use of ASP. This atom says that exactly one of the atoms $color(u,g)$, $color(u,r)$, and $color(u,b)$ must be true.

- for each edge (u,v) of G, three rules representing the constraint that u and v must have different color:

$$\begin{aligned} \leftarrow \quad & color(u,r), color(v,r), edge(u,v) \\ \leftarrow \quad & color(u,b), color(v,b), edge(u,v) \\ \leftarrow \quad & color(u,g), color(v,g), edge(u,v) \end{aligned}$$

It can be shown that for each graph G, (i) $\Pi(G)$ has no answer set, i.e., is inconsistent iff the 3-coloring problem of G does not have a solution; and (ii) if $\Pi(G)$ is consistent then each answer set of $\Pi(G)$ corresponds to a solution of the 3-coloring problem of G and vice versa.

2.3 Action Language $\mathscr{B}$

We review the basics of the action description language $\mathscr{B}$ Gelfond and Lifschitz (1998). An action theory in $\mathscr{B}$ is defined over two disjoint sets, a set of actions **A** and a set of fluents **F**. A *fluent literal* is either a fluent $f \in \mathbf{F}$ or its negation $\neg f$. A *fluent formula* is a propositional formula constructed from fluent literals. An action domain is a set of laws of the following form:

$$\textit{Executability condition}: \quad \textbf{executable } a \textbf{ if } p_1, \ldots, p_n \tag{1}$$

$$\textit{Dynamic law}: \quad a \textbf{ causes } f \textbf{ if } p_1, \ldots, p_n \tag{2}$$

$$\textit{Static Causal Law}: \quad f \textbf{ if } p_1, \ldots, p_n \tag{3}$$

where f and p_i's are fluent literals and a is an action. (1) encodes an executability condition of an action a. Intuitively, an executability condition of the form (1) states that a can only be executed if p_i's hold. (2), referred to as a *dynamic causal law*, represents the (conditional) effect of a. It states that f is caused to be true after the execution of a in any state of the world where $p_1, \ldots, p_n$ are true. When $n = 0$ in (2), we often omit laws of this type from the description. (3) represents a *static causal law*, i.e., a relationship between fluents. It conveys that whenever the fluent literals $p_1, \ldots, p_n$ hold then so is f. For convenience, we sometimes denote the set of laws of the form (3), (2), and (1) by K, D_D, and D_E, respectively, for each action domain D.

A domain given in $\mathscr{B}$ defines a transition function from pairs of actions and states[7] to sets of states whose precise definition is given below. Intuitively, given an action a and a state s, the transition function Φ defines the set of states $\Phi(a,s)$ that may be reached after executing the action a in state s. If $\Phi(a,s)$ is an empty set it means that the execution of a in s results in an error. We now formally define Φ.

Let D be a domain in $\mathscr{B}$. A set of fluent literals is said to be *consistent* if it does not contain f and $\neg f$ for some fluent f. An *interpretation* I of the fluents in D is a maximal consistent set of fluent literals of D. A fluent f is said to be true (resp. false) in I iff $f \in I$ (resp. $\neg f \in I$). The truth value of a fluent formula in I is defined recursively over the propositional connectives in the usual way. For example, $f \wedge g$ is true in I iff f is true in I and g is true in I. We say that a formula φ holds in I (or I satisfies φ), denoted by $I \models \varphi$, if φ is true in I.

[7] *states* are defined later

Let u be a consistent set of fluent literals and K a set of static causal laws. We say that u is closed under K if for every static causal law

$$f \textbf{ if } p_1, \ldots, p_n$$

in K, if $u \models p_1 \wedge \ldots \wedge p_n$ then $u \models f$. By $Cl_K(u)$ we denote the least consistent set of literals from D that contains u and is also closed under K. It is worth noting that $Cl_K(u)$ might be undefined. For instance, if u contains both f and $\neg f$ for some fluent f, then $Cl_K(u)$ cannot contain u and be consistent; another example is that if $u = \{f, g\}$ and K contains

$$f \textbf{ if } h \qquad \text{and} \qquad \neg h \textbf{ if } f, g$$

then $Cl_K(u)$ does not exist because it has to contain both h and $\neg h$, which means that it is inconsistent.

Formally, a *state* of D is an interpretation of the fluents in $\mathbf{F}$ that is closed under the set of static causal laws K of D.

An action a is *executable* in a state s if there exists an executability proposition

$$\textbf{executable} \quad a \quad \textbf{if} \quad f_1, \ldots, f_n$$

in D such that $s \models f_1 \wedge \ldots \wedge f_n$. Clearly, if $n = 0$, then a is executable in every state of D. The *direct effect of an action a* in a state s is the set

$$e(a,s) = \{f \mid a \quad \textbf{causes} \quad f \quad \textbf{if } f_1, \ldots, f_n \in D, s \models f_1 \wedge \ldots \wedge f_n\}.$$

For a domain D, $\Phi(a,s)$, the set of states that may be reached by executing a in s, is defined as follows.

1. If a is executable in s, then
$$\Phi(a,s) = \{s' \mid s' \text{ is a state and } s' = Cl_K\left(e(a,s) \cup \left(s \cap s'\right)\right)\};$$
2. If a is not executable in s, then $\Phi(a,s) = \emptyset$.

Every domain D in $\mathscr{B}$ has a unique transition function Φ, which we call the *transition function of D*. The transition function allows one to compute the set of states reached by the execution of a sequence of actions $\alpha = [a_1, \ldots, a_n]$ from a state s_0, denoted by $\hat{\Phi}(\alpha, s_0)$, as follows:

1. If $n = 0$ then $\hat{\Phi}(\alpha, s_0) = s_0$
2. If $n > 0$ then $\hat{\Phi}(\alpha, s_0) = \cup_{u \in \Phi(a_1, s_0)} \hat{\Phi}(\alpha', u)$ where $\alpha' = [a_2, \ldots, a_n]$ and if $\hat{\Phi}(\alpha', u) = \emptyset$ for some u then $\hat{\Phi}(\alpha, s_0) = \emptyset$.

2.4 Representation and Reasoning with CPS Ontology in ASP

Various researchers have explored the relationship between ASP and the Semantic Web (e.g., Eiter (2007); Nguyen et al. (2018b;a; 2020b)), in particular with the goal of leveraging existing ontologies. In these works, an ASP program is used for reasoning about classes, properties, inheritance, relations, etc. Given ASP's non-monotonic nature, it also provides sufficient flexibility for dealing in a principled way with default values, exceptions and for reasoning about the effects of actions and change.

We use a similar approach in this paper to leverage the existing CPS Ontology for reasoning tasks related to CPS and concerns. Our approach includes the ability to query the CPS Ontology for relevant knowledge and provide it to an ASP-based reasoning component. Because the

present paper is focused on the latter, for simplicity of presentation we assume that all relevant classes, instances, relations, properties of the CPS ontology are already encoded by an ASP program. We denote this program by $\Pi(\Omega)$ where Ω denotes the ontology, which is the CPS ontology in this case. We list the predicates that will be frequently discussed in this paper.

- `class(X)`: `X` is a class;
- `subClass(X,Y)`: `X` is a subclass of `Y`;
- `aspect(I)` (resp. `concern(I)`, `prop(I)`, `decomp_func(I)`): `I` is an individual of class aspect (resp. concern, property, decomposition function);
- `subCo(I,J)`: `J` is sub-concern of `I`; and
- `addBy(C,P)`: concern *C* is addressed by property *P* (a link from a property *P* to a concern *C* in the ontology);
- `positiveImpact(P,C)`: The satisfaction of property *P* impacts positively on the satisfaction of concern *C*.
- `func(F,C)`: *F* is a functional decomposition of concern *C*.

Listing 1: $\Pi(\Omega)$:ASP program for CPS Ontology Ω

```
1  class(X)       :- RDFtriple(X,"rdf:type","owl:Class").
2  subClass(X,Y)  :- RDFtriple(X,"rdfs:subClassOf",Y), class(X), class(Y).
3  subClass(X,Y)  :- subClass(X,Z), subClass(Z,Y).
4  instance(I)    :- RDFtriple(I,"rdf:type","owl:NamedIndividual").
5  isInstanceOf(I,X) :- instance(I), class(X), RDFtriple(I,"rdf:type",X).
6  isInstanceOf(I,Y) :- instance(I), class(X), class(Y), subClass(X,Y),
       isInstanceOf(I,X).
7  concern(C)     :- instance(C), isInstanceOf(C,"cpsf:Concern").
8  aspect(A)      :- instance(A), isInstanceOf(A,"cpsf:Aspect").
9  prop(P)        :- instance(P), isInstanceOf(P,"cpsf:Property").
10 decomp_func(F) :- instance(F), isInstanceOf(F,"cpsf:DecompFunc").
11 subCo(I,J)  :- concern(I), concern(J), RDFtriple(I,"cpsf:hasSubCon",J).
12 addBy(C,P)  :- prop(P), concern(C), RDFtriple(P,"cpsf:addrConcern",C).
13 func(F,C)   :- decomp_func(F), concern(C), RDFtriple(F,"cpsf:
       decompFunctionOf",C).
14 positiveImpact(P,C) :- concern(C), prop(P), RDFtriple(P,"cpsf:
       impactPositively",C).
```

Listing 1 represents the ASP program $\Pi(\Omega)$ of CPS Ontology Ω. The predicate `RDFtriple(S,P,O)` denotes the RDF triple store which has been queried and extracted from Ω by using SPARQL[8]. Lines 1–2 define the `class(X)` and `subClass(X,Y)` based on the ontology extraction. Line 3 reasons the extension about subclass relationship. Lines 4–6 encode the definitions of `instance(I)` and `isInstanceOf(I,X)` with the similar method. The concern, aspect, property and decomposition function instances are defined in Lines 7–10. And, the three rules in Lines 11–14 represent the encoding of `subCo(I,J)`, `addBy(C,P)`, `func(F,C)` and `positiveImpact(P,C)` relationships respectively.

Given a collection of individuals in the CPS ontology Ω, $\Pi(\Omega)$ will allow us to check $addBy(c,p)$, $subCo(i,j)$, $func(f,c)$, $positiveImpact(p,c)$, etc; whether a concern c is addressed by a property p, concern j is a sub-concern of concern i, f is functional decomposition of concern c, the satisfaction of p impacts positively on concern c, etc. respectively.

[8] https://www.w3.org/TR/rdf-sparql-query/

They are written as: $\Pi(\Omega) \models addBy(c,p)$, $\Pi(\Omega) \models subCo(i,j)$, $\Pi(\Omega) \models func(f,c)$,$\Pi(\Omega) \models positiveImpact(p,c)$, etc.

Similar rules for reasoning about the inheritance between concerns, inheritance between sub-concerns and concerns, etc. are introduced whenever they are used subsequently. We note that the CPS framework does come with an informal semantics about when a concern is supposedly be satisfied. The work in Balduccini et al. (2018) provides a preliminary discussion on how the satisfaction of a concern can be determined. It does not present a formal description of the CPS system as in this paper and does not address the functional decomposition issue though.

3 CPS Theory Specification

3.1 Formal Definition

In this section, we develop a formal definition of CPS theory and its semantics. The proposed notion of a CSP theory will allow one to specify and reason about the concerns of the CPS. Our discussion will focus on `Trustworthiness` aspect in the CPS ontology but the proposed methodology is generic and is applicable to the full CPS ontology. To motivate the definition, we use the following example:

Example 2 (*Extended from Balduccini et al. (2018)*)
Consider a lane keeping/assist system (LKAS) of an advanced car that uses a camera (`CAM`) and a situational awareness module (`SAM`). The SAM processes the video stream from the camera and controls the automated navigation system through a physical output. In addition, the system also has a battery (`BAT`).

CAM and SAM may use encrypted memory (`data_encrypted`) and a secure boot (`secure_boot`). Safety mechanisms in the navigation system cause it to shut down if issues are detected in the input received from SAM. The CAM and SAM can be in one of two operational modes, the basic mode (`basic_mode` or `b_mode`) and the advanced mode (`advanced_mode` or `a_mode`). The two properties address concern `Integrity` relevant to `operation` function. In advanced mode, the component consumes much more energy than if it were in basic mode. BAT serves the system energy consumption and relates with one of three properties, `saving_mode` (`s_mode`) or `normal_mode` (`n_mode`) or `powerful_mode` (`p_mode`). Three properties address concern `Integrity` relevant to the `energy` functionality.

The relationship between `SAM`, `CAM` and `BAT` are: (**1**) If both `SAM` and `CAM` are in `advanced_mode`, the battery has to work in `saving_mode`. (**2**) if `CAM` and `SAM` are in `basic_mode`, the battery can be in `powerful_mode` or `normal_mode` and (**3**) if one of `SAM` and `CAM` is in `advanced_mode` and the other one is in `basic_mode`, then the battery must work in `normal_mode`.

