Specifying and Reasoning about Concerns in Cyber-Physical System Using Answer Set Programming

Thanh Hai Nguyen[1] , **Matthew Bundas**[1] , **Tran Cao Son**[1] and
Marcello Balduccini[2] , **Kathleen Campbell Garwood**[2] and
Edward R. Griffor[3]
[1]New Mexico State University, [2]St. Joseph University, [3]NIST
{thanhnh, bundasma, stran}@nmsu.edu, {mbalducc, kcampbel}@sju.edu, edward.griffor@nist.gov

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 the concern related to the dependence or conflicts between concerns, how to mitigate an issue, and what would be the most suitable mitigation strategy for a given issue. It then shows how ASP can be used to address the aforementioned problems. The paper concludes with a discussion of the potentials of the proposed methodologies.

Introduction

Self-driving cars, the utility (potable water, wastewater) distribution systems, the electric power grid, and the transportation network are a few examples of cyber-physical systems (CPS)[1] that are (or soon to be) a part of our daily life. A CPS integrates computation and physical components. CPS technologies are poised to transform the way people interact with engineered systems with virtually limitless potential. This is a reason for excitement as well as worries concerning the use of CPS. Recent incidents such as the hack of the Ring security cameras[2] or the experiment-cum-stunt of self-driving cars[3], raise serious concerns about the security and privacy of CPS. These two examples raise the concern called `Trustworthiness`.

Indeed, there are several potential concerns related to the use of CPS. As such, for a widespread adoption of CPS, methodologies and systems must be developed to address concerns related to the CPS, its components, and its interactions with the outside world. Given a CPS specification, we are interested in the following questions: (*i*) will a certain concern be satisfied? (*ii*) is there any potential conflict between the concerns; and (*iii*) how can we generate a plan that addresses an issue. The next example shows that competing concerns is also a problem in CPS.

Example 1 *In the autonomous car system (ACS), an* `Integrity` *concern refers to the fact that the packets sending from the* `wind-sensor` *to the situational awareness module (*`SAM`*) should be fast and reliable; another concern is the* `Encryption` *concern, which dictates that all communication channel must be encrypted.*

Consider a situation in which the ACS has only one possible channel, a socket connection, which is fast, reliable, but not encrypted. In this situation, the two concerns are in conflict with each other. It is impossible to satisfy both concerns.

The National Institute of Standards and Technology (NIST) has taken the first step towards addressing the aforementioned challenge. In the last few years, NIST has established the CPS Public Working Group (CPS PWG) to bring together a broad range of CPS experts in an open public forum to help define and shape key characteristics of CPS, which allows us to better manage development and implementation within, and across, multiple smart application domains. This resulted in the CPS Framework (CPSF) (Griffor et al. 2017a; Griffor et al. 2017b; Wollman et al. 2017), which provides a principled design and analysis methodology for CPS that is intended to be applicable across all relevant domains of expertise. The methodology builds upon the CPS Framework core concepts of *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). Notably, CPSF also provides the foundation for the development of reasoning systems capable of answering questions related to concerns in a CPS (Balduccini et al. 2018).

This paper presents an ASP based solution to a variety of important problems related to the satisfaction of concerns of CPS. We formally propose the notion of a CPS system that (*i*) allows for the specification of CPS with functional decomposition of concerns; (*ii*) enables the automatic identification of conflicts between concerns; and (*iii*) enables the application of planning techniques in computing mitigation strategies.

The paper is organized as follows. The next section

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

[2]https://www.cnn.com/2019/12/12/tech/ring-security-camera-hacker-harassed-girl-trnd/index.html

[3]https://www.ft.com/content/6000981a-1e03-11e8-aaca-4574d7dabfb6

presents the background. Afterwards, we present a formalization of a CPS theory and its ASP encoding. We then discuss the different problems in CPS and propose solutions to them.

Background

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 atoms of a propositional language[4] and *not* represents (default) negation. 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 P^I.

Several extensions (e.g., *choice atoms*, *aggregates*, etc.) have been introduced to simplify the use of ASP. We will use and explain them whenever it is needed.

NIST CPS Framework CPSF provides a principled representation of CPS developed by the working group with various intents, including making it possible for experts from disparate disciplines to collaborate on the design, analysis, and maintenance of CPS. CPSF defines high-level concept of "Concern" with its refinement of "Aspect." CPSF comes with an associated ontology, where these concepts are formalized as classes and, for Aspect, subclasses. Specific concerns are represented as individuals: `Trustworthiness` as an individual of class Aspect, `Security` and `Cybersecurity` of class Concern. 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`. The top part of Fig. 1, excluding the nodes labeled SAM, CAM and BAT and links labeled "relates", shows a fragment of CPSF where circle nodes represent concerns and rectangle nodes represent properties. Connections between concerns represent

[4]For simplicity, we often use first order logic atoms in the text which represent all of its ground instantiations.

