Reasoning about Trustworthiness in Cyber-Physical Systems Using Ontology-Based Representation and ASP

Thanh Hai Nguyen[1 [0000−0001−9996−4720]], Tran Cao Son[1 [0000−0003−3689−8433]], Matthew Bundas[1], Marcello Balduccini[2], Kathleen Campbell Garwood[2], and Edward R. Griffor[3]

[1] New Mexico State University, Las Cruces, USA
{thanhnh,stran,bundasma}@nmsu.edu
[2] St. Joseph's University, Philadelphia, USA
{mbalducc,kcampbel}@sju.edu
[3] National Institute of Standards and Technology
edward.griffor@nist.gov

Abstract. This paper presents a framework for reasoning about trustworthiness in cyber-physical systems (CPS) that combines ontology-based reasoning and answer set programming (ASP). It introduces a formal definition of CPS and several problems related to trustworthiness of a CPS such as the problem of identification of the most vulnerable components of the system and of computing a strategy for mitigating an issue. It then shows how a combination of ontology based reasoning and ASP can be used to address the aforementioned problems. The paper concludes with a discussion of the potentials of the proposed methodologies.

Keywords: CPS · Ontology · ASP · Knowledge Representation · Reasoning · Planning

1 Introduction

Cyber-physical systems (CPS) are sophisticated systems that include engineered interacting networks of physical and computational components. These highly interconnected and integrated systems provide new functionalities to improve quality of life and enable technological advances in critical areas, such as personalized health care, emergency response, traffic flow management, smart manufacturing, defense and homeland security, and energy supply and use. In addition to CPS, there are other terms (e.g., Industrial Internet, Internet of Things (IoT), machine-to-machine (M2M), smart cities) that describe similar or related systems and concepts. CPS and related systems (including the IoT and the Industrial Internet) are widely recognized as having great potential to enable innovative applications and impact multiple economic sectors in the worldwide economy. Evidence of the promise of CPS is evident in autonomous vehicles, intelligent buildings, smart energy systems, robots, and smart medical devices. Realizing the promise of CPS requires interoperability among heterogeneous components and systems, supported by new reference architectures using shared vocabularies and definitions. Addressing the challenges and opportunities of CPS requires broad consensus in foundational concepts, and a shared understanding of capabilities and technologies

unique to CPS. The following example describes the Elevator Monitoring scenario, a simple concrete CPS with its concerns and requirements:

Example 1. Elevator Monitoring

> *The system consists of an* elevator cabin *(*`EC`*) which is controlled by a control panel (*`CP`*). The panel communicates with the* elevator receiver *(*`ER`*), and calls the elevator to the required floor. The receiver communicates with the elevator's* pulley function *(*`PF`*), which releases/clenches and allows the elevator to move down/up along the rope and pulley to the designed floor. The* elevator sensor & camera *(*`ESCam`*) will detect the number of passengers in the cabin. If passenger number $>$ 11, the sensor (*`ESCam`*) will communicate to the receiver (*`ER`*) to stop the elevator from moving and flashes a warning light indicating an overload.*

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) [11], which provides a principled design and analysis methodology for CPS that is intended to be applicable across all relevant domains of expertise. Preliminary research aimed at addressing questions related to the trustworthiness of a CPS utilizing the CPSF was presented in [2], which introduced a CPSF *ontology* capturing the keys elements of CPSF 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).

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. This paper builds upon and extends the work from [2] in a number of important directions. First of all, it introduces precise mathematical formalization of CPS and of a variety of important questions related to the trustworthiness of the components of a CPS and of a CPS as a whole: (*i*) what are the most vulnerable (least trustworthy) components in a CPS? (*ii*) how to mitigate (or prevent) an attack on a component? (*iii*) what is the probability that a mitigation strategy will succeed? Of the above, (i) and (iii) have also not been addressed before in the context of CPSF. Additionally, the paper establishes a clear relationship between CPSF and the formalization of CPS. Finally, it demonstrates how the combination of ontology-based reasoning and ASP, together with a suitable extension of the CPS ontology from [2], can be leveraged for the development of solutions to the above questions. Hybrid reasoning aims at combining the best of both worlds, particularly for scalability in practical applications. Take for instance a large CPS such as an aircraft carrier. Answering the above questions requires sophisticated reasoning, which typically struggles with scalability. By combining the system scalability and describability of ontology-based reasoning with the flexibility of

ASP, one can leverage the former to extract inference-rich information for the subsystems relevant to the task at hand, on which ASP-based reasoning can then carry out sophisticated reasoning in a more focused, efficient way.