The relationship between the LKAS domain and the CPS ontology is shown in Figure 2. Informally, the CPSF defines that the concern `Integrity` is *satisfied* if `secure_boot` is satisfied and its two functionalities, `operation` and `energy`, are satisfied; the `operation` functionality is satisfied if at least one of the properties {`advanced_mode`, `basic_mode`} is satisfied; and the `energy` functionality is satisfied if there is at least one of {`saving_mode`, `normal_mode`, `powerful_mode`} properties is satisfied. Intuitively, this can be represented by the following

formula:

$$(\texttt{secure\_boot}) \wedge (\texttt{advanced\_mode} \vee \texttt{basic\_mode}) \\ \wedge (\texttt{saving\_mode} \vee \texttt{normal\_mode} \vee \texttt{powerful\_mode}) \tag{4}$$

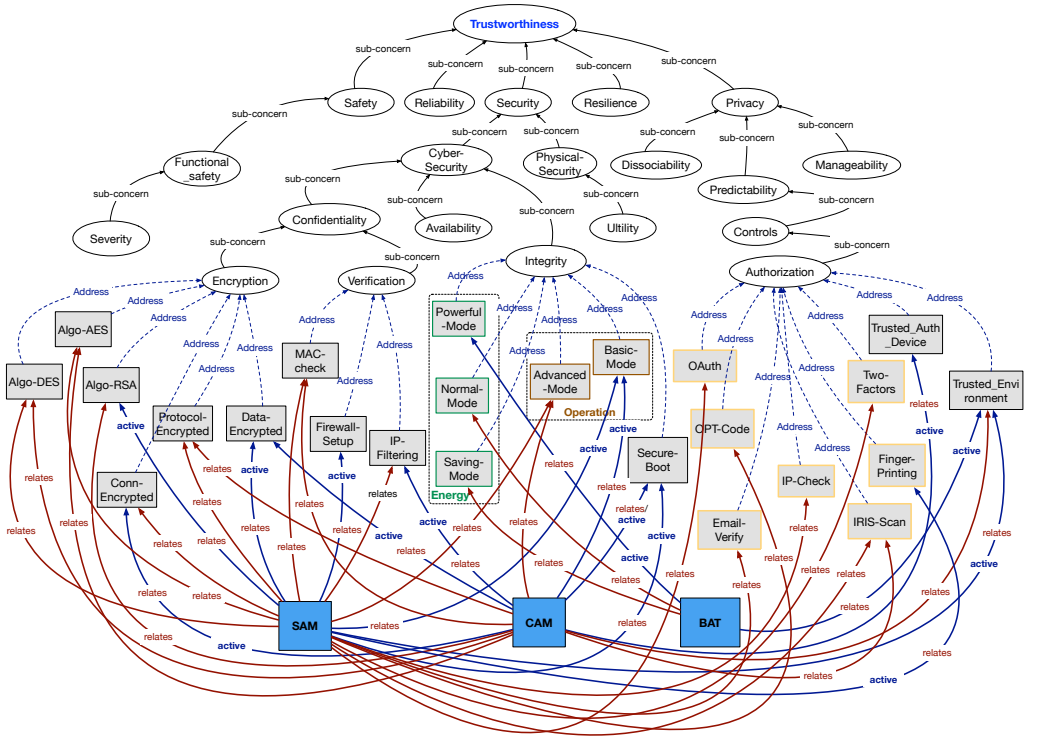

Fig. 2: CPS Ontology and LKAS domain

The example shows that a CPS system is a dynamic domain and contains different components, each associated with some properties which affect the satisfaction of concerns defined in the CPS ontology. In addition, the satisfaction of concerns depends on the truth values of formulae constructed using properties and a concern might be related to a group of properties. We will write $\omega(c)$ to denote the set of properties that *addresses* a concern c. We therefore define a CPS system as follows.

Definition 1 (*CPS System*)
A *CPS system* $\mathscr{S}$ is a tuple (CO, A, F, R, Γ) where:

- CO is a set of components;
- A is a set of actions that can be executed over $\mathscr{S}$;
- F is a finite set of fluents (or state variables) of the system;
- R is a set of relations that maps each physical component $co \in CO$ to a set of properties $R(co)$ defined in the CPS ontology; and
- Γ is a set of triples of the form (c, fu, ψ) where c is a concern, fu is a functional decomposition of concern c, and ψ is a formula constructed over $\omega(c)$.

In Definition 1, (A,F) represents the dynamic domain of $\mathscr{S}$, Γ represents constraints on the satisfaction of concerns in the CPSF ontology in $\mathscr{S}$, and R encodes the properties of components in $\mathscr{S}$ which are related to the concerns specified in the CPSF. As the truth values of these properties can be changed by actions, we assume that

$$\cup_{co\in CO} R(co) \cup \{active(co,p) \mid co \in CO, p \in R(co)\} \subseteq F.$$

where $active(co,p)$ is true means that the component co is currently active with property p. (A,F) is an action theory as described in Subsection 2.3. Note that (A,F) can be non-deterministic due to the presence of statements of the form (3). Although it is possible, this rarely happens in practical applications. We will, therefore, assume that (A,F) is deterministic throughout this paper. We illustrate Definition 1 in the following example.

Example 3
The CPS system in Example 2 can be described by $\mathscr{S}_{lkas} = (CO_{lkas}, A_{lkas}, F_{lkas}, R_{lkas}, \Gamma_{lkas})$ where:

- $CO_{lkas} = \{\texttt{SAM}, \texttt{CAM}, \texttt{BAT}\}$.
- F_{lkas} contains the following fluents:
  - `active(X,P)` denotes that component $X \in CO_{lkas}$ is working actively with property P, e.g., `active(cam,basic_mode)`, `active(cam,data_encrypted)`, `active(sam,finger_printing)` and `active(bat,normal_mode)` states that the camera is working in basic mode, with encrypted data, the SAM is authenticated by fingerprinting method and the battery is working in normal mode.
  - `on(X)` (`off(X)`) denotes that component X is (isn't) ready for use.
  - the set of properties that are related to the components (`P` denotes that the truth value of property P), e.g., `basic_mode`, `oauth`, etc. These properties are drawn in Figure 2 (rectangle boxes except the three components `SAM, CAM, BAT`).

  The relationship among the fluents are encoded below:

  - $active(BAT, saving\_mode)$ **if** $active(SAM, advanced\_mode), active(CAM, advanced\_mode)$ which encodes the statement if both `SAM` and `CAM` are in `advanced_mode`, the battery has to work in `saving_mode`.
  - $active(BAT, normal\_mode)$ **if** $active(SAM, advanced\_mode), active(CAM, basic\_mode)$ and
    $active(BAT, normal\_mode)$ **if** $active(SAM, basic\_mode), active(CAM, advanced\_mode)$
    encode the statement if one of `SAM` and `CAM` is in `advanced_mode` and the other one is in `basic_mode`, then the battery must work in `normal_mode`.
  - $active(BAT, powerful\_mode) \vee active(BAT, normal\_mode)$ **if** $active(SAM, basic\_mode), active(CAM, basic\_mode)$ which encodes the statement if both `SAM` and `CAM` are in `basic_mode`, the battery can be in `powerful_mode` or `normal_mode`.

- A_{lkas} contains the following actions:
  - `switM(X,M)`: switching the component X to a mode M. The set of the form (1) and (2) for the action that switches the `CAM` from `basic_mode` to `advanced_mode` `switM(cam,advanced_mode)` contains the following statements:

- **executable** switM(cam, advanced_mode) **if** on(cam), active(cam, basic_mode) which says that the action switM(cam, advanced_mode) can only be executed if the component CAM is on and in the basic_mode.
- switM(cam, advanced_mode) **causes** active(cam, advanced_mode), ¬active(cam, basic_mode).
  This states that if we switch the component CAM to the advanced_mode then it is in the advanced_mode and not in the basic_mode.

The statements for switM(cam, basic_mode) that switches the CAM from advanced_mode to basic_mode are similar. And the similar statements for switM(sam, basic_mode) and switM(sam, advanced_mode) which switch the component SAM to basic_mode and advanced_mode respectively.

— There are also actions that switch other components to different modes or methods. These are:

- switA(X, A): switching between authorization methods where $X = SAM$.
- switV(X, V): switching between verification methods where X can be *SAM* or *CAM*.
- switEM(X, EM): switching between encryption method where X can be *SAM* or *CAM*.
- switEA(X, EA): switching between encryption algorithms where X can be *SAM* or *CAM*.

The set of statements of the form (1) and (2) associated with these actions are similar to those associated with switM(X, M) and is omitted here for brevity.

— tOn(P) and tOff(P) denote the actions of enabling and disabling the truth value of property P, respectively. The sets of statements of the form (1) and (2) associated to each of these actions is similar. We list those associated with tOn(P) as an example:

- **executable** tOn(basic_mode) **if** ¬basic_mode: this can only be executed if the system property is not in the basic_mode.
- tOn(basic_mode) **causes** basic_mode: set the system property to basic_mode.

— patch(P) denotes action of patching some properties P with available patch software. The set of statements for action patch(P) could be:
executable patch(conn_encrypted) **if** ¬conn_encrypted, availablePatch(conn_encrypted)
patch(conn_encrypted) **causes** conn_encrypted

- R_{lkas} = {CAM ↦ {ip_filtering, algo_DES, algo_AES, algo_RSA, data_encrypted, conn_encrypted, mac_check, protocol_encrypted, secure_boot, basic_mode, advanced_mode, trusted_auth_device, trusted_environment, iris_scan}, SAM ↦ {data_encrypted, algo_RSA , algo_DES, algo_AES, protocol_encrypted, conn_encrypted, firewall_setup, mac_check, ip_filtering ,advanced_mode, basic_mode, finger_printing, two_factors, iris_scan, oauth , opt_code, email_verify , ip_check , trusted_environment , secure_boot}, BAT ↦ {powerful_mode, trusted_environment, normal_mode, saving_mode}}.
  The components and relations to the properties are illustrated by the arrow lines with "relates" labels in the bottom part of Figure 2.
- Γ_{lkas} contains the following triples (see also Figure 3):

— (integrity, operation, advanced_mode ∨ basic_mode) says the satisfaction of formula advanced_mode ∨ basic_mode addresses the concern integrity in the relevant functional decomposition operation.

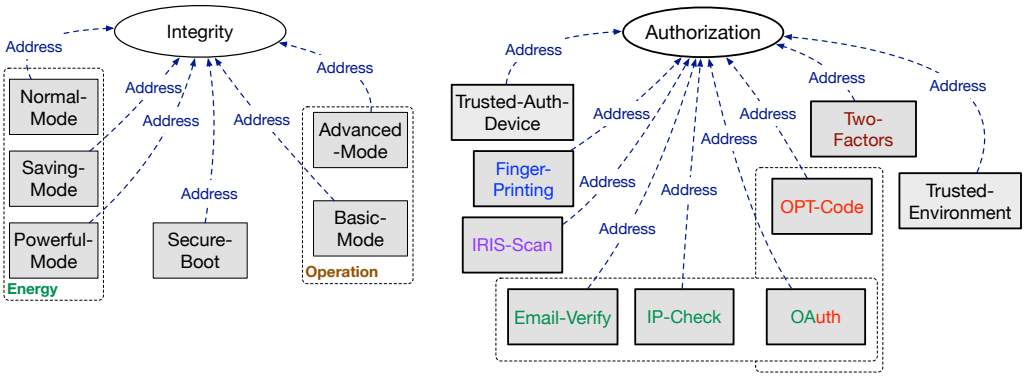

Fig. 3: `Integrity` and `Authorization` concerns with their Functionalities and Properties

— (`integrity`, `energy`, `saving_mode` ∨ `normal_mode` ∨ `powerful_mode`) denotes the formula `saving_mode` ∨ `normal_mode` ∨ `powerful_mode` addresses the concern `integrity` in the relevant functional decomposition `energy`.
— (`authorization`, `sign_in`, `oauth` ∧ `opt_code`) denotes the satisfaction of formula `oauth` ∧ `opt_code` addresses the relevant functional decomposition `sign_in` of the concern `authorization`.
— (`authorization`, `sign_in`, `two_factors` ∨ `finger_printing` ∨ `iris_scan`) denotes the formula `two_factors` ∨ `finger_printing` ∨ `iris_scan` addresses the concern `authorization` in the relevant functional decomposition `sign_in`.
— (`authorization`, `sign_in`, `oauth` ∧ `ip_check` ∧ `email_verify`) denotes that the concern `authorization` with the relevant functional decomposition `sign_in` is addressed by formula `oauth` ∧ `ip_check` ∧ `email_verify`.

In addition, the functional decomposition of the `Integrity` concern indicates that the formula (`secure_boot`) ∧ (`advanced_mode` ∨ `basic_mode`) ∧ (`saving_mode` ∨ `normal_mode` ∨ `powerful_mode`) addresses the `Integrity` concern.
Likewise, the formula

$$\begin{aligned}&\texttt{trusted\_auth\_device} \wedge \texttt{trusted\_environment} \wedge \\ &(\texttt{two\_factors} \vee \texttt{finger\_printing} \vee \texttt{iris\_scan} \vee \\ &(\texttt{oauth} \wedge \texttt{opt\_code}) \vee (\texttt{oauth} \wedge \texttt{ip\_check} \wedge \texttt{email\_verify}))\end{aligned}$$

addresses the `Authorization` concern.

Given a CPS system $\mathscr{S}$ with a set of fluents F, a *state* s of $\mathscr{S}$ is an interpretation of F that satisfies the set of static causal laws of the form (3) (Subsection 2.3).

Definition 2 (*CPS Theory*)
A *CPS theory* is a pair $(\mathscr{S}, I)$ where $\mathscr{S}$ is a CPS system and I is a state representing the *initial configuration* of $\mathscr{S}$.

3.2 The Semantics of CPS Theories

Given $(\mathscr{S}, I)$ where $\mathscr{S} = (CO, A, F, R, \Gamma)$, the action domain (A, F) specifies a transition function $\Phi_{\mathscr{S}}$ between states (Subsection 2.3). In each state, the satisfaction of a particular concern in the CPSF is evaluated using the relationship R and the components C. We will define this relation

next. First, we note that a concern in a CPS can be related to some components in $\mathscr{S}$, directly through the R relation and the formulae in Γ or indirectly through the inheritance in the CPS ontology. Observe that the development of the CPS relies on the following intuition:

- A concern might have several sub-concern;
- A concern might be addressed by a set of functional decompositions which are represented by Boolean formulae.

This leads to the following informal meaning of the notion of satisfaction of a concern in a state of the CPS:

- For each concern c, if Γ does not contain any tuple of the form (c, fu, ψ) then c is satisfied in a state s when every of its direct subconcerns is satisfied; for example, the `Trustworthiness` concern is satisfied in a state s of the LKAS system if its children, `Safety`, `Reliability`, `Security`, `Resilience`, and `Privacy`, are satisfied; and every of its properties is satisfied.
- For each concern c, if Γ contains some tuple of the form (c, fu, ψ) then c is satisfied when $\psi_c = \wedge_{(c,fu,\psi)\in\Gamma}\psi$ is satisfied in s and every property p related to c–as specified by the CPS ontology–is satisfied in s; for example, the `Integrity` concern is satisfied in the state s of the LKAS system if the formula (4) is satisfied in s where `secure_boot` is a property related to `Integrity` and the other conjuncts are the two disjunctions representing the two functional decomposition of `Integrity`.

Next, we formalize precisely the notion of satisfaction of a concern. Let $\Lambda(c)$ be the conjunction of $\wedge_{(c,fu,\psi)\in\Gamma}\psi$ and all properties that are related to c and not appearing in any formula of the form $(c, fu, \psi) \in \Gamma$. For example, in formula (4), the last two conjuncts are the two functional decompositions of `Integrity` from Γ_{lkas} and the first conjunct is a property that does not appear in any functional decomposition of `Integrity`. In the following, we denote $\langle c\rangle$ is the set of descendants of c such that for each $d \in \langle c\rangle$, d has no sub-concern.

Definition 3
Let s be a state in $\mathscr{S} = (CO, A, F, R, \Gamma)$ and c be a concern. We say that *c is satisfied in s*, denoted by $s \models c$, if

- $s \models \Lambda(c)$; and
- every sub-concern c' of c is satisfied by s.