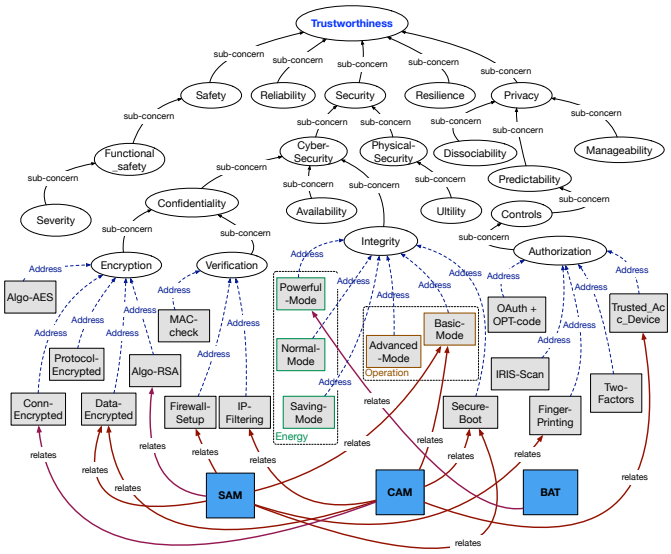

Figure 1: A Fragment of the Trustworthiness Concern Ontology

sub-concern relations. Link from a property to a concern represents that the property *addresses* an aspect of the concern which might be grouped into different *functionality*. For example, `Integrity` is a sub-concern of `Cyber-security`, which is a sub-concern of `Security`, a sub-concern of `Trustworthiness`; `Secure-Boot`, `Powerful-Mode` properties address the `Integrity` concern; `Powerful-Mode`, `Normal-Mode`, and `Saving-Mode` address the `Energy` functionality of the `Integrity` concern; etc.

CPSF provides an informal description of "what does it mean for a concern to be satisfied?" Preliminary work applying CPSF in reasoning about satisfaction of concerns can be found in (Balduccini et al. 2018) which requires a strong notion of satisfaction of concern in that a concern is satisfied if all *properties* related to it are satisfied and all its *sub-concerns* are satisfied.

Representing & Reasoning with CPSF Ontology in ASP There have been work on using ASP in the context of Semantic Web (e.g., (Eiter 2007; Nguyen, Son, and Pontelli 2018; Nguyen, Potelli, and Son 2018)) that emphasize the use of ontologies. In these works, an ASP program is used for reasoning about classes, properties, inheritance, relations, etc. We will use a similar approach in this paper. For simplicity of representation, we assume that all classes, instances, relations, properties of the CPS ontology are encoded in an ASP program[5]. 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`;

[5]They could be used on a need-based basis via an ontology-query solver. This will be necessary when the ontology is large.

- `aspect(I)` (resp. `concern(I)`, `prop(I)`): `I` is an individual of class aspect (resp. concern, property);
- `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);
- `func(F,C)`: F is a functional decomposition of concern C.

To reason about a subclass relationship, the encoding contains the rule:

```
subClass(X,Y):-subClass(Z,Y),subClass(X,Z).
```

Similar rules for reasoning about the inheritance between concerns, inheritance between subconcerns and concerns, etc. are introduced whenever they are used subsequently. We note that the CPSF 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.

CPS Theory: Formal Definition

In this section, we will formalize the notions of a CPS system and a CPS theory. To motivate this notion, let us look at a concrete example.

Example 2 (Extended from (Balduccini et al. 2018)) *Consider a lane keeping/assist (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, through a physical output, the automated navigation system. 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 work on two operational modes, either basic mode (*`basic_mode` *or* `b_mode`*) or advanced mode (*`advanced_mode` *or* `a_mode`*). 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. CAM is capable of two recording modes, one at 25 fps (frames per second) and the other at 50 fps. The selection of the recording mode is made by SAM, by acting on a flag of the camera's configuration. 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`*.*

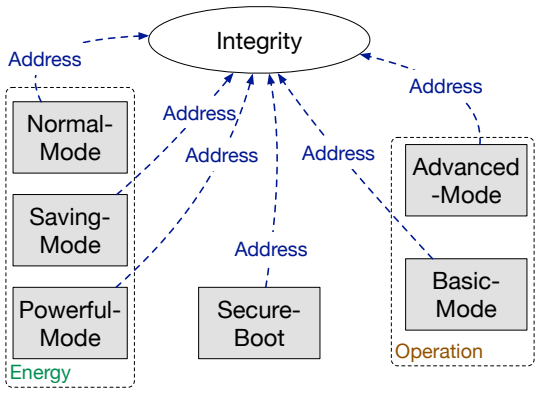

Figure 2: `Integrity` and its Functionalities and Properties

Before we get to the precise definition of a CPS system, let us discuss the relationship between the CPS and the ontology. 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: $(\texttt{secure\_boot}) \wedge (\texttt{advanced\_mode} \vee \texttt{basic\_mode}) \wedge (\texttt{saving\_mode} \vee \texttt{normal\_mode} \vee \texttt{powerful\_mode})$

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 execute over* $\mathscr{S}$*;*
- F *is a finite set of fluents (or state variables) of the system;*
- 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* $\omega(c)$*; and*
- 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.*

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 these properties can be changed by actions, we assume that: $\bigcup_{co \in CO} R(co) \cup \{active(co, p) \mid co \in CO, p \in R(co)\} \subseteq F$. (A, F) is associated with a set of statements in

one of the following forms:

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

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

$$\textit{State constraint}: \quad f \textbf{ if } p_1,\ldots,p_n \quad (3)$$

where fluent f or p_i stands for a fluent $p \in F$ or its negation $\neg p$. (1) states that a can only be executed if $p_1,\ldots,p_n$ are true; (2) says that if a is executed in a state where $p_1,\ldots,p_n$ are true then f will become true; and (3) states that f is true if if $p_1,\ldots,p_n$ are true. Semantics of dynamic domain is defined in (Gelfond and Lifschitz 1998). 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.