2 Background

2.1 CPS Framework (CPSF), CPS Ontology, and Representation

CPSF defines a set of *concerns* related to a CPS such as *Trustworthiness, Functionality, Safety, etc*. These concerns are organized in a multi-rooted, tree-like structure (a "forest" in graph theory), where branching corresponds to the *decomposition of concerns*. Each tree is called a concern tree. The concerns at the roots of this structure, the highest level concerns, are called *aspects*. *Properties*[4] are features of a CPS that address a certain concern, e.g. encrypted traffic might address the privacy concern (see Table 1 for details). Specifically, the *Trustworthiness* aspect is related to the trustworthiness of CPS, including concerns such as *security*, *privacy*, *safety*, *reliability*, and *resilience*. In this paper, we adopt the definition of trustworthiness from the NIST CPSF, 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.*

The *CPS Ontology*, introduced in [2], defines concepts and individuals related to *Trustworthiness*, and the relationships between them (e.g., sub-class of, has-subconcern, etc.). Figure 1, excluding the nodes labeled `CP`, `ER`, `PF`, `EC` and `ESCam` and links labeled "relates", 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[5]) and the Web Ontology Language (OWL[6]) 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 by a RDF-triple (*subject*, *predicate*, *object*). Table 1 lists the main classes and relationships in the CPS ontology. Most of the class names are self-describing. Intuitively, a relationship between a property and a concern indicates that the property positively (or negatively) affects the associated concern when it is true (or false). 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`.

The notion of *satisfaction* of a concern is also introduced in [2]. In the simplest case, a concern is satisfied when all properties related to it, directly or indirectly (via the subconcern relation) must be *true*. Inference can then be applied to propagate the satisfac-

[4] There is an unfortunate clash of terminology between OWL properties and CPS properties. Throughout this paper, the intended meaning of the word 'property' can be identified from the context of its use.

[5] https://www.w3.org/TR/rdf-concepts/

[6] https://www.w3.org/TR/owl-features/

tion, or lack thereof, of concerns and properties throughout a concern tree. For example, given a concern that is not *satisfied*, one can leverage relations `has-subconcern` and/or `addressed-by` to identify the sub-concerns and/or properties that are not satisfied, either directly or indirectly.

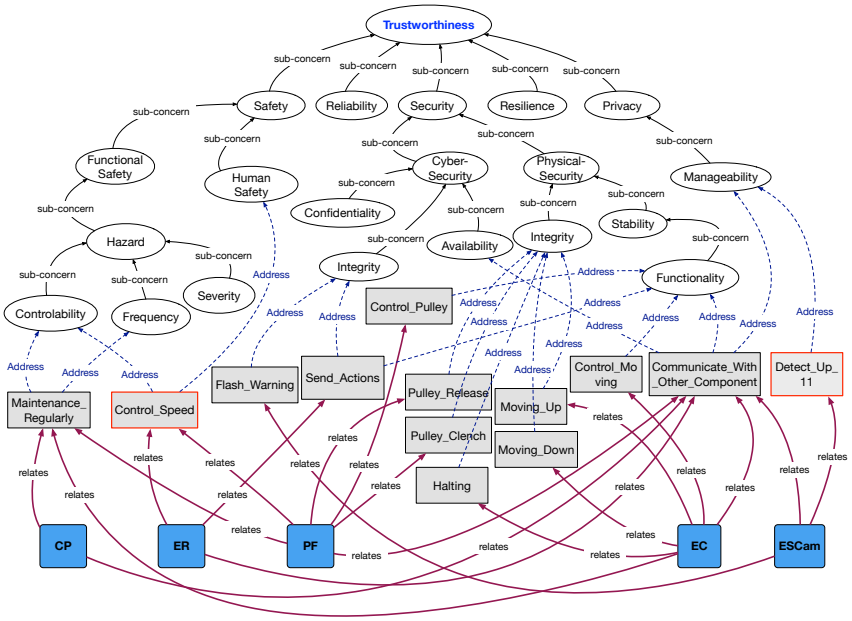

Fig. 1: A Fragment of the Trustworthiness Concern Ontology

2.2 Answer Set Programming

Answer set programming (ASP) [12,16] 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 a_i's, and b_i's are atoms of a propositional language[7] 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^- 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 b* in the bodies of the remaining rules. Given an interpretation I,

[7] For simplicity, we often use first order logic atoms in the text which represent all of its 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**
hasSubConcern	The object property represents the `has-subconcern` relationship between the concerns.
addrConcern	The object property represents the `addressed-by` relation between a concern and a property.