Having defined when a concern is satisfied in a state, we can define the notion of satisfaction of a concern after the execution of a sequence of actions as follows. Recall the transition function $\Phi_{\mathscr{S}}$ dictates how the system changes from one state to another state and the set of states resulting from the execution of a sequence of actions α from a state can be computed by $\hat{\Phi}_{\mathscr{S}}$. Therefore, we can define the satisfaction of a concern c after

Definition 4
Let $(\mathscr{S}, I)$ be a CPS theory, α a sequence of actions, and c a concern in the CPS Ontology. c is *satisfied* after the execution of a sequence of actions α from the initial state I, denoted by $(\mathscr{S}, I) \models c$ **after** α, iff

$$\hat{\Phi}_{\mathscr{S}}(\alpha, I) \neq \emptyset \wedge \forall u \in \hat{\Phi}_{\mathscr{S}}(\alpha, I).[u \models c] \tag{5}$$

In the above definition, the condition $\hat{\Phi}(\alpha, I) \neq \emptyset$ guarantees that α is a valid sequence of actions, i.e., its execution in I does not fail. The second condition is the standard definition of logical entailment.

Definitions 3-4 provide the basis for us to answer questions related to the satisfaction of a concern in a state or after a sequence of actions is executed, i.e., the *concern satisfaction* problem. In the following, we will discuss other problems that are of importance for the design and development of CPS systems.

3.3 Reasoning Tasks in CPS

Knowing when a concern is (is not) satisfied is very important. We now discuss the issues related to the satisfaction of concerns in a CPS. We focus on the following problems:

1. What is the most/least trustworthy[9] component in a CPS?
2. Are there non-compliance in a given CPS? How to detect non-compliance?
3. What to do if an (external or internal) event occurs and leads to an undesirable situation? How to recover from such situation?
4. What is a best or most preferred mitigation strategy for a given situation?

In what follows, we provide precise formulations of the aforementioned tasks and propose solutions for them. For simplicity of presentation, we focus on discussing these questions with respect to a given state. The answers to these questions after the execution of a sequence of actions from the initial state can be defined similarly to the definition of the satisfaction of a concern via the function Φ, as in Definition 4. Our implementation covers both situations.

3.3.1 Most/Least Trustworthy Components

Given $\mathscr{S} = (CO, A, F, R, \Gamma)$ and a state s in $\mathscr{S}$. A component $x \in CO$ might be related to many concerns through the properties in $R(x)$, whose truth values depend on the state s. Recall that for each property p and component x, $active(x, p)$ is true in s indicates that component is active with property p in the state s; furthermore, the CPS ontology contains the specification that p *positively* or *negatively* impacts a concern c. The latter are defined by the predicates $addBy(c, p)$ and $positiveImpact(p, c)$ in Ω (Subsection 2.4). As such, when a component is active with a property, it can positively impact a concern. For example, in Figure 2 and 3, the property `secure_boot` addresses the `Integrity` concern and is described to impact positively on the satisfaction of `Integrity` concern by Ω. In the current state, the component `SAM` is working on property `secure_boot`. Assuming that concern `Integrity` is satisfied in this state, we say that component `SAM` *directly positively affects* to the `Integrity` concern through property `secure_boot`. We say that a component x *directly impacts* a concern c in state s through a property p if the following conditions hold:

1. x works with property p in state s; and
2. p addresses concern c and p is true in s.

[9] Recall that our discussion focuses on trustworthiness but it can easily be adapted to other aspects defined in the CPS ontology.

If x directly impacts c in state s through p and the CPS ontology specifies that the satisfaction of property p *impacts positively* on the satisfaction of c and c is satisfied in state s, then we say that x *directly and positively affects* c.

As the notion of concern satisfaction is propagated through the sub-concern relationship, it is natural for us to define that component x *impacts* (resp. *affects positively*) concern c through property p in a state s, denoted by $impact(x,c,s)$ (resp. $pos(x,p,c,s)$), if (i) x directly addresses (resp. direct positively affects) c through a property p; or (ii) there exists some sub-concern c' of c that is addressed (resp. positively affected) by x.

In the above example (see also Figure 2), the component `SAM` *directly positively affects* to the `Integrity` concern through property `secure_boot` then `SAM` also *affects positively* concerns `Cyber-Security`, `Security` and `Trustworthiness` in the concern tree through property `secure_boot`.

Given a component x, the ratio between the number of concerns that are positively affected by x and the number of concerns that are addressed by x characterizes how effectively x influences the system. For this reason, we will use this number to characterize the trustworthiness of components in the system. So, we define

$$tw(x,s) = \frac{\Sigma_{p\in R(x)} \mid \{c \mid s \models c \wedge positiveImpact(p,c) \wedge p \in s \wedge active(x,p)\} \mid}{\Sigma_{p\in R(x)} \mid \{c \mid (s \not\models c \vee \neg positiveImpact(p,c)) \wedge addBy(c,p) \wedge p \in s \wedge active(x,p)\} \mid + 1} \tag{6}$$

Assume that all concerns and properties are equally important, we could compare the trustworthiness of a component $x \in CO$ with that of a component $x' \in CO$ by comparing the ratios tw.

Definition 5
For a CPS system $\mathscr{S} = (CO, A, F, R, \Gamma)$, $x_1, x_2 \in CO$, and state s of $\mathscr{S}$,

- x_1 is *more trustworthy* than x_2 in s, denoted by $x_1 \succ_s x_2$ (or x_2 is *less trustworthy* than x_1, denoted by $x_2 \prec_s x_1$), if
  - $tw(x_1,s) > tw(x_2,s)$; or
  - $tw(x_1,s) = tw(x_2,s) = 0$ and $impact(x_1,s) < impact(x_2,s)$ where $impact(x,s) = \Sigma_{p\in R(x)} \mid \{c \mid (s \not\models c \vee \neg positiveImpact(p,c)) \wedge addBy(c,p) \wedge p \in s \wedge active(x,p)\} \mid$.
- x_1 is *as trustworthy as* x_2 in s, denoted by $x_1 \sim_s x_2$, if
  - $tw(x_1,s) = tw(x_2,s) > 0$; or
  - $tw(x_1,s) = tw(x_2,s) = 0$ and $impact(x_1,s) = impact(x_2,s)$.

$x_1 \succeq_s x_2$ denotes that $x_1 \succ_s x_2$ or $x_1 \sim_s x_2$. x is a most (least) trustworthy component of $\mathscr{S}$ in s if $x \succeq_s x'$ ($x' \succeq_s x$) for every $x' \in CO$.

Proposition 1
Let $\mathscr{S} = (CO, A, F, R, \Gamma)$ be a CPS system and s be a state in $\mathscr{S}$. The relation $\succeq_s$ over the components of $\mathscr{S}$ is transitive, symmetric, and total.

Proof
It is easy to see that for any pair of components, either $c_1 \succ_s c_2$, $c_2 \sim_s c_1$, or $c_1 \sim_s c_2$. Furthermore, $c \sim_s c$. It follows that $\succeq_s$ is therefore transitive, symmetric, and total. □

3.3.2 Non-compliance Detection in CPS

The design of a CPS is often subject to competing constraints from various people or organizations with different focus and type of expertise. This may result in sets of constraints that are unsatisfiable, e.g., a set of concerns cannot (never) be satisfied, giving rise to a non-compliance. Example 1 shows that there exists a situation in which competing concerns cannot be satisfied at the same time. In general, the problem is formulated as follows.

Definition 6 (*Lack of Compliance*)
Given the CPS system $\mathscr{S} = (CO, A, F, R, \Gamma)$, an integer n, a set of actions $SA \subseteq A$, and a set of concerns SC, we say that $\mathscr{S}$ is

1. *weakly n-noncompliant* wrt. (SA, SC) if there exists a sequence α of at most n actions in SA and an initial state I, such that $(\mathscr{S}, I) \not\models c$ **after** α for some concern $c \in SC$.
2. *strongly n-noncompliant* wrt. (SA, SC) if for every sequence α of at most n actions in SA and an initial state I, $(\mathscr{S}, I) \not\models c$ **after** α for some concern $c \in SC$.

Given an integer k, *weakly k-noncompliant* implies that there is a potential that some concern in the set SC of concerns might not be satisfied. *Strongly k-noncompliant* indicates that there is always some concern that cannot be satisfied. Systems that are *strongly k-noncompliant* might need to be re-designed.

It is easy to see that, by Definition 4, checking whether a system is *weakly k-noncompliant* is equivalent to identifying a plan of length k or less that "makes some concern unsatisfied." On the other hand, checking whether a system is *strongly k-noncompliant* is equivalent to identifying a plan of length less than k that "satisfies all concerns". Since we assume that the specification language for CPS is propositional and planning for bounded plans is NP-complete, we can easily derive the following results:

Proposition 2
Given $\mathscr{S}$, (SA, SC), and k, checking whether $\mathscr{S}$ is *weakly k-noncompliant* is NP-complete and checking whether $\mathscr{S}$ is *strongly k-noncompliant* is co-NP-complete.

Proof
This relies on the fact that checking whether a planning problem has a solution of length k is NP-complete (e.g., the PLAN-LENGTH problem in Ghallab et al. (2004)). □

3.3.3 Mitigation Strategies

Let $\mathscr{S} = (CO, A, F, R, \Gamma)$ be a CPS system and s be a state of $\mathscr{S}$. When some concerns are *unsatisfied* in s, we need a way to *mitigate* the issue. Since the execution of actions can change the satisfaction of concerns, the mitigation of an issue can be achieved by identifying a plan that suitably changes the state of properties related to the concerns. The mitigation problem in a CPS can be defined as follows:

Definition 7 (*Mitigation Strategy*)
Let $\mathscr{S} = (CO, A, F, R, \Gamma)$ be a CPS domain and s a state in $\mathscr{S}$. Let Σ be a set of concerns in Ω. A *mitigation strategy* addressing Σ is a plan α whose execution at the initial state s results in a state s' such that for every $c \in \Sigma$, c is satisfied in s'.

Definition 7 assumes that all plans are equal. This is often not the case in a CPS system. To illustrate this issue,

Example 4
Consider the LKAS system in Example 2. The initial state I_{lkas} is given by: `CAM` and `SAM` are in `basic_mode` and `secure_boot`, `BAT` is in `powerful_mode` and every properties in I_{lkas} are observed to be *True*. The energy consumption constraints of `BAT` are encoded in Listing 2. Figure 4 shows a fragment of the CPS theory that is related to the problem described in this example.

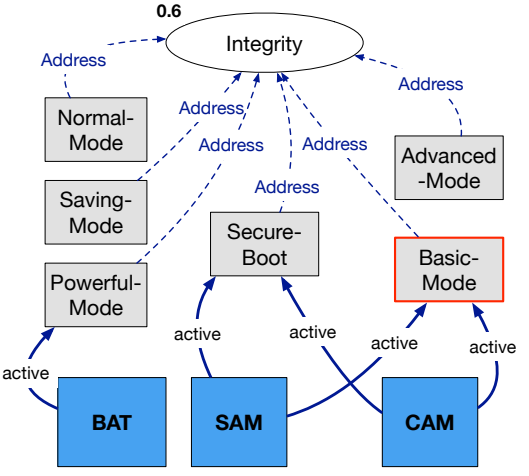

Fig. 4: Current configuration of Δ_{lkas} related to `Integrity` concern after cyber-attack

Listing 2: Π^{c}_{lkas}: Battery consumption constraints in Δ_{lkas}

```
1 h(active(bat,saving_mode),T) :- h(active(cam,advanced_mode),T),
      h(active(sam,advanced_mode),T), step(T).
2 1{h(active(bat,powerful_mode),T); h(active(bat,normal_mode),T)}1 :-
      h(active(cam,basic_mode),T), h(active(sam,basic_mode),T), step(T).
3 h(active(bat,normal_mode),T) :- h(active(X,advanced_mode),T), X!=Y,
      h(active(Y,basic_mode),T), step(T).
4 :- h(active(bat,M1),T), h(active(bat,M2),T), M1!=M2, step(T).
```

A cyber-attack occurs and the controller module is attacked, which causes `basic_mode` to become `False` while `advanced_mode` is (`True`). Given this information, we need a mitigation strategy for the set $\Sigma = \{Integrity\}$. The mitigation strategies (with the length is 2) can be generated as following:

- α_1= [`tOn(basic_mode)`]
- α_2= [`switM(cam,advanced_mode)` , `switM(sam,advanced_mode)`]
- α_3= [`switM(sam,advanced_mode)` , `switM(cam,advanced_mode)`]
- α_4= [`switM(sam,advanced_mode)` , `tOn(basic_mode)`]
- α_5= [`switM(cam,advanced_mode)` , `tOn(basic_mode)`]

As shown in the example, it is desirable to identify the *best* mitigation strategy. In this paper, we propose two alternatives. The first alternative relies on a notion called likelihood of satisfaction of concerns and the second alternative considers the uncertainty of actions.

Likelihood of Satisfaction (LoS) of Concerns We introduce a notion called *likelihood of satisfaction (LoS) of concern* and use it to distinguish mitigation strategies. Our notion relies on the positive impacts of properties on concerns within the system (Subsection 2.4). For example, property `secure_boot` positively impacts `Integrity` in Example 2 (denoted by `positiveImpact(secure_boot,integrity)`). For a concern c, we denote with $rel^{+}(c)$ the set of all properties that positively impact a concern c. Furthermore, $rel^{+}_{sat}(c,s)$ is the set of properties in $rel^{+}(c)$ which hold in state s. The ratio between these two numbers can be used to characterize the *positive impact degree* of concern c in state s as follows:

$$deg^{+}(c,s) = \begin{cases} \dfrac{\mid rel^{+}_{sat}(c,s) \mid}{\mid rel^{+}(c) \mid} & \text{if } rel^{+}(c) \neq \emptyset \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

We note that rel^{+}_{sat} and tw might appear similar but they are different in the following way: rel^{+}_{sat} is concerned with the relationship between properties and concerns while tw focuses on the relationship between components and concerns.

We define the likelihood of satisfaction of a concern as follows.

Definition 8 (*Likelihood of Concern Satisfaction*)
Given a CPS system $\mathscr{S}$, a state s in $\mathscr{S}$, and a concern c, the likelihood of the satisfaction (LoS) of c in s, denoted by $\varphi_{LoS}(c,s)$, is defined by:

$$\varphi_{LoS}(c,s) = \begin{cases} deg^{+}(c,s) * \Pi_{x \in sub(c)} \varphi_{LoS}(x,s) & \text{if } sub(c) \neq \emptyset \\ deg^{+}(c,s) & \text{if } sub(c) = \emptyset \end{cases} \tag{8}$$

where $sub(c)$ is the set of subconcerns of c.