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

- $CO_{lkas} = \{\texttt{SAM}, \texttt{CAM}, \texttt{BAT}\}$.
- A_{lkas} *contains the following actions:*
  - `switM(X,M)`, `switA(X,A)`, `switV(X,V)`, `switEM(X,EM)` *and* `switEA(X,EA)` *which denote the actions of switching component X to mode M, authorization method A, verification method V, encryption method EM and encryption algorithm EA respectively, where the set of statements for action* `switM(X,M)` *could be:*
    executable `switM(cam,a_mode)` **if** `on(cam), active(cam,basic_mode)`
    `switM(cam,a_mode)` **causes** `active(cam,a_mode)`
    `switM(cam,a_mode)` **causes** `on(cam)`
    `switM(cam,a_mode)` **causes** $\neg$`active(cam,basic_mode)`
    where `a_mode` *stands for* `advanced_mode` *and similar statements for* `switA(X,A)`, `switV(X,V)`, `switEM(X,EM)` *and* `switEA(X,EA)` *respectively.*
  - `tOn(P)` *and* `tOff(P)` *denotes actions of enabling and disabling the truth value of property P. The set of statements for action* `tOn(P)` *could be:*
    executable `tOn(basic_mode)` **if** `audit(basic_mode,false)`
    `tOn(basic_mode)` **causes** `audit(basic_mode,true)`
    `tOn(basic_mode)` **causes** $\neg$`audit(basic_mode,false)`
    and the similar statements for action `tOff(P)`.
- F_{lkas} *contains the following fluents:*
  - `active(X,P)` *denotes that component X is working with property P. e.g.,* `active(cam,basic_mode)`, `active(cam,data_encrypted)`, `active(sam,finger_printing)` *and* `active(bat,normal_mode)` *denote that CAM is working in basic mode, with encrypted data, SAM is authenticated by finger printing method and BAT is working in normal mode.*
  - `on(X)` *(*`off(X)`*) denotes that component X is (isn't) ready to use.*
- $R_{lkas} = \{$`CAM` $\mapsto \{$`ip_filtering, data_encrypted, conn_encrypted, protocol_encrypted, mac_check, secure_boot, basic_mode, trusted_acc_device, advanced_mode`$\}$, `SAM` $\mapsto \{$`data_encrypted, algo_RSA, algo_AES, protocol_encrypted, conn_encrypted, firewall_setup, advanced_mode, basic_mode, finger_printing, two_factors, iris_scan, secure_boot`$\}$, `BAT` $\mapsto \{$`powerful_mode, normal_mode, saving_mode`$\}$.
  For visualization, the components and relations to the properties are drawn in the bottom part of figure 1.
- Γ_{lkas} *contains the following triples (see also Fig. 2 and 3):*
  - *(*`integrity, operation, advanced_mode` $\vee$ `basic_mode` *) says the satisfaction of formula* `advanced_mode` $\vee$ `basic_mode` *addresses the concern* `integrity` *in the relevant functional decomposition* `operation`.
  - *(*`integrity, energy, saving_mode` $\vee$ `normal_mode` $\vee$ `powerful_mode`*) denotes the formula* `saving_mode` $\vee$ `normal_mode` $\vee$ `powerful_mode` *addresses the concern* `integrity` *in the relevant functional decomposition* `energy`.
  - *(*`authorization, sign_in, oauth` $\wedge$ `opt_code`*) denotes the satisfaction of formula* `oauth` $\wedge$ `opt_code` *addresses the relevant functional decomposition* `sign_in` *of the concern* `authorization`.
  - *(*`authorization, sign_in, two_factors` $\vee$ `finger_printing` $\vee$ `iris_scan`*) denotes the formula* `two_factors` $\vee$ `finger_printing` $\vee$ `iris_scan` *addresses the concern* `authorization` *in the relevant functional decomposition* `sign_in`.
  - *(*`authorization, sign_in, oauth` $\wedge$ `ip_check` $\wedge$ `email_verify`*) denotes that the concern* `authorization` *with the relevant functional decomposition* `sign_in` *is addressed by formula* `oauth` $\wedge$ `ip_check` $\wedge$ `email_verify`.

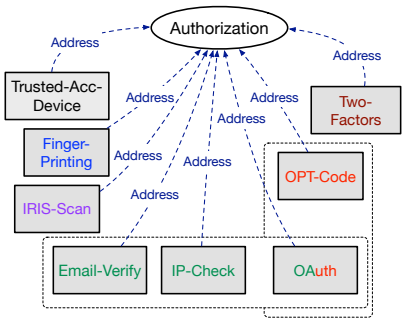

Figure 3: `Authorization` concern and its Functionalities

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). Precise definition of this notion can be found in (Gelfond and Lifschitz 1998). In the following, we will follow this convention to describe a state s as a subset of F and say that $f \in s$ is true in s and $f \notin s$ is false in s.

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* $\mathscr{S}$.

Given $(\mathscr{S},I)$ where $\mathscr{S} = (CO,A,F,R,\Gamma)$, the action domain (A,F) specifies a transition function between states. In each state, the satisfaction of a particular concern in the CPSF is evaluated using the relationship R and the components C. As the transition function induced by (A,F) has been defined in (Gelfond and Lifschitz 1998), we will next discuss how the satisfaction of concerns in the CPSF can be evaluated. This is done by using ASP.