Table 1: Main components of the CPS Ontology

observe that the program Π^I is a program with no occurrence of *not b*. An interpretation I is an *answer set* [8] of Π if I is the least model (wrt. $\subseteq$) of Π^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. Efficient and scalable ASP solvers are available[8].

3 OA4CPS: A Hybrid Reasoner for CPS

We now describe OA4CPS, a hybrid reasoning system for CPS, which combines ontology-based reasoning with ASP. As a prototypical system for reasoning about CPS, OA4CPS focuses on the trustworthiness aspect of CPS. The techniques developed in this paper can be easily extended to cover other aspects described in CPSF. Figure 2 depicts the overview of OA4CPS. The input of OA4CPS is a *CPS Theory Specification* $\Delta = (\mathscr{S}, I)$ (details follow shortly). The *translator* combines Δ with the relevant fragment Ω of the CPS Ontology obtained from a suitable query. The *translator* then produces an ASP program $\Pi(\Delta, n)$ where n is an integer, denoting the time horizon of interest for reasoning tasks over Δ. $\Pi(\Delta, n)$ is utilized by the *reasoner* to answer questions related to the trustworthiness of the system. The answers generated by the *reasoner* can be presented to the users in different formats (e.g., sunburst chart, hierarchy, etc.) by a *visualizer*. The focus of this section will be on the *reasoner* and how CPS specifications are translated

[8] See, e.g., https://potassco.org/clingo/.

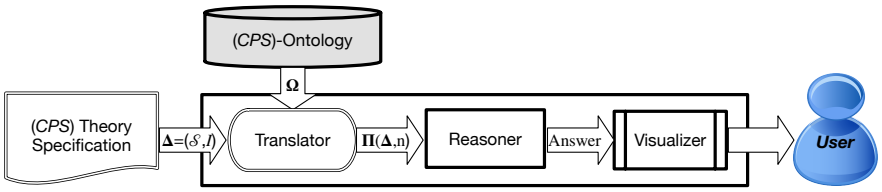

Fig. 2: Overall architecture of OA4CPS

to ASP programs for use with the *reasoner*. Details about the implementation of the translator and the visualizer are given in the next section.

3.1 From CPS Theory Specification to ASP Encoding

We view a CPS as consisting of *interacting cyber and physical components*, which affect properties defined in the CPS ontology, e.g. trustworthiness of the system. For this reason, we extend the CPS ontology with a `component` class, a `relates` relationship which represents the connection between components and properties in the CPS ontology, and an `action` class which allows for the representation and reasoning about actions in CPS. In this paper, we employ the action ontology described in [15] where the action profile (preconditions, effects, input, output, etc.) is described by an OWL ontology and reasoning about effects of actions is formalized via action language $\mathscr{B}$[10]. From now on, "CPS ontology" refers to the CPS ontology in [2] with these features.

In the CPS ontology, a CPS is characterized by a set of concerns (represented as ontology individuals), a set of properties, and a relation linking properties to the concerns they address. Formally, a *CPS domain* $\mathscr{S}$ is a tuple (C,A,F,R) where C 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 of $\mathscr{S}$; R is a set of relations that maps each $c \in C$ to a set of properties $R(c)$ defined in the CPS ontology. A *state* s of $\mathscr{S}$ is an interpretation[9] of F. A *CPS theory* is a pair $(\mathscr{S},I)$ where $\mathscr{S}$ is a CPS domain and I is a state representing the *initial configuration* of $\mathscr{S}$. The execution of actions changes the states, e.g., the truth values of the properties, thereby affecting the trustworthiness of the system. As such, we can utilize the framework for reasoning about actions and changes to reason about the trustworthiness of a CPS. In this paper we adopt a hybrid approach to reasoning that leverages the power of ontology-based reasoning for drawing (time-independent) inferences about a CPS and its components and the flexibility of ASP for non-monotonic reasoning and reasoning about actions and change. For this reason, the translator takes in input a CPS theory $\Delta = (\mathscr{S},I)$, queries the CPS ontology for relevant information, and produces the ASP program $\Pi(\Delta,n)$ discussed earlier. We use ASP as it is well-known for working with ontologies (e.g., [4,7,15,14]). Due to the space limitation, we will illustrate the translation from Δ to $\Pi(\Delta,n)$ using a simplified CPS $\Delta_{Ele} = (\mathscr{S}_{Ele}, I_{Ele})$ for controlling an elevator in Example 1. We define $\Pi(\Delta_{Ele},n) = \Pi(\Omega) \cup \Pi_d(\mathscr{S}_{Ele}) \cup \Pi(I_{Ele}) \cup \Pi_n(\mathscr{S}_{Ele})$. Each component of $\Pi(\Delta_{Ele})$ is defined next.