Having defined the LoS of different concerns, we can now use this notion in comparing mitigation strategies. It is worth to mention that CPSF defines nine aspect, i.e., top-level concerns, (e.g., `trustworthiness`, `functionality`, `timing`, etc.). Let TC_{Ω} be the set of top-level concerns in the CPS ontology. We discuss two possibilities:

- *Weighted LoS*: Each top-level concern is associated with a number, i.e., each $c \in TC_{\Omega}$ is associated with a weight W_c (e.g., $W_{functionality}$ for `functionality`, $W_{trustworthy}$ for `trustworthiness`, etc.). The weights represent the importance of the top-level concerns in the CPS. They can be used to compute the weighted LoS of a system $\mathscr{S}$ in state s

  $$w(\mathscr{S},s) = \Sigma_{c \in TC_{\Omega}} \varphi_{LoS}(c,s) * W_c \tag{9}$$

  This weighted LoS can be used to define a preference relation between mitigation strategies such as $\beta \prec \alpha$ (α is better than β) iff $\max_{s' \in \Phi_{\mathscr{S}}(\alpha,s)} w(\mathscr{S},s') \geq \max_{s' \in \Phi_{\mathscr{S}}(\beta,s)} w(\mathscr{S},s')$.
- *Specified Preferences LoS*: An alternative to the weighted LoS of a system is to allow the users to specify a partial ordering over the set TC_{Ω} which will be used to define a preference relation among mitigation strategies using well-known preference aggregation strategies (e.g., lexicographic ordering). For example, if `Functionality` > `Business` then a mitigation strategy α is better than a mitigation strategy β, written as $\beta \prec \alpha$, iff $\max_{s' \in \Phi_{\mathscr{S}}(\alpha,s)} \varphi_{LoS}(\texttt{Functionality},s') \geq \max_{s' \in \Phi_{\mathscr{S}}(\beta,s)} \varphi_{LoS}(\texttt{Business},s')$.

It is easy to see that the above preference relation $\prec$ is also transitive, symmetric, and reflexive and if some strategies exist then most preferred strategies can be computed.

Example 5 (*Continuing from Example 4*)

Let us consider the strategies generated in Example 4. All five mitigation strategies ($\alpha_1, \alpha_2, \alpha_3, \alpha_4$ and α_5) generated in Section 4.4 can be used to address the issue raised by the cyber-attack. Specifically, the fragment of final state (G_{α_i}) relevant to `Integrity` concern of each plan (α_i) is given below:

- G_{α_1} is $\{$`CAM` $\mapsto$ `basic_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `basic_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `powerful_mode` $\}$ or $\{$`CAM` $\mapsto$ `basic_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `basic_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `normal_mode` $\}$.
  In which, we define $G^1_{\alpha_1}$ is $\{$`CAM` $\mapsto$ `basic_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `basic_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `powerful_mode` $\}$, and $G^2_{\alpha_1}$ is $\{$`CAM` $\mapsto$ `basic_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `basic_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `normal_mode` $\}$.
- G_{α_2} and G_{α_3}: $\{$`CAM` $\mapsto$ `advanced_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `advanced_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `saving_mode`$\}$
- G_{α_4} is $\{$`CAM` $\mapsto$ `basic_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `advanced_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `normal_mode`$\}$
- G_{α_5} is $\{$`CAM` $\mapsto$ `advanced_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `basic_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `normal_mode`$\}$

In each considered state, the statement $X \mapsto P$ denotes that component X is working with property P. For example, `BAT` $\mapsto$ `saving_mode` says that the battery is working in saving mode.

Considering the five final configurations of different mitigation strategies in the example above, we have:

$$deg^+\left(Integrity, G^1_{\alpha_1}\right) = 0.6, \quad \varphi_{LoS}\left(Integrity, G^1_{\alpha_1}\right) = 0.6\ ;$$
$$deg^+\left(Integrity, G^2_{\alpha_1}\right) = 0.4, \quad \varphi_{LoS}\left(Integrity, G^2_{\alpha_1}\right) = 0.4\ ;$$
$$deg^+\left(Integrity, G_{\alpha_2}\right) = 0.8, \quad \varphi_{LoS}\left(Integrity, G_{\alpha_2}\right) = 0.8;$$
$$deg^+\left(Integrity, G_{\alpha_3}\right) = 0.8, \quad \varphi_{LoS}\left(Integrity, G_{\alpha_3}\right) = 0.8;$$
$$deg^+\left(Integrity, G_{\alpha_4}\right) = 0.6, \quad \varphi_{LoS}\left(Integrity, G_{\alpha_4}\right) = 0.6 \text{ and}$$
$$deg^+\left(Integrity, G_{\alpha_5}\right) = 0.6, \quad \varphi_{LoS}\left(Integrity, G_{\alpha_5}\right) = 0.6$$

We also have that $deg^+\left(availability, \_\right) = 1$, $deg^+\left(security, \_\right) = 1$, $deg^+\left(trustworthiness, \_\right) = 1$, etc. In addition, we also have the LoS values of `trustworthiness` aspect in the five different final configurations as following:

$$\varphi_{LoS}\left(Trustworthiness, G^1_{\alpha_1}\right) = 0.0497,$$
$$\varphi_{LoS}\left(Trustworthiness, G^2_{\alpha_1}\right) = 0.0331,$$
$$\varphi_{LoS}\left(Trustworthiness, G_{\alpha_2}\right) = 0.0662,$$
$$\varphi_{LoS}\left(Trustworthiness, G_{\alpha_3}\right) = 0.0662,$$
$$\varphi_{LoS}\left(Trustworthiness, G_{\alpha_4}\right) = 0.0497, \text{ and}$$
$$\varphi_{LoS}\left(Trustworthiness, G_{\alpha_5}\right) = 0.0497.$$

Figure 5 shows the `trustworthiness` tree for the final configurations of mitigation strategies α_2 and α_3 (G_{α_2} and G_{α_3}), where LoS values are computed and displayed as a number at the top-left of each concern. In all 5 possible strategies, mitigation strategies α_2 and α_3 are also the best mitigation strategies which are especially relevant to the `trustworthiness` attribute, where the LoS of `trustworthiness` aspect in final state (G_{α_2} and G_{α_3}) is maximum. In this figure, the LoS of `trustworthiness` (root concern) is 0.0662 (`llh_sat(trustworthiness)=0.0662`). By applying a similar methodology for all remaining aspects (i.e., `business`, `functional`, `timing` etc.), we can calculate LoS values for all nine aspects in CPS Ontology.

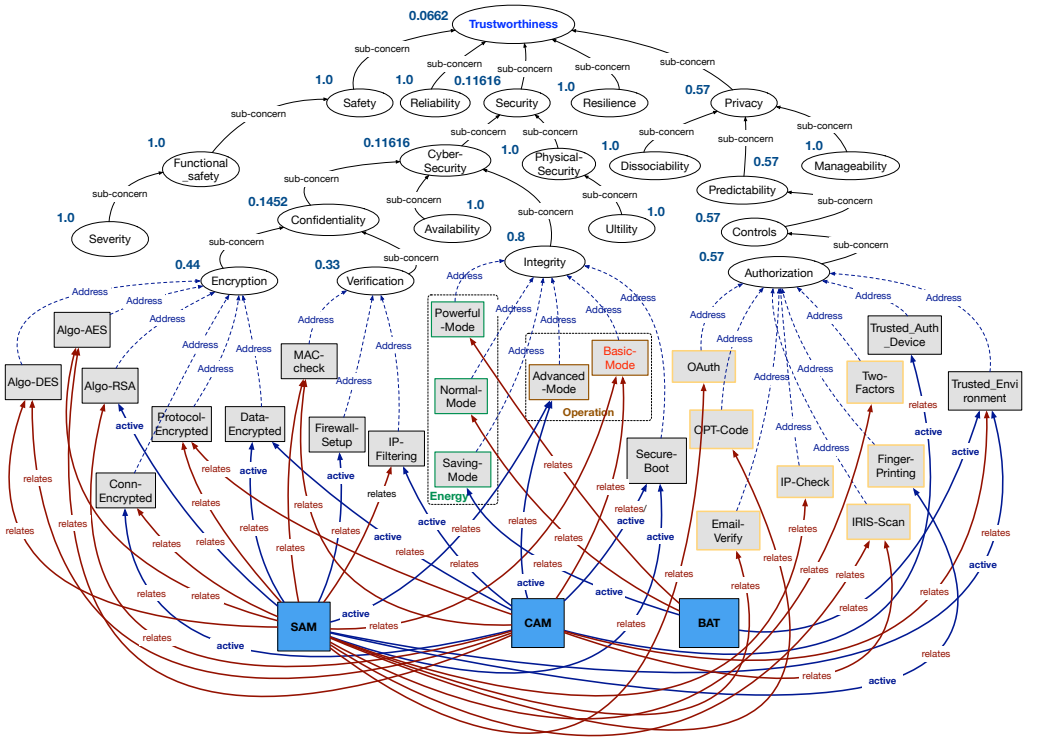

Fig. 5: Trustworthiness concern tree with LoS of concerns computation

Mitigation Strategy with The Best Chance to Succeed Preferred mitigation strategies computed using LoS of concern satisfaction assume that actions always succeeded. In practice, actions might not always succeed. In this case, it is preferable to identify strategies with the best chance of success. Assume that each action a is associated with a set of statements of the form:

$$a \quad \textbf{success\_with} \quad v \quad \textbf{if} \quad X \tag{10}$$

where $v \in [0,1]$ and X is a consistent set of literals in $\mathscr{S}$. This statement says that if each $l \in X$ is true in a state s and a is executable in s then v is the probability of a's execution in s succeeds. We assume that if a occurs in two statements "a **success_with** v_1 **if** X_1" and "a **success_with** v_2 **if** X_2" with $X_1 \neq X_2$ then $v_1 = v_2$ or there exists $p \in F$ such that $\{p, \neg p\} \subseteq X_1 \cup X_2$. Furthermore, for a state s in which no statement associated with some action a is applicable, we assume that a succeeds with probability 1 in s if it is executable in s. It is easy to see that this set of statements defines a mapping $pr : A \times States \to [0,1]$ where $States$ denotes the set of all states of $\mathscr{S}$ and $pr(a,s)$ represents the probability that the execution of a in s succeeds. Thus, the execution of a sequence of actions (or a strategy) $\alpha = [a_0, \ldots, a_{n-1}]$ in a state s succeeds with the probability $\Pi_{i=0}^{n-1} pr(a_i, s_i)$ where $s_0 = s$, and for $i > 0$, s_i is the result of the execution of a_{i-1} in s_{i-1}. This can be used to define a preference relation between strategies similar to the use of LoS of concern satisfaction, i.e., we prefer strategies whose probability of success is maximal. We omit the formal definition here for brevity.

It is worth mentioning that the specification by statements of the form (10) is at the action level. It is assumed that if action a succeeds with a probability v, it means that all of its potential effects will be achieved with the probability v. In some applications, it might be more proper to

consider a finer level of probabilistic specification of effects such as if action a succeeds then with a probability p_i, e_i will be true, for $i = 1,\ldots,k$. To work with this type of applications, a probabilistic action language such as the one proposed in Baral et al. (2002) or a specification using Markov decision process could be used. We will leave the discussion related to this type of applications for the future.

4 An ASP-Based Implementation for Reasoning Tasks in CPS Theories

This section develops an ASP encoding given a CPS theory, building on the work on planning in ASP and on formalizing CPS (e.g., Gelfond and Lifschitz (1993); Balduccini et al. (2018)). The code is available at `https://github.com/thanhnh-infinity/Research_CPS`. We start with the encoding of the theory (Subsection 4.1). Afterwards, we develop, for each reasoning task, an ASP module (Subsections 4.2–4.7) which, when added to the encoding of the domain, will compute the answers for the task.

Throughout this section, we assume that $(\mathscr{S},I)$ where $\mathscr{S} = (CO,A,F,R,\Gamma)$ is a CPS. The encoding of $(\mathscr{S},I)$ in ASP will be denoted with $\Pi(\mathscr{S})^n$, where n is a non-negative integer representing the horizon of the system that we are interested in. We note that the encoding of the CPS ontology (Subsection 2.1 and 2.4), $\Pi(\Omega)$, will be automatically added to any program developed in this section. For this reason, whenever we write $\Pi(\mathscr{S})^n$ we mean $\Pi(\mathscr{S})^n \cup \Pi(\Omega)$.

4.1 ASP Encoding of a CPS Theory

The encoding of a CPS theory contains two parts, one encodes the domain and another the initial state. We first discuss the encoding of the domain.

4.1.1 Encoding of the Domain $\mathscr{S}$

$\Pi(\mathscr{S})^n$ contains the following rules[10].

- *The set of rules declaring the time steps:* for each $0 \leq t \leq n$, an atom $step(t)$, i.e., the rule $step(t) \leftarrow$.
- *The set of rules encoding the components:* for each $co \in CO$, an atom $comp(co)$.
- *The set of rules encoding actions:* for each $a \in A$, an atom $action(a)$.
- *The set of rules encoding fluents:* for each $f \in F$, an atom $fluent(f)$.
- *The set of rules encoding relations:* for each $co \in CO$ and $p \in R(co)$, an atom $relation(co,p)$.
- *The set of rules encoding functional dependencies:* for each $(c,fu,\varphi) \in \Gamma$, an atom $formula(id_\varphi)$, an atom $addFun(c,fu,id_\varphi)$, and a set of atoms encoding φ, where id_φ is a unique identifier associated to φ and c is a concern.
- *The rules for reasoning about actions and changes (see, e.g., Son et al. (2006))*:
  - For each executability condition of the form (1) the rule:
    $\texttt{exec(a,T)} :- \texttt{step(T)}, \texttt{h}^*\texttt{(p}_1\texttt{,T)}, \ldots, \texttt{h}^*\texttt{(p}_n\texttt{,T)}.$

[10] We follow the convention in logic programming and use strings starting with lower/uppercase letter to denote constants/variables. In addition, this program can be generated automatically given that $\mathscr{S}$ is specified in the syntax given in Section 3.