ASP Encoding of a CPS Theory

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)). We start with the encoding of the theory. Given $(\mathscr{S},I)$ where $\mathscr{S} = (CO,A,F,R,\Gamma)$, $\Pi(\mathscr{S})^n$, where n is a non-negative integer representing the horizon of the system that we are interested in, contains the following atoms[6]:

- for each $0 \leq t \leq n$, an atom $step(t)$;
- for each $co \in CO$, an atom $comp(co)$;
- for each $a \in A$, an atom $action(a)$;
- for each $f \in F$, an atom $fluent(f)$;
- for each $co \in CO$ and $p \in R(co)$, an atom $relation(co,p)$;
- 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, 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
    $\texttt{exec(a,T)} :- \texttt{step(T)}, \texttt{h}^*(\texttt{p}_1,\texttt{T}), \ldots, \texttt{h}^*(\texttt{p}_\texttt{n},\texttt{T})$.
  - For each dynamic causal law of the form (2)
    $\texttt{h}^*(\texttt{f},\texttt{T}+1) :- \texttt{step(T)}, \texttt{occurs(a,T)}, \texttt{h}^*(\texttt{p}_1,\texttt{T}), \ldots, \texttt{h}^*(\texttt{p}_\texttt{n},\texttt{T})$.
  - For each state constraint of the form (3):
    $\texttt{h}^*(\texttt{f},\texttt{T}) :- \texttt{step(T)}, \texttt{h}^*(\texttt{p}_1,\texttt{T}), \ldots, \texttt{h}^*(\texttt{p}_\texttt{n},\texttt{T})$.
  - The rules encoding the inertial axiom:
    $\texttt{h(f,T+1)} :- \texttt{step(T)}, \texttt{h(f,T)}, \texttt{not } \neg\texttt{h(f,T+1)}$.
    $\neg\texttt{h(f,T+1)} :- \texttt{step(T)}, \neg\texttt{h(f,T)}, \texttt{not h(f,T+1)}$.

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

Listing 1: Example program $\Pi(\mathscr{S}_{lkas})$ for LKAS

```
comp(sam). comp(cam). comp(bat).
action(switD(sam,data_encrypted)).
action(switM(cam,advanced_mode)).
relation(cam,ip_filtering).
relation(cam,data_encrypted).
relation(cam,conn_encrypted).
relation(cam,secure_boot).
relation(cam,basic_mode).
relation(cam,trusted_acc_device).
relation(sam,data_encrypted).
relation(sam,algo_RSA).
relation(sam,firewall_setup).
relation(sam,secure_boot).
relation(sam,basic_mode).
relation(sam,finger_printing).
relation(bat,powerful_mode).
```

[6]We follow the convention in logic programming and use strings starting with lower/uppercase letter to denote constants/-variables.

Listing 1 shows the encoding of components, actions, and relations of $\mathscr{S}_{lkas}$. Listing 2 shows the ASP implementation reasoning for Γ_{lkas} that is illustrated in Figures 2 and 3. Each *formula* of a concern c is identified by a unique identification number i.e. 1-5, denoted by `formula/1`. The predicate `func(F,C)` states that F is the functional decomposition of concern C. The predicate `member(P,G)` states that property P is a member in formula G. The predicate `addFun(C,F,G)` encodes that properties in formula G address the concern C over relevant functional decomposition F.

Listing 2: A part of ASP encoding for Γ_{lkas} and Integrity concern

```
concern(integrity).
prop(advanced_mode). prop(basic_mode).
prop(saving_mode). prop(normal_mode).
prop(powerful_mode). prop(secure_boot).
addBy(integrity,advanced_mode).
addBy(integrity,basic_mode).
addBy(integrity,saving_mode).
addBy(integrity,normal_mode).
addBy(integrity,powerful_mode).
addBy(integrity,secure_boot).
formula(1..5).
func(operation_func,integrity).
func(energy_func,integrity).
member(advanced_mode,1).
member(basic_mode,2).
member(powerful_mode,3).
member(normal_mode,4).
member(saving_mode,5).
addFun(integrity,operation_func,1).
addFun(integrity,operation_func,2).
addFun(integrity,energy_func,3).
addFun(integrity,energy_func,4).
addFun(integrity,energy_func,5).
```

The encoding of the initial configuration I of a CPS theory $(\mathscr{S},I)$, denoted by $\Pi(I)$, consists of atoms of the form $obs(f,true)$ and $obs(f,false)$ for $f \in I$ and two rules that define the predicate h and $\neg$h at the time step 0. Listing 3 shows a part of the initial configuration of $\mathscr{S}_{lkas}$ and the rules that define h and $\neg$h (Lines 12-13) which state that a property does hold if it is in I and does not hold otherwise.

Listing 3: A part of initial configuration of $\Pi(I_{lkas})$

```
obs(finger_printing,true). obs(oauth,true).
obs(two_factors,true). obs(opt_code,true).
obs(iris_scan,false). obs(ip_check,true).
obs(email_verify,false).
obs(trusted_acc_device,true).
obs(secure_boot,true).
obs(basic_mode,true).
obs(advanced_mode,true).
obs(saving_mode,true).
obs(normal_mode,true).
obs(powerful_mode,true).
h(P,0) :- obs(P,true), prop(P).
¬h(P,0) :- obs(P,false), prop(P).
```

Listing 4 is for reasoning about the satisfaction of concerns in CPS theory: $\texttt{h}(\texttt{sat}(\texttt{C}),\texttt{T})$ states that concern C is satisfied at the time step T; and $\texttt{h}(\texttt{sat}(\texttt{C},\texttt{F}),\texttt{T})$ states that concern C relevant to functional decomposition F is satisfied at the

time step T.