— For each dynamic causal law of the form (2):
$\mathtt{h}^*(\mathtt{f},\mathtt{T+1}) :- \mathtt{step}(\mathtt{T}), \mathtt{occurs}(\mathtt{a},\mathtt{T}), \mathtt{h}^*(\mathtt{p}_1,\mathtt{T}),\ldots,\mathtt{h}^*(\mathtt{p}_n,\mathtt{T}).$
— For each state constraint of the form (3):
$\mathtt{h}^*(\mathtt{f},\mathtt{T}) :- \mathtt{step}(\mathtt{T}), \mathtt{h}^*(\mathtt{p}_1,\mathtt{T}),\ldots,\mathtt{h}^*(\mathtt{p}_n,\mathtt{T}).$
— The rules encoding the inertia axiom:
$\mathtt{h}(\mathtt{f},\mathtt{T+1}) :- \mathtt{step}(\mathtt{T}),\mathtt{h}(\mathtt{f},\mathtt{T}),\mathtt{not}\ \neg\mathtt{h}(\mathtt{f},\mathtt{T+1}).$
$\neg\mathtt{h}(\mathtt{f},\mathtt{T+1}) :- \mathtt{step}(\mathtt{T}),\neg\mathtt{h}(\mathtt{f},\mathtt{T}),\mathtt{not}\ \mathtt{h}(\mathtt{f},\mathtt{T+1}).$

where $\mathtt{h}^*(\mathtt{x},\mathtt{T})$ stands for $\mathtt{h}(\mathtt{x},\mathtt{T})$ if $x \in F$ is a fluent and $\neg\mathtt{h}(\mathtt{y},\mathtt{T})$ if $x = \neg y$ and $y \in F$.

We illustrate the ASP encoding of a CPS by presenting the encoding of the LKAS theory in Example 2. Listing 3 shows the encoding of components, actions, and relations of $\mathscr{S}_{lkas}$ without the encoding of the initial state. Listing 4 shows the ASP encoding for Γ_{lkas} (see Figure 3).

Listing 3: Example program $\Pi(\mathscr{S}_{lkas})^n$ for LKAS

```
comp(sam). comp(cam). comp(bat).
relation(cam,algo_AES). relation(cam,algo_RSA).
relation(cam,algo_DES). relation(cam,ip_filtering).
relation(cam,conn_encrypted). relation(cam,data_encrypted).
relation(cam,protocol_encrypted). relation(cam,mac_check).
relation(cam,secure_boot). relation(cam,iris_scan).
relation(cam,advanced_mode). relation(cam,basic_mode).
relation(cam,trusted_auth_device). relation(cam,trusted_environment).
relation(sam,algo_AES). relation(sam,algo_RSA).
relation(sam,algo_DES). relation(sam,mac_check).
relation(sam,conn_encrypted). relation(sam,data_encrypted).
relation(sam,ip_filtering). relation(sam,secure_boot).
relation(sam,protocol_encrypted). relation(sam,firewall_setup).
relation(sam,advanced_mode). relation(sam,basic_mode).
relation(sam,finger_printing). relation(sam,two_factors).
relation(sam,iris_scan). relation(sam,oauth).
relation(sam,opt_code). relation(sam,email_verify).
relation(sam,ip_check). relation(sam,trusted_environment).
relation(bat,powerful_mode). relation(bat,normal_mode).
relation(bat,saving_mode). relation(bat,trusted_environment).
...
action(tOn(X))  :- prop(X). action(tOff(X)) :- prop(X).
exec(tOn(X),T)  :- ¬h(X,T), prop(X), step(T).
exec(tOff(X),T) :- h(X,T), prop(X), step(T).
h(X,T+1)  :- occurs(tOn(X),T), step(T).
¬h(X,T+1) :- occurs(tOff(X),T), step(T).
action(patch(X)):- prop(X).
exec(patch(X),T):- prop(X), availablePatch(X), ¬h(X,T), step(T).
h(X,T+1)        :- occurs(patch(X),T), step(T).
...
action(switM(cam,basic_mode)). action(switM(cam,advanced_mode)).
action(switM(sam,basic_mode)). action(switM(sam,advanced_mode)).
action(switM(bat,saving_mode)). action(switM(bat,normal_mode)).
action(switM(bat,powerful_mode)).
exec(switM(X,basic_mode),T) :- relation(X,basic_mode),
    not h(active(X,basic_mode),T), comp(X), h(basic_mode,T), step(T).
h(active(X,basic_mode),T+1) :- occurs(switM(X,basic_mode),T), step(T).
¬h(active(X,advanced_mode),T+1) :- occurs(switM(X,basic_mode),T),
```

```
39      h(active(X,advanced_mode),T), step(T).
40  exec(switM(X,advanced_mode),T) :-  comp(X), relation(X,advanced_mode),
41      not h(active(X,advanced_mode),T), h(advanced_mode,T), step(T).
42  h(active(X,advanced_mode),T+1) :- occurs(switM(X,advanced_mode),T),
        step(T).
43  ¬h(active(X,basic_mode),T+1):- step(T), h(active(X,basic_mode),T),
        occurs(switM(X,advanced_mode),T).
44  ...
```

In Listing 3, Line 1 encodes the components; Lines 2–20 encode the relations; Lines 22–29 encode the actions `tOn` and `tOff`. The remaining lines of code encode other actions in similar fashion.

Each *formula* φ related to a concern c is associated with a unique identifier φ^I and is converted into a CNF $\varphi_1 \wedge \ldots \wedge \varphi_k$, each φ_i will be associated with a unique identifier φ_i^I. The set of identifiers are declared using the predicate `formula/1`. It will be declared as `disjunction` or `conjunction`. Furthermore, set notation is used to encode a disjunction or conjunction, i.e., the predicate `member(X,G)` states that the formulae X is a member of a disjunction or a conjunction G. The predicate `func(F,C)` states that F is the functional decomposition of concern C.

Listing 4: A part of ASP program $\Pi(\mathscr{S}_{lkas})^n$ encoding Γ_{lkas} for `Integrity` and `Authorization` concerns

```
1   formula(0..3).
2   ...
3   concern(integrity).
4   conjunction(0). addConcern(integrity,0).
5   member(secure_boot,0). member(energy_func,0).
6   member(operation_func,0).
7   func(operation_func,integrity). func(energy_func,integrity).
8   disjunction(operation_func). formula(operation_func).
9   member(advanced_mode,operation_func).
10  member(basic_mode,operation_func).
11  disjunction(energy_func). formula(energy_func).
12  member(powerful_mode,energy_func). member(normal_mode,energy_func).
13  member(saving_mode,energy_func).
14  ...
15  concern(authorization).
16  conjunction(1). addConcern(authorization,1).
17  member(trusted_auth_device,1).
18  member(trusted_environment,1).
19  member(sign_in_func,1).
20  func(sign_in_func,authorization).
21  disjunction(sign_in_func).
22  formula(sign_in_func).
23  member(finger_printing,sign_in_func).
24  member(iris_scan,sign_in_func).
25  member(two_factors,sign_in_func).
26  member(2,sign_in_func). member(3,sign_in_func).
27  conjunction(2).
28  member(oauth,2). member(opt_code,2).
29  conjunction(3).
30  member(oauth,3). member(ip_check,3). member(email_verify,3).
31  ...
```

In Listing 4, the first line uses a special syntax, a short hand, declaring four atoms `formula(0),...,formula(3)`. The declaration and encoding of the `Integrity` concern and its related formulas, properties and decomposition functions are presented in Lines 3–13. In which, line 3 declares the concern `Integrity`. Lines 4–6 encode the conjunctive formula (`conjunction(0)`) that addresses the `Integrity` concern and its membership (e.g., the property `secure_boot` and the two decomposition functions of the `Integrity` concern). Line 7 specifies the two functional dependencies of the `Integrity` concern which are `operation_func` and `energy_func`. Lines 8–13 specify how the formulae address the functional decompositions. Lines 8–10 declare the disjunctive formula `operation_func` and define the membership between properties and this formula (e.g., `member(advanced_mode,operation_func)`, `member(basic_mode,operation_func)` says that `advanced_mode` and `basic_mode` are elements of the disjunction `operation_func`). Similar encoding is applied for disjunctive formulae `energy_func` in Lines 11–13. Lines 15–30 encode information related to the `Authorization` concern.

4.1.2 Encoding of the Initial State

The encoding of the initial state I of a CPS theory $(\mathscr{S},I)$, denoted by $\Pi(I)$, contains, for each fluent f, $h(f,0)$ if f is true in I or $\neg h(f,0)$ if f is false in I. Listing 5 shows a snippet of the initial state of $\mathscr{S}_{lkas}$ with Lines 1–7 specifying the true/false properties and Lines 9–17 the specific information about which components operate in which properties in LKAS in the initial state.

Listing 5: An example for a part of initial configuration of $\Pi(I_{lkas})$

```
1  h(finger_printing,0). h(oauth,0). h(ip_check,0).
2  h(two_factors,0). h(opt_code,0).
3  h(trusted_auth_device,0). h(trusted_environment,0). h(secure_boot,0).
4  h(powerful_mode,0). h(saving_mode,0). h(normal_mode,0).
5  h(basic_mode,0). h(advanced_mode,0).
6  ...
7  ¬h(iris_scan,0). ¬h(email_verify,0). ¬h(firewall_setup,0).
8  ...
9  h(active(sam,secure_boot),0). h(active(sam,algo_RSA),0).
10 h(active(sam,basic_mode),0). h(active(sam,data_encrypted),0).
11 h(active(sam,firewall_setup),0). h(active(sam,finger_printing),0).
12 h(active(sam,trusted_environment),0).
13 h(active(cam,ip_filtering),0). h(active(cam,data_encrypted),0).
14 h(active(cam,conn_encrypted),0). h(active(cam,secure_boot),0).
15 h(active(cam,trusted_auth_device),0). h(active(cam,basic_mode),0).
16 h(active(bat,powerful_mode),0). h(active(bat,trusted_environment),0).
17 ...
```

The following property (see, Son et al. (2006)) will be important for our discussion. It shows that $\Pi(\mathscr{S})^n$ correctly computes the function $\Phi_{\mathscr{S}}$.

Proposition 3

Let s be a state in $\mathscr{S}$. Let $\Pi = \Pi(\mathscr{S})^1 \cup \{h^*(f,0) \mid f \in s\}$. Assume that a is an action that is executable in s. Then, $s' \in \Phi_{\mathscr{S}}(a,s)$ iff there exists an answer set S of $\Pi \cup \{occurs(a,0)\}$ such that $\{h^*(f,1) \mid f \in s'\} \subseteq A$.

It is worth mentioning that $\Pi(\mathscr{S})^n$ allows us to reason about effects of actions in the following sense: assume that $[a_0,\ldots,a_{n-1}]$ is a sequence of actions, then $\Pi(\mathscr{S})^n \cup \{occurs(a_i,i) \mid i = 0,\ldots,n-1\}$ has an answer set S if and only if (*i*) a_0 is executable in the state I; (*ii*) for each $i > 0$, a_i is executable after the execution of the sequence $[a_0,\ldots,a_{i-1}]$; (*iii*) for each i, the set $\{f \mid f \in F, h(f,i) \in S\} \cup \{\neg f \mid f \in F, \neg h(f,i) \in S\}$ is a state of $\mathscr{S}$.

4.2 Computing Satisfaction of Concerns

We will next present a set of ASP rules for reasoning about the satisfaction of concerns as specified in Definitions 3–4. Since a concern is satisfied if *all* of its functional decompositions and properties are satisfied, we define rules for computing the predicate `h(sat(C),T)` which states that concern *C* is satisfied at the step `T`. The rules are given in Listing 6.

Listing 6: Π_{sat} : Concern Satisfaction Reasoning in Ω

```
1  formula(¬G) :- formula(G).
2  prop(¬G)    :- prop(G).
3  h(¬F,T):- step(T), 1{formula(F);prop(F)}, ¬h(F,T).
4  h(F,T) :- step(T), formula(F), disjunction(F), member(G,F), h(G,T).
5  ¬h(F,T):- step(T), formula(F), disjunction(F), not h(F,T).
6  ¬h(F,T):- step(T), 1{formula(G);prop(G)}, formula(F), conjunction(F),
       member(G,F), not h(G,T).
7  h(F,T) :- step(T), formula(F), conjunction(F), not ¬h(F,T).
8  ¬h(sat(C),T) :- concern(C), addConcern(C,F), not h(F,T), step(T).
9  ¬h(sat(X),T) :- subCo(X,Y), not h(sat(Y),T), concern(X), concern(Y),
       step(T).
10 ¬h(sat(X),T) :- subCo(X,Y), ¬h(sat(Y),T), concern(X), concern(Y),
       step(T).
11 h(sat(C),T) :- not ¬h(sat(C),T), concern(C), step(T).
```

The first two lines declare that the negation of a formula or a property is also a formula and thus can be a member of a disjunction or conjunction. The rule on Line 3 says that `h(¬F,T)` is true if the negation of `F` is true. This rule uses a special syntax `1{formula(F);prop(F)}` which says that there exists at least one `F` is both a formula and a property. The rule on Line 4 states that `h(F,T)` is true if `F` is a disjunction and one of its disjuncts is true. The next rule (Line 5) states that `¬h(F,T)` for a disjunction `F` is true if it cannot be proven that `F` is true. This rule applies the well-known negation-as-failure operator in establishing the truth value of `¬h(F,T)`. Similarly, the next two rules establish the truth value of a conjunction `F`, i.e., `h(F,T)` is true if none of its conjuncts is false. The remaining rules are used to establish the truth value of `h(sat(C),T)`, the satisfaction of concern `C` at step `T`. Line 8 states that if the formula addressing the concern `C` cannot be proven to be true then the concern is not satisfied. Rules in line 9-10 propagate the unsatisfaction of a concern from its subconcerns. Finally, a concern is satisfied if it cannot be proven to be unsatisfied (Line 11). We can prove the following proposition that relates the implementation and Definition 3.

Proposition 4 (*Concern Satisfaction*)
For a CPS theory $\Delta = (\mathscr{S}, I)$ and a concern c, c is satisfied (or unsatisfied) in I if $h(sat(c),0)$ (or $\neg h(sat(c),0)$) belongs to every answer set of $\Pi(\Delta)$, where $\Pi(\Delta) = \Pi(\mathscr{S})^0 \cup \Pi(I) \cup \Pi_{sat}$.

Proof
It is easy to see that for any formula φ over the fluents in $\mathscr{S}$, the encoding and the rules encoding a formula, and the rules in Lines 1–7, $I \models \Lambda(c)$ iff $\mathtt{h}(sat(\Lambda(c)^I),0)$ belongs to every answer set of $\Pi(\Delta)$ where $\Lambda(c)^I$ is the identifier associated to the formula $\Lambda(c)$. Lines 9–10 show that if c has a sub-concern that is not satisfied then it is not satisfied and hence Rule 11 cannot be applied. As such, we have that $h(sat(c),0)$ is in an answer set of $\Pi(\Delta)$ iff the formula $\Lambda(c)$ is true and all sub-concerns of c are satisfied in that answer set iff c is satisfied in I. □

Since we will be working with the satisfaction of concerns in the following sections, we will therefore need to include Π_{sat} in $\Pi(\mathscr{S})^n$. From now on, whenever we refer to $\Pi(\mathscr{S})^n$, we mean $\Pi(\mathscr{S})^n \cup \Pi(I) \cup \Pi_{sat}$.

4.3 Computing Most/Least Trustworthy Components

Proposition 1 shows that $\succeq_s$ has min/maximal elements, i.e., least/most trustworthy components of a system always exist. The program $\Pi_{mlt}(\mathscr{S})$ for computing these components is listed below.

Listing 7: Π_{mlt}: Computing Most/Least Trustworthy Components

```
1  r(X,P,C,T) :- comp(X), prop(P), concern(C), step(T), h(active(X,P),T),
       h(P,T), addBy(C,P).
2  pos(X,P,C,T) :- r(X,P,C,T), positiveImpact(P,C), h(sat(C),T), step(T).
3  nPos(X,P,C,T):- r(X,P,C,T), not positiveImpact(P,C), step(T).
4  nPos(X,P,C,T):- r(X,P,C,T), not h(sat(C),T), step(T).
5  pos(X,P,C,T) :- pos(X,P,C1,T), subCo(C,C1), step(T).
6  nPos(X,P,C,T):- nPos(X,P,C1,T), subCo(C,C1), step(T).
7  twcp(X,TW,T) :- TW=#count{C,P:pos(X,P,C,T), prop(P), concern(C)},
       comp(X), step(T).
8  twcn(X,TW,T) :- TW=#count{C,P:nPos(X,P,C,T), prop(P), concern(C)},
       comp(X), step(T).
9  higher(X1,X2,T) :- twcp(X1,TWp1,T), twcp(X2,TWp2,T), twcn(X1,TWn1,T),
       twcn(X2,TWn2,T), d1=TWp1/(TWn1 + 1), d2=TWp2/(TWn2 + 1), d1 > d2,
       step(T), TWp1!=0, TWp2!=0.
10 higher(X1,X2,T):- step(T), twcp(X1,0,T), twcp(X2,0,T), twcn(X1,TWn1,T),
       twcn(X2,TWn2,T), TWn1 < TWn2.
11 equal(X1,X2,T) :- twcp(X1,TWp1,T), twcp(X2,TWp2,T), twcn(X1,TWn1,T),
       twcn(X2,TWn2,T), d1=TWp1/(TWn1 + 1), d2=TWp2/(TWn2 + 1), d1 = d2,
       step(T), TWp1!=0, TWp2!=0.
12 equal(X1,X2,T) :- step(T), twcp(X1,0,T), twcp(X2,0,T), twcn(X1,TWn1,T),
       twcn(X2,TWn2,T), TWn1=TWn2.
13 not_highestTW(X2,T) :- comp(X1), comp(X2), higher(X1,X2,T), step(T).
14 not_lowestTW(X1,T)  :- comp(X1), comp(X2), higher(X1,X2,T), step(T).
15 most(X,T)  :- comp(X), not not_highestTW(X,T), step(T).
16 least(X,T) :- comp(X), not not_lowestTW(X,T), step(T).
```

In Listing 7, `addBy(C,P)` and `positiveImpact(P,C)` are defined in the program $\Pi(\Omega)$ (Subsection 2.4). `addBy(C,P)` is true means that a property P addresses a concern C. `positiveImpact(P,C)` is true means that the satisfaction of property P impacts positively on the satisfaction of concern C. The predicate `r(X,P,C,T)` (Line 1) encodes the relationship between X, P and C at the time T which says that the component X is working with the property P at time T and P addresses concern C. The second rule (Line 2) defines the predicate `pos(X,P,C,T)` that encodes the positive affected relationship between component X and concern C at time step T through property P which is true if the concern C is satisfied

and `positiveImpact(P,C)` and `r(X,P,C,T)` hold. Lines 3–4 define `nPos(X,P,C,T)`, which holds at time T if `r(X,P,C,T)` holds but either `positiveImpact(P,C)` is not defined in Ω or concern C is not satisfied. This element is used for the computation of the denominator of Equation (6). The rest of the listing defines the relationship `higher` between components encoding the $\succeq_T$ where T represents the state at the time T of the system and identifying the most and least trustworthy components. Lines 5–6 propagate the *positive affected* and *impact* relations (`pos/4`, `nPos/4`) of a concern from its subconcerns. $twcp(x,tw,t)$ (resp. $twcn(x,tw,t)$) encodes the number of concerns positively affected (resp. impacted) by component x at step t. The atom $\#count\{C,P : pos(X,C,P,T), prop(P), concern(C)\}$ is an aggregate atom in ASP and encodes the cardinality of the set of all concerns positively impacted by P and X.

We can show that the following proposition holds.

Proposition 5
For a CPS theory $\Delta = (\mathscr{S}, I)$ and an answer set S of program $\Pi(\mathscr{S})^n \cup \Pi(I) \cup \Pi_{mlt}$, if $most(x,t) \in S$ (resp. $least(x,t) \in S$) then x is a most (resp. least) trustworthy component in the state s_t.

The proof follows immediately from the definition of the predicate *addBy*, *positiveImpact* and the definition of aggregate functions in ASP. As such, to identify the most trustworthy component of $\mathscr{S}$, we only need to compute an answer set S of $\Pi(\mathscr{S})^n \cup \Pi(I) \cup \Pi_{mlt}$ and use Proposition 5.

Example 6
Consider the $\mathscr{S}_{lkas}$ domain.

- Let us consider the initial configuration I^1_{lkas} of LKAS system where every properties are observed to be true. For $\Delta_{lkas} = (\mathscr{S}_{lkas}, I^1_{lkas})$, we can easily see that (from Figure 2) the atoms: $pos(cam, advanced\_mode, integrity, 0)$, $pos(cam, secure\_boot, cyber\_security, 0)$, etc. belong to every answer set of $\Pi(\Delta_{lkas}) = \Pi(\mathscr{S}_{lkas})^n \cup \Pi(I^1_{lkas}) \cup \Pi^{lkas}_{mlt}$. Similar atoms are present to record the number of concerns affected by different properties. Furthermore, $twcp(cam, 28, 0)$, $twcn(cam, 6, 0)$, $twcp(sam, 40, 0)$, $twcn(sam, 0, 0)$, $twcp(bat, 6, 0)$ and $twcn(bat, 5, 0)$ belong to any answer set of $\Pi(\mathscr{S}_{lkas})^n \cup \Pi(I^1_{lkas}) \cup \Pi^{lkas}_{mlt}$: *SAM* is the most trustworthy component; *BAT* is the least trustworthy components at step 0.
- Now, let us consider I^2_{lkas} of LKAS system (Figure 2) where there are two properties that are observed to be *False*: `Firewall-Setup` and `Trusted-Auth-Device`. For $\Delta_{lkas} = (\mathscr{S}_{lkas}, I^2_{lkas})$, the computation of the program $\Pi(\mathscr{S}_{lkas})^n \cup \Pi(I^2_{lkas}) \cup \Pi^{lkas}_{mlt}$ shows us: $twcp(cam, 22, 0)$, $twcn(cam, 6, 0)$, $twcp(sam, 22, 0)$, $twcn(sam, 12, 0)$, $twcp(bat, 0, 0)$ and $twcn(bat, 11, 0)$ belong to any answer set of $\Pi(\mathscr{S}_{lkas})^n \cup \Pi(I^2_{lkas}) \cup \Pi^{lkas}_{mlt}$. In this situation, *CAM* is the most trustworthy component; *BAT* is the least trustworthy components at step 0.

We conclude this part with a brief discussion on possible definitions of $\succeq$. The proposed definition assumes everything being equal (e.g. all concerns and properties are equally important, the roles of every components in a CPS system are equal, etc.). In practice, the ordering $\succeq$ might be qualitative and user-dependent, e.g., an user might prefer confidentiality over integrity. $\succeq$ can be defined over a qualitative ordering and implemented in ASP in a similar fashion that preferences have been implemented (e.g., Gelfond and Son (1998)).

4.4 Computing Mitigation Strategies

The program $\Pi(\mathscr{S})^n \cup \Pi_{sat}$ can be for computing a mitigation strategy by adding the rules shown in Listing 8:

Listing 8: Π^n_{plan}: Generating Plan

```
1 1{occurs(A,T):action(A)}1 :- step(T), T<n.
2 :- occurs(A,T), not exec(A,T).
3 :- not h(sat(c), n).
```

The first rule containing the atom $1\{\texttt{occurs}(\texttt{A},\texttt{T}) : \texttt{action}(\texttt{A})\}1$ — a choice atom — is intuitively used to generate the action occurrences and says that at any step T, exactly one action must occur. The second rule states that an action can only occur if it is executable. The last rule helps enforce that $h(sat(c),n)$ must be true in the last state, at step n. For a set of concerns Σ, let $\Pi^n_{plan}[\Sigma]$ be the program obtained from Π^n_{plan} by replacing its last rule with the set $\{:-\texttt{not}\quad \texttt{h}(\texttt{sat}(\texttt{c}),\texttt{n}).\mid \texttt{c} \in \Sigma\}$. Based on the results in answer set planning, we can show:

Proposition 6
Let $\Delta = (\mathscr{S}, I)$ be a CPS theory and Σ be a set of concerns in Ω. Then, $[a_0,\ldots,a_{n-1}]$ is a mitigation strategy for Σ iff $\Pi(\Delta) \cup \Pi^n_{plan}[\Sigma]$ has an answer set S such that $occurs(a_i,i) \in S$ for every $i = 0,\ldots,n-1$.

The proof of this proposition relies on the properties of $\Pi(\Delta)$ discussed in previous section and the set of constraints in $\Pi^n_{plan}[\Sigma]$.

4.5 Non-compliance Detection in CPS Systems

The program $\Pi(\mathscr{S})^n \cup \Pi_{sat}$ can be used in non-compliance detection by adding the rules shown in Listing 9:

Listing 9: $\Pi^n(SA,SC)$: Non-compliance Detection

```
1 1{occurs(A,T):sa_action(A)}1 :- step(T), T<n, not conflict(T).
2 :- occurs(A,T), not exec(A,T), step(T).
3 1{h(F,0); ¬h(F,0)}1 :- fluent(F).
4 conflict(T)   :- sc_concern(C), ¬h(sat(C),T), step(T).
5 conflict(T+1) :- conflict(T), step(T).
6 :- not conflict(n).
```

The first two rules are similar to the rules for the planning program, with the exception that the action selection focuses on the actions in the set *SA*. The third rule generates an arbitrary initial state. The rules 4-5 state that if some concern in *SC* is not satisfied at time T then a conflict arises and the constraint on the last rule says that we would like to create a conflict at step n.

We assume that actions in *SA* are specified by atoms of the form $sa\_action(a)$ and concerns in *SC* are specified by atoms of the form $sc\_concern(c)$. It is easy to see that an answer set S of $\Pi(\mathscr{S})^n \cup \Pi_{sat} \cup \Pi^n(SA,SC)$ represents a situation in which the system will eventually not satisfy some concern in *SC*. Specifically, if the sequence of actions $[a_0,\ldots,a_t]$ such that $occurs(a_i,i) \in S$ and, for $s > t$, there exists no $occurs(a_s,s) \in S$, is executed in the initial state (the set $\{f \mid h(f,0) \in S, f \in F\} \cup \{\neg f \mid \neg h(f,0) \in S, f \in F\}$) then some concern in *SC* will not be satisfied after n steps. In other words, to check whether $\mathscr{S}$ is weakly n-noncompliant, we only need to check whether $\pi_n = \Pi(\mathscr{S})^n \cup \Pi_{sat} \cup \Pi^n(SA,SC)$ as an answer set of not. The proof of this property relies on the definition of an answer set for a program with constraints, which say that the constraint :-

`not conflict(n).` must be false in the answer set, which in turn implies that `conflict(n)` must be true.

If $\mathscr{S}$ is weakly n-noncompliant, we can do one more check to see whether it is strongly n-complaint as follows. Let π'_n be a program obtained from π_n by replacing "`:- not conflict(n)`" with "`:- conflict(n).`" We can show that if π'_n has no answer set then for every initial state of $\mathscr{S}$ no action sequence is executable or there exists some action sequence such that `conflict(n).` is true. Combining with the fact that $\mathscr{S}$ is weakly n-noncompliant, this implies that the domain is strongly n-noncompliant. Again, the proof of this property relies on the definition of answer sets of programs with constraints, which say that the constraint `:- conflict(n).` must be false in an answer set, which in turn implies that `conflict(n)` must be false. However, the program having no answer set implies that every executable sequence of actions will generate `conflict(n)`.

4.6 Likelihood of Concerns Satisfaction and Preferred Mitigation Strategies

In this subsection, we present an ASP program for computing LoS of concerns and preferred mitigation strategies using LoS. Listing 10 shows the ASP encoding for computing of LoS of concerns. It defines the predicate `llh_sat(C,N,T)` which states that the likelihood of satisfaction of concern C at time step T is N. It starts with the definition of different predicates `nAllPosCon/3` and `nActPosCon/3` representing $rel^{+}(c)$ and $rel^{+}_{sat}(c,s)$ at the step T, i.e., the number of all possible positively impacting properties on concern C and the number of positively impacting properties on concern C holding in step T, respectively. Recall that `positiveImpact(P,C)` is defined as in Subsection 4.3. Line 5 creates an ordering between subconcerns of concern C for the computation of `llh_sat(C,N,T)`. The LoS for a concern without a subconcern is computed in Line 8. Rules on the lines 9-12 compute the LoS of concerns in accordance with the order created by rule on Line 1. `llh_sat(C,N,T)` is then computed using Equation 8.