Listing 4: Π_{sat} : Satisfaction Reasoning in Ω

```
1  ¬sat_formula(C,F,G,T):- concern(C),prop(P),
2      formula(G), func(F,C), addBy(C,P),
3      member(P,G), addFun(C,F,G), not h(P,T),
4      step(T).
5  sat_formula(C,F,G,T):-concern(C),func(F,C),
6      formula(G), addFun(C,F,G), step(T),
7      not ¬sat_formula(C,F,G,T).
8  ¬h(sat(C,F),T):- concern(C), func(F,C),
9      not sat_formula(C,F,_,T).
10 h(sat(C,F),T):- not ¬h(sat(C,F),T),
11     func(F,C).
12 ¬h(sat(C),T) :- concern(C), func(F,C),
13     not h(sat(C,F),T).
14 ¬h(sat(C),T) :- concern(C), func(F,C),
15     ¬h(sat(C,F),T).
16 ¬h(sat(C),T):- ¬h(P,T), addBy(C,P),
17     not member(P,_).
18 ¬h(sat(X),T):- subCo(X,Y), not h(sat(Y),T).
19 ¬h(sat(X),T):- subCo(X,Y), ¬h(sat(Y),T).
20 h(sat(C),T):- not ¬h(sat(C),T), concern(C).
```

The first rule (lines 1-4) encodes the reasoning for `sat_formula/4` which checks whether are there any unsatisfied property that is a member in a formula G addressing concern C relevant to functional decomposition F. The next rule (lines 5-7) states that `sat_formula/4` is satisfied if it cannot be proven to be unsatisfied. The third rule (lines 8-9) reasons the unsatisfaction of a concern C relevant to functional decomposition F at time step T if the predicate `sat_formula/4` does not hold. The rule in line 10 says that a concern C relevant to functional decomposition F is satisfied if it cannot be proven to be unsatisfied. Lines 12-15 represent that if there is not any evidence for the satisfaction of concern C relevant to functional decomposition F or it is not satisfied, then this concern is unsatisfied. Rule in lines 16-17 states that concern C is not addressed if some of its properties that are not belong to any formula G of C are false. Rules in lines 18-19 propagate the unsatisfaction of a concern from its subconcerns. Finally, a concern is satisfied if it cannot be proven to be unsatisfied. The satisfaction of concerns is defined next.

Definition 3 (Concern Satisfaction) *Given a CPS theory $\Delta = (\mathscr{S}, I)$ and a concern c, we say that c is satisfied (or unsatisfied) at the time t if $h(sat(c),t)$ (or $\neg h(sat(c),t)$) is in every answer set of $\Pi(\Delta)$, denoted by $(\mathscr{S}, I) \models h(sat(c),t)$ or $(\mathscr{S}, I) \models \neg h(sat(c),t)$ respectively.*

We note that notion of satisfaction of a concern has been considered in (Balduccini et al. 2018). The major difference between this paper and the work lies in the introduction of formulas representing the satisfaction of concerns.

For each CPS theory $\Delta = (\mathscr{S}, I)$, we denote $\Pi(\Delta) = \Pi(\mathscr{S})^n \cup \Pi(I) \cup \Pi_{sat}$. It is worth mentioning that $\Pi(\Delta)$ 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 $\Pi(\Delta) \cup \{occurs(a_i, i) \mid i = 0, \ldots, n-1\}$ 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+1} is executable after the execution of the sequence $[a_0, \ldots, a_i]$; (*iii*) for each i, the set $\{h(f,i) \mid f \in F, h(f,i) \in S\}$ is a state of $\mathscr{S}$.

Conflict Detection in CPS Systems

A serious issue in CPS is that some concern cannot be satisfied. Example 1 shows that there exists situation in which competing concerns cannot be satisfied at the same time. In general, the problem is formulated as follows.

Definition 4 (Conflict) *Given the CPS system $\mathscr{S} = (CO, A, F, R, \Gamma)$, an integer k, a set of actions $SA \subseteq A$, and a set of concerns SC, we say that $\mathscr{S}$ is* k-conflict *wrt. (SA, SC) if there exists a sequence α of at most k actions in SA and an initial state I, such that $(\mathscr{S}, I) \models \neg h(sat(c), k+1)$ for some concern $c \in SC$.*

The program $\Pi(\mathscr{S})^{k+1} \cup \Pi_{sat}$ can be used in conflict detection by adding the following rules:

Listing 5: $\Pi^k(SA, SC)$: Conflict Detection

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

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 $\Pi(\mathscr{S})^{k+1} \cup \Pi_{sat} \cup \Pi^{k+1}(SA, SC)$ 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 c will not be satisfied after k steps.

Computing Mitigation Strategies

The mitigation problem in a CPS can be defined as follows:

Definition 5 (Mitigation Strategy) *Let $\Delta = (\mathscr{S}, I)$ be a CPS theory where $\mathscr{S} = (CO, A, F, R, \Gamma)$ is a CPS domain. Let Σ be a set of concerns. A mitigation strategy addressing Σ is a plan $[a_1, \ldots, a_k]$ whose execution at the initial state s results in a state s' such that for every $c \in \Sigma$, c is satisfied in s'.*

Listing 6: 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{occurs(A,T):action(A)}1` — a *choice atom* — is 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 $\{:-\texttt{not h(sat(c),n).} \mid \texttt{c} \in \Sigma\}$. Based on the results in answer set planning, we can show:

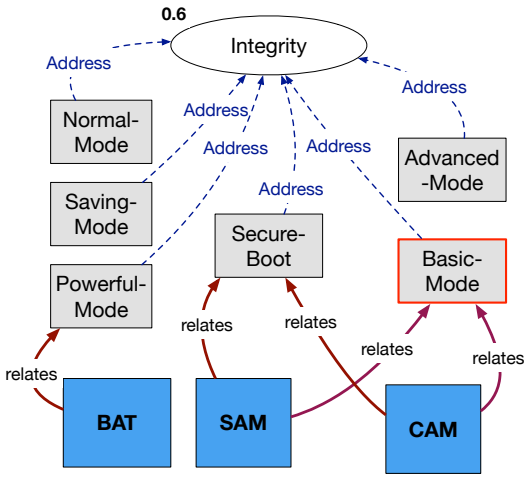

Figure 4: Current configuration of Δ_{lkas} related to Integrity concern after cyber-attack

Proposition 1 *Let $\Delta = (\mathscr{S}, I)$ be a CPS theory and Σ be a set of concerns. 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 $\Pi^n_{plan}[\Sigma]$. We omit the details for brevity.

Definition 5 assumes that all plans are equal. This is often not the case in a CPS system. To illustrate this issue, 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`. The energy consumption constraints of `BAT` are encoding in listing 7. Figure 4 shows a fragment of the CPS theory that is related to the problem described in this example.

Listing 7: Battery consumption constraints in Δ_{lkas}

```
1  h(active(bat,saving_mode),T):-
2      h(active(cam,advanced_mode),T),
3      h(active(sam,advanced_mode),T).
4  1{h(active(bat,powerful_mode),T);
5      h(active(bat,normal_mode),T)}1:-
6      h(active(cam,basic_mode),T),
7      h(active(sam,basic_mode),T).
8  h(active(bat,normal_mode),T):-
9      h(active(X,advanced_mode),T),
10     h(active(Y,basic_mode),T), X!=Y .
11 :- h(active(bat,M1),T),
12     h(active(bat,M2),T), M1!=M2.
```

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 program $\Pi^p(\Delta_{lkas})$ can generate the following strategies (for $n = 2$):

- α_1= `tOn(b_mode)`
- α_2= `switM(cam,a_mode)` $\cdot$ `switM(sam,a_mode)`
- α_3= `switM(sam,a_mode)` $\cdot$ `switM(cam,a_mode)`
- α_4= `switM(sam,a_mode)` $\cdot$ `tOn(b_mode)`
- α_5= `switM(cam,a_mode)` $\cdot$ `tOn(b_mode)`

All five above mitigation strategies can be used to address the issue raised by the attack. Specifically, the final state of each plan is given below:

- G_{α_1} is $\{$`CAM` $\mapsto$ `b_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `b_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `p_mode` $\}$ or $\{$`CAM` $\mapsto$ `b_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `b_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `n_mode` $\}$
- G_{α_2} and G_{α_3}: $\{$`CAM` $\mapsto$ `a_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `a_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `s_mode` $\}$
- G_{α_4} is $\{$`CAM` $\mapsto$ `b_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `a_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `n_mode` $\}$
- G_{α_5} is $\{$`CAM` $\mapsto$ `a_mode`, `CAM` $\mapsto$ `secure_boot`, `SAM` $\mapsto$ `b_mode`, `SAM` $\mapsto$ `secure_boot`, `BAT` $\mapsto$ `n_mode` $\}$

In each considered state, the statement $X \mapsto P$ denotes that component X is working with property P. For example, `BAT` $\mapsto$ `s_mode` says that the battery is working in saving mode. Observe that the components, though its activation of properties, could affect positively or negatively on the concerns. Furthermore, how a property can affect positively or negatively on the concerns is described in the CPS Ontology (Balduccini et al. 2018). For example, considering example 2, the three components can positively affect the `Integrity` concern if $\{$`CAM` $\mapsto$ $\{$`secure_boot`, `advanced_mode`$\}$, `SAM` $\mapsto$ $\{$`advanced_mode`, `secure_boot`$\}$, `BAT` $\mapsto$ $\{$`powerful_mode`$\}$ hold. For this reason, we will introduce a notion called *likelihood of satisfaction (LoS) of concern* and use them to distinguish mitigation strategies. There are, of course, different ways of looking at the LoS. Our notion relies on the positive impacts of properties on concerns within the system, i.e., property `secure_boot` positively impacts `Integrity`. For a concern c, we denote with $rel^+(c)$ the set of all properties that positively impacting concern c.

Definition 6 (Impact Degree) *Given a CPS ontology, c is a concern and s is a configuration (state) of $\mathscr{S}$, the* positive impact degree *of concern c in state s, denoted by $deg^+(c,s)$, is defined as:*

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

where $rel^+_{sat}(c,s)$ is the set of properties in $rel^+(c)$ which hold in state s

Considering the five final configurations of different mitigation strategies in our example above, we have: $deg^+(Integrity, G^1_{\alpha_1}) = 0.6$, $deg^+(Integrity, G^2_{\alpha_1}) = 0.4$, $deg^+(Integrity, G_{\alpha_2}) = 0.8$, $deg^+(Integrity, G_{\alpha_3}) = 0.8$, $deg^+(Integrity, G_{\alpha_4}) = 0.6$, and $deg^+(Integrity, G_{\alpha_5}) = 0.6$. We also have that $deg^+(availability, \_) = 1$, $deg^+(security, \_) = 1$, $deg^+(trustworthiness, \_) = 1$, etc. Listing 8 shows the ASP program for computing the positive impact degree of a concern following Definition 6, where the predicates `nAllPosCon/3` and `nActPosCon/3` define 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. The predicate `possImpactsPos(Com,P,C)` denotes that the relation between component *Com* and property P is a *positive relation* and impacts positively to concern C. This predicate is generated from CPS ontology.

Listing 8: Computing Positive Impacts Degree

```
1  nAllPosCon(C,N2,T):- concern(C), step(T),
2    N2=#count{P,Com : comp(Com), prop(P),
3    possImpactsPos(Com,P,C), addBy(C,P)}.
4  nActPosCon(C,N1,T):- concern(C), step(T),
5    N1=#count{P,Com : comp(Com), prop(P),
6    possImpactsPos(Com,P,C), addBy(C,P),
7    h(active(Com,P),T)}.
8  deg_pos(C,1,T) :- step(T), concern(C),
9    nAllPosCon(C,0,T).
10 deg_pos(C,N1*100/N2,T) :- concern(C),
11   nActPosCon(C,N1,T),
12   nAllPosCon(C,N2,T),N2!=0.
```

In listing 8, lines 1-3 (4-7) encode the rules for compute `nAllPosCon/3` (`nActPosCon/3`) which represent $rel^{+}(c)$ and $rel^{+}_{sat}(c,s)$, respectively. The other rules calculate the positive impact degree of concern C in step T by defining `deg_pos(C,V,T)` which states that V is the positive impact degree of C at step T. We next define the likelihood of satisfaction of a concern.