Listing 10: Π_{LoS}: Computing Likelihood of Concerns Satisfaction

```
1  nAllPosCon(C,N2,T):- concern(C), step(T), N2=#count{P,Com : comp(Com),
       prop(P), positiveImpact(P,C), addBy(C,P), relation(Com,P)}.
2  nActPosCon(C,N1,T):- concern(C), step(T), N1=#count{P,Com : comp(Com),
       prop(P), positiveImpact(P,C), addBy(C,P), relation(Com,P),
     h(active(Com,P),T)}.
3  deg_pos(C,1,T)          :- step(T), concern(C), nAllPosCon(C,0,T).
4  deg_pos(C,N1*100/N2,T) :- nAllPosCon(C,N2,T), nActPosCon(C,N1,T),
     concern(C), N2!=0.
5  order(SC,C,N) :- subCo(C,SC), N={SC < SCp : subCo(C,SCp)}.
6  hSubCo(C) :- subCo(C,SC), concern(C), concern(SC).
7  ¬hSubCo(C):- concern(C), not hSubCo(C).
8  llh_sat_sub(C,1,T) :- step(T), concern(C), ¬hSubCo(C).
9  llh_sat(C,N1*N2,T) :- step(T), concern(C), llh_sat_sub(C,N1,T),
     deg_pos(C,N2,T).
10 llh_sat_sub_aux(C,0,X,T) :- step(T), subCo(C,SC), order(SC,C,0),
     llh_sat(SC,X,T).
11 llh_sat_sub_aux(C,N,X*Y,T) :- step(T), subCo(C,SC), order(SC,C,N),
     llh_sat(SC,Y,T), llh_sat_sub_aux(C,N-1,X,T).
12 llh_sat_sub(C,X,T) :- llh_sat_sub_aux(C,N,X,T), step(T), concern(C),
     not llh_sat_sub_aux(C,N+1,_,T).
```

It is easy to check that the above program correctly computes the values of $deg^+(c,s)$ and $\varphi_{LoS}(c,s)$. Indeed, the program $\Pi(\Delta_{lkas}) = \Pi(\mathscr{S}_{lkas})^n \cup \Pi(I_{lkas}) \cup \Pi^c_{lkas} \cup \Pi_{sat} \cup \Pi^n_{plan} \cup \Pi_{LoS}$ correctly computes the LoS of concerns for various concerns as shown in Subsection 3.3.3 (Figure 5).

Having computed LoS of concerns and φ_{LoS}, identifying the best strategies in according to the two approaches in Subsection 3.3.3 is simple. We only need to add rules that aggregates the LoS of the top-level concerns specified in the CPS with their corresponding weights or preferences. This is done as follows:

- *Weighted LoS*: Listing 11 computes the weighted LoS of the final state. The rule is self-explanatory.

Listing 11: Computing Weighted LoS

```
1  scoreLoS(Sc,T) :- llh_sat(functionality,V_fun,T), wLoS(
       functionality,W_fun), llh_sat(business,V_bus,T), wLoS(business,W_bus
       ), llh_sat(human,V_hum,T), wLoS(human,W_hum), llh_sat(
       trustworthiness,V_tru,T), wLoS(trustworthiness,W_tru), llh_sat(
       timing,V_tim,T), wLoS(timing,W_tim), llh_sat(data,V_dat,T), wLoS(
       data,W_dat), llh_sat(boundaries,V_bou,T), wLoS(boundaries,W_bou),
       llh_sat(composition,V_com,T), wLoS(composition,W_com), llh_sat(
       lifestyle,V_lif,T), wLoS(lifestyle,W_lif), Sc = V_fun*W_fun + V_bus*W_bus
       + V_hum*W_hum + V_tru*W_tru + V_tim*W_tim + V_dat*W_dat + V_bou*W_bou + V_com*W_com +
       V_lif*,W_lif.
```

- *Specified Preferences LoS*: ASP solver provides a convenient way for computing preferences based on lexicographic order among elements of a set. Assume that `Trustworthiness` is preferred to `Business` then the two statements

  `#maximize`$\{V_1$`@k : llh_sat(trustworthiness,` V_1`, n)`$\}$

  `#maximize`$\{V_2$`@k': llh_sat(business,` V_2`, n)`$\}$

  with $k > k'$ and n is the length of the plan will return answer sets in the lexicographic order, preferring the concern `Trustworthiness` over `Business`. With these statements, any specified preferred LoS over the set of top-level concern can be implemented easily.

4.7 Computing Mitigation Strategy with The Best Chance to Succeed

To compute strategies with the maximal probability of success, we only need to extend the program Π^n_{plan} with the following rules:

- for each statement "a **success_with** v **if** $p_1,\ldots,p_n$", the two rules:

  $\texttt{pr(a,v,T)} \text{ :- } \texttt{h}^*(\texttt{p}_1,\texttt{T}),\ldots,\texttt{h}^*(\texttt{p}_\texttt{n},\texttt{T}).$

  $\texttt{dpr(a,T)} \text{ :- } \texttt{h}^*(\texttt{p}_1,\texttt{T}),\ldots,\texttt{h}^*(\texttt{p}_\texttt{n},\texttt{T}).$

  which check for the satisfaction of the condition in a statement defining the probability of success in the step T and states that it is defined.
- the rule:

  $\texttt{pr(A,1,T)} \text{ :- } \texttt{exec(A,T)},\texttt{not dpr(A,T)}.$

  which says that by default, the probability of success of a at step T is 1.
- computing the probability of the state at step T:

  $\texttt{prob(1,0)}.$

  $\texttt{prob(U*V,T+1)} \text{ :- } \texttt{prob(U,T)},\texttt{occurs(A,T)},\texttt{pr(A,V,T)}.$

  where the first rule says that the probability of the state at the time 0 is 1; $prob(v,t)$ states

that the probability of reaching the state at the step t is v and is computed using the second rule.

Let $\Pi^n_{bestPrS}$ be Π^n_{plan} and the above rules. We have that if $[a_0,\ldots,a_{n-1}]$ and S is an answer set of $\Pi(\Delta)\cup\Pi^n_{bestPrS}\cup\{occurs(a_i,i)\mid i=0,\ldots,n-1\}$ then $prob\left(\Pi_{i=0}^{n-1} pr(a_i,s_i),n\right)\in S$. To compute the best strategy, we add the rule

$$\texttt{\#maximize\{V : prob(V,n)\}}.$$

to the program $\Pi^n_{bestPrS}$.

Example 7

Continue with Example 2 after a cyber-attack occurs and causes the property `basic-mode` to be *False*. As in Section 4.4, the five mitigation strategies ($\alpha_1,\alpha_2,\alpha_3,\alpha_4$ and α_5) are generated to restore the LKAS system. Assume that the probability of success of $\texttt{tOn}(\texttt{basic\_mode})$, $\texttt{switM}(\texttt{cam},\texttt{advanced\_mode})$, and $\texttt{switM}(\texttt{sam},\texttt{advanced\_mode})$ are 0.2, 0.6, 0.7 in every state, respectively. In this case, the strategies α_2 and α_3 have the maximal probability to succeed.

5 Towards a Decision-Support System for CPSF

As a demonstration of the potential use of our approach, in this section we give a brief overview of a decision-support system (version 0.1) that is being built for use by CPS designers, managers and operators. We also include preliminary considerations on performance aspects.

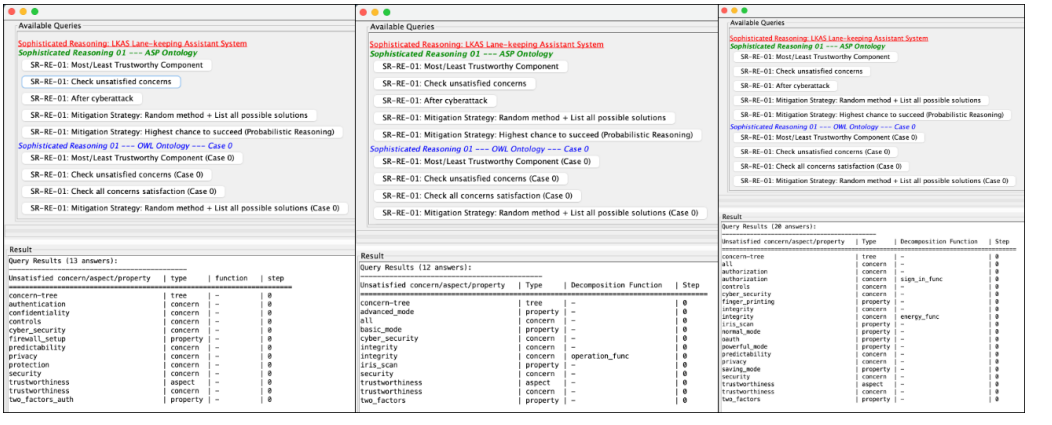

Fig. 6: Computing Satisfaction of Concerns in Reasoning Component

The decision-support system relies on an ASP-based implementation for reasoning tasks in CPS theories (described in Section 4) with the different modules for answering queries described in Section 3.3, and comprises a *reasoning component* and a *visualization component*. Figure 6 shows the *reasoning component* at work on computing satisfaction of concerns related to the LKAS domain example (described in Section 4.2). Figure 7 illustrates the reasoning component at work on other modules (Section 4.3– 4.7) with different situations related to the LKAS domain. Notice how the user can ask the system to reason about satisfaction of concerns, to produce mitigation plans as well as to select the most preferred mitigation strategy, etc.

The output of the reasoning component can then be fed to the *visualization component*, where advanced visualization techniques allow practitioners to get a birds-eye view of the CPS or dive

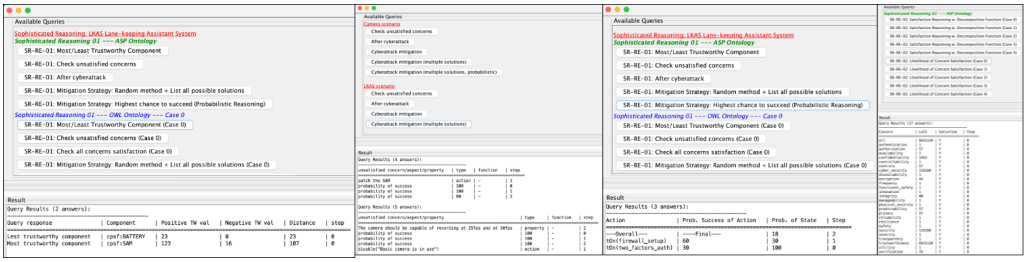

Fig. 7: Other reasoning modules in Reasoning Component

into specific details. For instance, the sunburst visual from Figure 8 provides a view of the CPS from Figure 2 where the aspects are presented in the inner most ring. Moving outwards, the visualization shows concerns from increasingly deeper parts of the concern tree and properties. The left-hand side of the figure depicts the visualization in the case in which all concerns are satisfied (blue), while the right-hand side shows how the sunburst changes when certain concerns (highlighted as red) are not satisfied. Focusing on the right-hand side, the text box open over the visual reports that the trustworthiness aspect is currently not not satisfied and the level at which this concern is not being met is the concern of privacy and the property of manageability. The visual allows for a pinpoint where within the CPS framework issues have arisen that when addressed can enable a working state. We omit the details of *visualization component* description as it is not the focus of this paper.

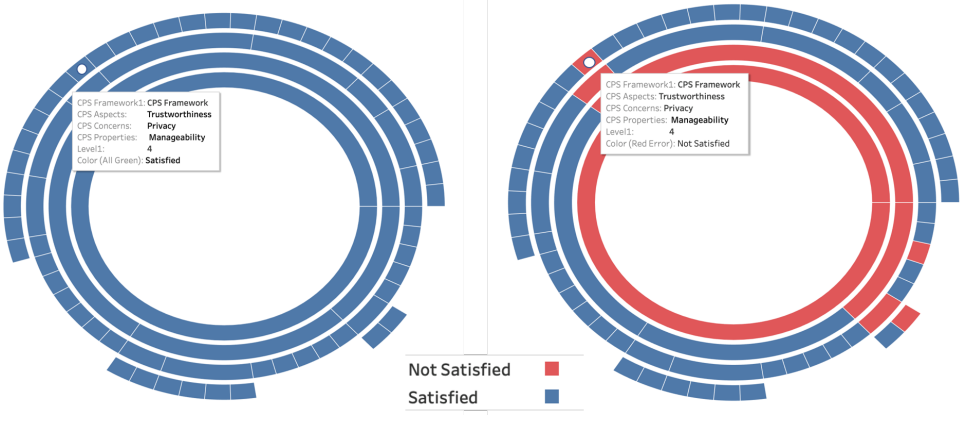

Fig. 8: Visualization component

To ensure flexibility and to allow for investigation on the scalability on larger CPS, the decision-support system is designed to support a variety of hybrid ontology-ASP reasoning engines. Currently, we consider four reasoning engines: the *naïve engine* is implemented by connecting, in a loosely-coupled manner[11], the SPARQL reasoner[12] and the Clingo ASP solver. This

[11] By loosely-coupled, we mean that the components see each other as black-boxes and only exchange information, via simple interfaces, at the end of their respective computations. Compare this with a tightly-coupled architecture, where the components have a richer interfaces for exchange state information and controlling each other's execution flow while their computations are still running.

[12] https://www.w3.org/TR/rdf-sparql-query/

engine issues a single SPARQL query to the ontology reasoner at the beginning of the computation, fetching all necessary data. The *Clingo-Python engine* is another loosely-coupled engine, leveraging Clingo's ability to run Python code at the beginning of the computation. This engine issues multiple queries in correspondence to the occurrences of special "external predicates" in the ASP program, which in principle allows for a more focused selection of the content of the ontology. The *DLVHex2 engine* also uses a similar fragmentation of queries, but the underlying solver allows for the queries to be executed at run-time, which potentially results in more focused queries, executed only when strictly needed. Finally, the *Hexlite engine* leverages a similar approach, but was specifically designed as a smaller, more performant alternative to *DLVHex2*.

In this preliminary phase of our investigation on scalability, all reasoning engines have exhibited similar performance, as exemplified by Table 2. The table summarizes the results of question-answering experiments on the Lane Keeping/Assist System (LKAS) domain and on the Smart Elevator domain Nguyen et al. (2020a). The reasoning tasks considered are for answering queries discussed earlier, including:

- ($\mathbf{Q}_1$) Computing satisfaction of concerns.
- ($\mathbf{Q}_2$) Computing most/least trustworthy components.
- ($\mathbf{Q}_3$) Generating mitigation strategies.
- ($\mathbf{Q}_4$) Non-compliance detection in a CPS.
- ($\mathbf{Q}_5$) Selecting the best mitigation strategy by preferred mitigation strategies.
- ($\mathbf{Q}_6$) Computing the likelihood of concerns satisfaction.

In Table 2, the performance of the execution for each query ($\mathbf{Q}_1$-$\mathbf{Q}_6$)[13] is measured by the average processing time of reasoning computations in our experiment CPS theories (LKAS and Smart Elevator) with different initial situations (different initial configurations). While the results show that the naïve engine is marginally better than the others, the differences are quite negligible, all within 10%. It is conceivable that larger-scale experiments will eventually exhibit similar patterns to those found in other research on the scalability of hybrid systems (e.g., Balduccini and Lierler (2017)). A thorough analysis will be the subject of a separate paper where we have done some preliminary experiment with our CPS reasoning system and found that it can work ontologies with more than 150K triples, 85 classes, 61K individuals, 30 object properties, 40 data properties, and 45 subclass relations within a minute.