Definition 7 (Likelihood of Concern Satisfaction) *Given a CPS $\mathscr{S}$, a state s if $\mathscr{S}$, and a concern c. The likelihood of the satisfaction (LoS) of c in s, denoted by $\varphi_{llh}(c,s)$, is defined by*

$$\varphi_{llh}(c,s) = \begin{cases} deg^{+}(c,s) * \Pi_{x \in sub(c)} \varphi_{llh}(x,s), & \text{if } sub(c) \neq \varnothing \\ deg^{+}(c,s), & \text{if } sub(c) = \varnothing \end{cases} \tag{4}$$

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

Listing 9 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.

Listing 9: Computing Likelihood of Concerns Satisfaction

```
1  order(SC,C,N):- subCo(C,SC),
2    N={SC < SCp : subCo(C,SCp)}.
3  hSubCo(C) :- subCo(C,SC).
4  ¬hSubCo(C):- concern(C), not hSubCo(C).
5  llh_sat_sub(C,1,T) :- step(T), concern(C),
       ¬hSubCo(C).
6  llh_sat(C,N1*N2,T):- step(T), concern(C),
       llh_sat_sub(C,N1,T), deg_pos(C,N2,T).
7  llh_sat_sub_aux(C,0,X,T):- step(T),
8    subCo(C,SC), order(SC,C,0),
9    llh_sat(SC,X,T).
10 llh_sat_sub_aux(C,N,X*Y,T):- step(T),
11   subCo(C,SC), order(SC,C,N),
12   llh_sat(SC,Y,T),
13   llh_sat_sub_aux(C,N-1,X,T).
14 llh_sat_sub(C,X,T):- step(T), concern(C),
15   llh_sat_sub_aux(C,N,X,T),
16   not llh_sat_sub_aux(C,N+1,_,T).
```

In listing 9, the rule on Lines 1-2 creates an ordering between subconcerns of concern C for the computation of

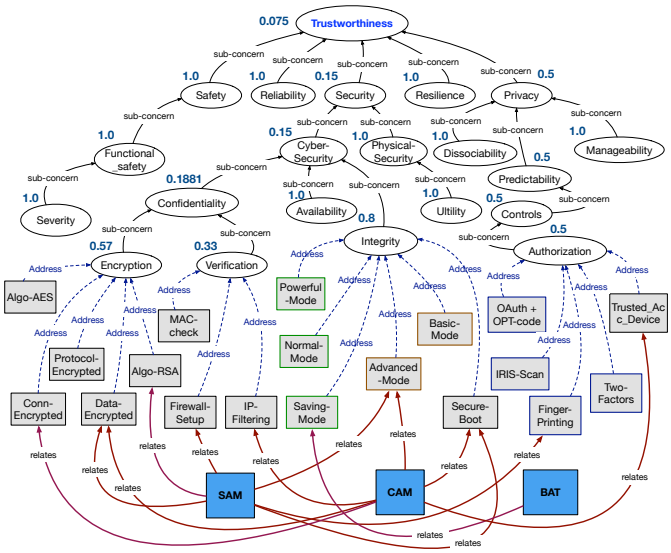

Figure 5: Trustworthiness concern tree with LoCS

`llh_sat(C,N,T)`. Any concern without a subconcern will be computed first (Line 5). Rules on the lines 7-16 compute the LoS of concerns in accordance with the order created by rule on lines 1-2. `llh_sat(C,N,T)` is then computed using Equation 4.

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 possible strategies, there are also the best mitigation strategies which are especially relevant to the `trustworthiness` attribute, where the LoS of `trustworthiness` aspect in final state is maximum. In this figure, the LoS of `trustworthiness` (root concern) is 0.075 (`llh_sat(trustworthiness)=0.075`). 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.

Selecting Most Preferred Mitigation Strategy. 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 top-level concerns ((e.g., `trustworthiness`, `functionality`, `timing`, etc.). Therefore, there are different ways to use this number. We discuss two possibilities:

- *Weighted LoS*: The CPS users (end users, administrators, manufacturers, managers, etc.) specifies the weights W_{fun}, W_{bus}, W_{hum}, W_{tru}, W_{tim}, W_{dat}, W_{bou}, W_{com} and W_{lif} which they would like to assign for the nine aspects: `Functionality`, `Business`, `Human`, `Trustworthiness`, `Timing`, `Data`, `Boundaries`, `Composition` and `Lifecycle` respectively. The weighted LoS of a system $\mathscr{S}$ in state s is then computed by $w(\mathscr{S},s) = llh\_sat(Functionality,s) * W_{fun} + llh\_sat(Business,s) * W_{bus} + llh\_sat(Human,s) * W_{hum} + llh\_sat(Trustworthiness,s) * W_{tru} +$

$llh\_sat(Timing,s) * W_{tim} + llh\_sat(Data,s) * W_{dat} + llh\_sat(Boundaries,s) * W_{bou} + llh\_sat(Composition,s) * W_{com} + llh\_sat(Lifestyle,s) * W_{lif}$.
Under this view, $w(\mathscr{S},n)$, where n denotes the final time step of Π^n_{plan}, can easily be computed. We omit the rule for brevity. To select the most preferred mitigation strategy (highest LoS), we only need to add the following rule:
`#maximize{Sc : scoreLoS(Sc,T), last(T)}.`
to the program, where $scoreLoS(sc,n)$ indicates that $w(\mathscr{S},n) = sc$.

- *Specified Preferences LoS*: An alternative to the weighted LoS of a system is to allow the users to specify a partial ordering of the set of attributes which will be used in identifying the preferred strategies by lexical ordering in accordance to the preferences. For example, assume that the preference ordering is $x_1 > x_2 > x_3 > x_4 > x_5 > x_6 > x_7 > x_8 > x_9$ where $x_i > x_j$ which means that attribute x_i is preferred to the attribute $x_j \neq x_i$ and $x_i \in \{$`Functionality, Business, Human, Trustworthiness, Timing, Data, Boundaries, Composition, Lifecycle`$\}$. As the values of the LoS of aspects in different states in $\mathscr{S}$ behave differently, we write $llh\_sat(x,s) \prec llh\_sat(x,s')$ to denote that state s' is better than s w.r.t the attribute x. The most preferred final configuration of mitigation strategies is defined via a lexicographic ordering: $p \prec p'$ if there is $1 \leq i \leq 9$ such that $llh\_sat_{x_j}(p) = llh\_sat_{x_j}(p')$ for $j < i$ and $llh\_sat_{x_i}(p) \prec llh\_sat_{x_i}(p')$. This can easily be implemented using the priority level in **clingo**.
$\#maximize\{llh\_sat(x_1)@9\}$.
$\#maximize\{llh\_sat(x_2)@8\}$.
$\#maximize\{llh\_sat(x_3)@7\}$.
$\#maximize\{llh\_sat(x_4)@6\}$.
$\#maximize\{llh\_sat(x_5)@5\}$.
$\#maximize\{llh\_sat(x_6)@4\}$.
$\#maximize\{llh\_sat(x_7)@3\}$.
$\#maximize\{llh\_sat(x_8)@2\}$.
$\#maximize\{llh\_sat(x_9)@1\}$.
For a concrete example, assume that the ordering preference is $Trustworthiness > Functionality > Timing > Human > Composition > Data > Lifecycle > Business > Boundaries$, the corresponding ASP encoding is as follow where T_{last} is final planning step:
$\#maximize\{V_T@9 : llh\_sat(trustworthiness,V_T,T_{last})\}$.
$\#maximize\{V_F@8 : llh\_sat(functionality,V_F,T_{last})\}$.
$\#maximize\{V_{TI}@7 : llh\_sat(timing,V_{TI},T_{last})\}$.
$\#maximize\{V_H@6 : llh\_sat(human,V_H,T_{last})\}$.
$\#maximize\{V_C@5 : llh\_sat(composition,V_C,T_{last})\}$.
$\#maximize\{V_D@4 : llh\_sat(data,V_D,T_{last})\}$.
$\#maximize\{V_L@3 : llh\_sat(lifecycle,V_L,T_{last})\}$.
$\#maximize\{V_B@2 : llh\_sat(business,V_B,T_{last})\}$.
$\#maximize\{V_{BO}@1 : llh\_sat(boundaries,V_{BO},T_{last})\}$.

Conclusions, Related Work, and Discussion

The paper presents a precise definition of a CPS, which, in conjunction with the CPS Framework, allows for the representing and reasoning of various problems that are of interest in the study of CPS. Specifically, the paper presents an ASP based solution for the verification of various concerns in a CPS. It discusses the problem of identifying conflict concerns in a CPS system. Finally, it discusses different methods for selecting a preferred mitigation strategy. To the best of our knowledge, all of these contributions are new to the research in Cyber-Physical System.

Due to space constraints, we limit our overview of related work to what we consider 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, Kim, and Trivedi 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 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, Gelfond, and Rushton 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).

The proposed methodologies in this 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`) that can deal with millions of atoms, incomplete information, default reasoning, and features that allow ASP to interact with constraint solvers and external systems.

References

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

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.

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.

Baral, C.; Gelfond, M.; and Rushton, N. 2009. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming* 9(1):57–144.

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. 2007. Answer set programming for the semantic web. In Dahl, V., and Niemelä, I., eds., *Logic Programming, 23rd International Conference, ICLP 2007, Porto, Portugal, September 8-13, 2007, Proceedings*, volume 4670 of *Lecture Notes in Computer Science*, 23–26. Springer.

Gelfond, M., and Lifschitz, V. 1990. Logic programs with classical negation. In Warren, D., and Szeredi, P., eds., *Logic Programming: Proceedings of the Seventh International Conference*, 579–597.

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).

Griffor, E.; Greer, C.; Wollman, D. A.; and Burns, M. J. 2017a. Framework for cyber-physical systems: volume 1, overview.

Griffor, E.; Greer, C.; Wollman, D. A.; and Burns, M. J. 2017b. Framework for cyber-physical systems: Volume 2, working group reports.

Marek, V., and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-year Perspective*, 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.

Nguyen, T. H.; Potelli, E.; and Son, T. C. 2018. Phylotastic: An experiment in creating, manipulating, and evolving phylogenetic biology workflows using logic programming. *Theory and Practice of Logic Programming* 18(3-4):656–672.

Nguyen, T. H.; Son, T. C.; and Pontelli, E. 2018. Automatic web services composition for phylotastic. In *Practical Aspects of Declarative Languages - 20th International Symposium, PADL 2018, Los Angeles, CA, USA, January 8-9, 2018, Proceedings*, 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.

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.

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.

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.

Wollman, D. A.; Weiss, M. A.; Li-Baboud, Y.-S.; Griffor, E.; and Burns, M. J. 2017. Framework for cyber-physical systems: Volume 3, timing annex.