6 Related Work

Due to the difference in level of abstraction, most of the approaches from the literature can be viewed as orthogonal and complementary to ours. Thus, we focus our review of related work on what we consider to be the most relevant approaches.

The literature from the area of cybersecurity is often focused on the notion of graph-based attack models. Of particular relevance is the work on Attack-Countermeasure Trees (ACT) Roy et al. (2012). An ACT specifies how an attacker can achieve a specific goal on a IT system, even when mitigation or detection measures are in place. While ACT are focused on the Cybersecurity concern, our approach is rather generally applicable to the broader Trustworthiness aspect of

[13] We use a Macbook Pro 16 running macOS Big Sur Version 11.5.2, 32GB RAM DDR4, 2.6Ghz 6-Core Intel Core i9, and ASP solver `Clingo`

Reasoning Tasks	LKAS Domain				Smart Elevator Domain			
	Naïve	Clingo -Python	DLVHex2	Hexlite	Naïve	Clingo -Python	DLVHex2	Hexlite
$\mathbf{Q}_1$	1.35s	1.48s	1.32s	1.37s	1.31s	1.45s	1.30s	1.35s
$\mathbf{Q}_2$	1.28s	1.43s	1.29s	1.32s	1.25s	1.32s	1.22s	1.30s
$\mathbf{Q}_3$	1.36s	1.52s	1.38s	1.41s	1.33s	1.49s	1.37s	1.39s
$\mathbf{Q}_4$	1.41s	1.52s	1.41s	1.45s	1.40s	1.53s	1.41s	1.47s
$\mathbf{Q}_5$	1.38s	1.47s	1.42s	1.39s	1.26s	1.39s	1.33s	1.35s
$\mathbf{Q}_6$	1.74s	1.93s	1.79s	1.81s	1.78s	1.95s	1.77s	1.86s

Table 2: CPS domains Querying, Extracting and Reasoning Summary

CPS and can in principle be extended to arbitrary aspects of CPS and their dependencies. The underlying formalization methodology also allows for capturing sophisticated temporal models and ramified effects of actions. In principle, our approach can be extended to allow for quantitative reasoning, e.g., by leveraging recent work on Constraint ASP and probabilistic ASP Balduccini and Lierler (2017); Ostrowski and Schaub (2012); Baral et al. (2009). As we showed above, one may then generate answers to queries that are *optimal* with respect to some metrics. It is worth pointing out that the combination of physical (non-linear) interaction and logical (discrete or Boolean) interaction of CPS can be modeled as a mixed-integer, non-linear optimization problem (MINLP) extended with logical inference. MINLP approaches can support a limited form of logic, e.g., through disjunctive programming Balas (1975). But these methods seem to struggle with supporting richer logics and inferences such as "what-if" explorations. For relevant work in this direction, we refer the reader to Mistr et al. (2017); D'Iddio and Huth (2017).

One major focus in the area of cybersecurity is the identification and mitigation of compromised devices. Behavior analysis and behavioral detection are some of the approaches used in this area. Uluagac et al. (2019) proposes a system-level framework for the identification of compromised smart grid devices. The approach employs a combination of system call and function call tracing, which are paired with signal processing and statistical analysis. In a similar vein, Shoukry et al. (2018) covers model-based techniques for addressing the problem of sensors that can be manipulated by an attacker. It is worth noting that, in our methodology, the presence or lack of compromised devices or components – and even the type of compromise – can be captured by means of properties, which in turn affect specific concerns. Techniques such as those described in the cited papers can then be used to determine whether such properties are satisfied or not.

Another related, complementary approach is presented in Aerts et al. (2017), where the authors tackle the problem of validation and verification of requirements. The paper proposes model-based testing as a solution to two key problems in validation and verification of requirements:

translating requirements into concrete test inputs and determining what the outcome of such tests says about the satisfaction of the requirements. From this point of view, the approach from Aerts et al. (2017) can be used to provide the information about satisfaction of requirements that is necessary for the reasoning tasks covered by in our investigation.

Lee (2016) analyzes the role of models in the engineering of CPS and argues for classes of models that trade accuracy and detail in favor of simplicity and clarity of semantics. This idea is in line with the considerations that prompted the development of CPSF, and which are infused in our work through its legacy. In a related fashion, Roehm et al. (2019) proposes a survey of *conformance relations*, where the term describes the link between functional behavior of a model and the behavior of the implemented system (or of a more concretized model). Conformance relations are typically applied to the task of analyzing requirements and their link to the CPS being modeled, and in that sense Roehm et al. (2019) is orthogonal to our work. On the other hand, the paper elicits the interesting issue of whether the characterization of CPS from CPSF might be viewed, itself, as a conformance relation. This is an open question, which we plan to address in the future.

Tepjit et al. (2019) presents a rich survey of frameworks for implementing reasoning mechanisms in smart CPS. It is to be noted that the focus of the paper is on the reasoning mechanisms that occur within a CPS in order to achieve "smartness" Tepjit et al. (2019), while our focus is on reasoning mechanisms that allow designers, maintainers and operators to reason about a CPS – where the CPS itself may or may not be "smart." There is certainly a certain degree of overlap between this paper and our work, but also of important differences. In particular, the reasoning mechanisms we discussed here are not always applicable at the system level, which is the focus of Tepjit et al. (2019). For instance, our techniques could be used in real-time by a CPS to determine whether its functional aspect is satisfied, but it may be unrealistic for a CPS to reason about its own trustworthiness. From another point of view, reasoning mechanisms discussed in Tepjit et al. (2019), such as planning and decision-making, can be viewed as tools for the satisfaction of properties. In this sense, a designer might want to use the results of that paper to ensure that the decision-making mechanisms implemented *within* a CPS satisfy certain properties that are responsible for ensuring the functional aspect of the CPS or even its trustworthiness.

The methodologies proposed in our paper build on a vast number of research results in ASP and related areas such as answer set planning, reasoning about actions, etc. and could be easily extended to deal with other aspects discussed in CPSF. They are well-positioned for real-world applications given the efficiency and scalability of ASP-solvers (e.g., `clingo` Gebser et al. (2007)) that can deal with millions of atoms, incomplete information, default reasoning, and features that allow ASP to interact with constraint solvers and external systems.

7 Conclusions and Future Work

The paper presents a precise definition of a CPS, which, in conjunction with the CPS Ontology Framework by NIST, allows for the representing and reasoning of various problems that are of interest in the study of CPS. Specifically, the paper defines several problems related to the satisfaction of concerns of a CPS theories such as the problem of identifying non-compliant CPS systems, the problem of identifying the most/least trustworthy or vulnerable components, computing mitigation strategies, a most preferred mitigation strategies, or strategies with the best chance to succeed. For each problem, the paper presents a formal definition of "*what is the problem?*" and provides an ASP program that can automatically verify such properties. To

the best of our knowledge, all of these contributions are new to the research in Cyber-Physical Systems.

The current ASP implementation[14] (version 0.1) provides a first step towards developing a tool for CPS practitioners and designers. It automatically translates a system specification as an ontological description (e.g., as seen in Figure 5) to ASP code and allows users to ask questions related to the aforementioned issues. It has been validated against small systems. One of our goals in the immediate near future is to develop an user-friendly interface that allows users to design or model their real-world CPS and identify potential issues within their systems and possible ways to address these issues before these issues become harmful.

 Portions of this publication and research effort are made possible through the help and support of NIST via cooperative agreements 70NANB18H257 and 70NANB21H167.

Acknowledgment. Matthew Bundas has been supported by the GAANN grant #P200A180005. Tran Cao Son has been partially supported by the following NSF grants 1914635, 1757207, and 1812628.

References

AERTS, A., RENIERS, M. A., AND MOUSAVI, M. R. 2017. *Cyber-Physical Systems - Foundations, Principles and Applications*, chapter 19. Model-Based Testing of Cyber-Physical Systems, pp. 287–304. Intelligent Data-Centric Systems.

BALAS, E. Disjunctive programming: Cutting planes from logical conditions. In *Nonlinear Programming 2* 1975, pp. 279–312. Elsevier.

BALDUCCINI, M., GRIFFOR, E., HUTH, M., VISHIK, C., BURNS, M., AND WOLLMAN, D. A. 2018. Ontology-based reasoning about the trustworthiness of cyber-physical systems. *ArXiv*, *abs/1803.07438*, 1.

BALDUCCINI, M. AND LIERLER, Y. 2017. Constraint Answer Set Solver EZCSP and Why Integration Schemas Matter. *Journal of Theory and Practice of Logic Programming (TPLP)*, *17*, 4, 462–515.

BARAL, C., GELFOND, M., AND RUSHTON, N. 2009. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*, *9*, 1, 57–144.

BARAL, C., TRAN, N., AND TUAN, L.-C. Reasoning about actions in a probabilistic setting. In *AAAI/IAAI* 2002, pp. 507–512.

D'IDDIO, A. C. AND HUTH, M. 2017. ManyOpt: An Extensible Tool for Mixed, Non-Linear Optimization Through SMT Solving. *CoRR*, *abs/1702.01332*.

EITER, T. Answer set programming for the semantic web. In DAHL, V. AND NIEMELÄ, I., editors, *Logic Programming, 23rd International Conference, ICLP 2007, Porto, Portugal, September 8-13, 2007, Proceedings* 2007, volume 4670 of *Lecture Notes in Computer Science*, pp. 23–26. Springer.

GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. clasp: A conflict-driven answer set solver. In BARAL, C., BREWKA, G., AND SCHLIPF, J., editors, *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)* 2007, volume 4483 of *Lecture Notes in Artificial Intelligence*, pp. 260–265. Springer-Verlag.

GELFOND, M. AND LIFSCHITZ, V. Logic programs with classical negation. In WARREN, D. AND SZEREDI, P., editors, *Logic Programming: Proceedings of the Seventh International Conference* 1990, pp. 579–597.

[14] Available at `https://github.com/thanhnh-infinity/Research_CPS`

GELFOND, M. AND LIFSCHITZ, V. 1993. Representing actions and change by logic programs. *Journal of Logic Programming*, *17*, 2,3,4, 301–323.

GELFOND, M. AND LIFSCHITZ, V. 1998. Action Languages. *Electronic Transactions on Artificial Intelligence*, *3*, 6.

GELFOND, M. AND SON, T. C. Prioritized default theory. In *Selected Papers from the Workshop on Logic Programming and Knowledge Representation 1997* 1998, pp. 164–223. Springer Verlag, LNAI 1471.

GHALLAB, M., NAU, D., AND TRAVERSO, P. 2004. *Automated planning: theory and practice*. Morgan Kaufmann Publishers.

GRIFFOR, E., GREER, C., WOLLMAN, D. A., AND BURNS, M. J. Framework for cyber-physical systems: volume 1, overview 2017a.

GRIFFOR, E., GREER, C., WOLLMAN, D. A., AND BURNS, M. J. Framework for cyber-physical systems: Volume 2, working group reports 2017b.

LEE, E. A. 2016. Fundamental Limits of Cyber-Physical Systems Modeling. *ACM Transactions on Cyber-Physical Systems*, *1*, 1, 1–26.

MAREK, V. AND TRUSZCZYŃSKI, M. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-year Perspective* 1999, pp. 375–398.

MISTR, M., D'IDDIO, A. C., HUTH, M., AND MISENER, R. 2017. Satisfiability modulo theories for process systems engineering. eprints for the optimization community.

MOSCHOPOULOS, J. 2001. Ship Control Technology; A US Navy Perspective. *IFAC Proceedings Volumes*, *34*, 7, 381–388.

NGUYEN, T., SON, T. C., BUNDAS, M., BALDUCCINI, M., GARWOOD, K. C., AND GRIFFOR, E. Reasoning about trustworthiness in cyber-physical systems using ontology-based representation and asp. In *PRIMA* 2020a.

NGUYEN, T. H., PONTELLI, E., AND SON, T. C. On repairing web services workflows. In KOMENDANTSKAYA, E. AND LIU, Y. A., editors, *Practical Aspects of Declarative Languages* 2020b, pp. 37–53, Cham. Springer International Publishing.

NGUYEN, T. H., POTELLI, E., AND SON, T. C. 2018a. Phylotastic: An experiment in creating, manipulating, and evolving phylogenetic biology workflows using logic programming. *Theory and Practice of Logic Programming*, *18*a, 3-4, 656–672.

NGUYEN, T. H., SON, T. C., AND PONTELLI, E. Automatic web services composition for phylotastic. In *PADL 2018, Los Angeles, CA, USA, January 8-9, 2018, Proceedings* 2018b, pp. 186–202.

NIEMELÄ, I. 1999. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, *25*, 3,4, 241–273.

NIEMELÄ, I., SIMONS, P., AND SOININEN, T. Stable model semantics for weight constraint rules. In *Proceedings of the 5th International Conference on on Logic Programming and Nonmonotonic Reasoning* 1999, pp. 315–332.

OSTROWSKI, M. AND SCHAUB, T. 2012. ASP Modulo CSP: The Clingcon System. *Journal of Theory and Practice of Logic Programming (TPLP)*, *12*, 4–5, 485–503.

ROEHM, H., OEHLERKING, J., WOEHRLE, M., AND ALTHOFF, M. 2019. Model Conformance for Cyber-Physical Systems: A Survey. *ACM Transactions on Cyber-Physical Systems*, *3*, 3, 1–26.

ROY, A., KIM, D. S., AND TRIVEDI, K. S. 2012. Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. *Security and Communication Networks*, *5*, 8, 929–943.

SHOUKRY, Y., CHONG, M., WAKAIKI, M., NUZZO, P., SESHIA, S. A., HESPANHA, J. P., AND TABUADA, P. 2018. SMT-Based Observer Design for Cyber-Physical Systems under Sensor Attacks. *ACM Transactions on Cyber-Physical Systems*, *2*, 1, 1–27.

SON, T., BARAL, C., TRAN, N., AND MCILRAITH, S. 2006. Domain-dependent knowledge in answer set planning. *ACM Trans. Comput. Logic*, *7*, 4, 613–657.

TEPJIT, S., HORVATH, I., AND RUSAK, Z. 2019. The State of Framework Development for Implementing Reasoning Mechanisms in Smart Cyber-Physical Systems: A Literature Review. *Journal of Computational Design and Engineering*, *6*, 527–541.

ULUAGAC, C. S., AKSU, H., AND BABUN, L. 2019. A System-level Behavioral Detection Framework for Compromised CPS Devices: Smart-Grid Case. *ACM Transactions on Cyber-Physical Systems*, *4*, 2.

WOLLMAN, D. A., WEISS, M. A., LI-BABOUD, Y.-S., GRIFFOR, E., AND BURNS, M. J. Framework for cyber-physical systems: Volume 3, timing annex 2017.