[9] In the following, we will follow the 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.

CPS Ontology. Let Ω be obtained by querying the CPS ontology for the task at hand. $\Pi(\Omega)$ is the ASP encoding of Ω and includes representations of the classes, individuals, and relationships as well as the supporting rules. Listing 1.1 depicts selected rules of $\Pi(\Omega)$.

Listing 1.1: $\Pi(\Omega)$ *(selected rules)*

```
class(concern). class(aspect). class(property). class(action).
subClass(aspect,concern). concern(X) :- aspect(X).
aspect(trustworthiness). isInstOf(trustworthiness,aspect).
concern(security). isInstOf(security,concern).
concern(cybersecurity). isInstOf(cybersecurity,concern).
concern(integrity). isInstOf(integrity,concern).
...
subCo(cybersecurity,integrity). subCo(security,cybersecurity).
subCo(trustworthiness,security).
...
prop(flash_warning). isInstOf(flash_warning,property).
addBy(integrity,flash_warning).
...
subClass(X,Y):-subClass(Z,Y), subClass(X,Z).
...
```

Components, Fluents, and Relations. The components of $\mathscr{S}_{Ele}$ are `EC`, `CP`, `ER`, `PF`, and `ESCam`. They are related to various properties defined in the CPS Ontology, e.g., `CP` is related to `maintenance_regularly`, `communicate_with_other`, etc. These components and relationships are depicted in the bottom part of Fig. 1, where squares (links labeled *relates*) represent components and (relationships). A component X is operating in different modes, each mode is a property P; this is denoted with a fluent $im(X,P)$ ('im' stands for 'in mode'). The *relates* relationship is translated into predicate of the form $rel(X,P)$. The sets of components, fluents, and relations of $\mathscr{S}_{Ele}$ are translated into a set of ASP facts, denoted by $\Pi_d(\mathscr{S}_{Ele})$, as follows:

Listing 1.2: $\Pi_d(\mathscr{S}_{Ele})$

```
comp(cp). comp(er). comp(pf). comp(ec). comp(escam).
fluent(im(escam,flash_warning)).fluent(im(er,control_speed)).
fluent(im(escam,detect_up_11)).fluent(im(ec,halting)).
...
rel(cp,maintenance_regularly). rel(er,send_actions).
rel(er,communicate_with_other_comp). rel(er,control_speed).
rel(pf,control_speed). rel(ec,control_moving).
rel(escam,flash_warning). rel(escam,detect_up_11).
...
```

Initial Configuration. Initially, properties associated with $\mathscr{S}_{Ele}$ can be *true* ($\top$) or *false* ($\bot$). The initial state is translated into a set of facts of the form $o(x,\top)$ or $o(x,\bot)$ for $x \in I$ or $x \notin I$ (Lines 1-5, Listing 1.3). An initial configuration I_{Ele} of the values of $\mathscr{S}_{Ele}$ is done by two rules that define the predicate `h` and $\neg$`h` at the time step 0 (Lines 6-7, Listing 1.3). Line 8 describes which properties having available patch.

Listing 1.3: $\Pi(I_{Ele})$

```
o(maintenance_regularly,⊤). o(control_speed,⊥).
o(flash_warning,⊤). o(send_actions,⊤). o(pulley_release,⊤).
```

```
o(pulley_clench,⊤). o(control_pulley,⊤). o(control_moving,⊤).
o(moving_up,⊤). o(moving_down,⊤). o(halting,⊤).
o(detect_up_11,⊥). o(communicate_with_other_comp,⊤).
h(P,0) :- o(P,⊤), prop(P).
-h(P,0) :- o(P,⊥), prop(P).
availablePatch(control_speed).
```

Actions and Constraints. Actions change the status of the properties in $\mathscr{S}_{Ele}$ and constraints describe dependencies among CPS properties. For brevity, we focus on a few affecting the trustworthiness of the system. For simplicity of presentation, we assume that each property P can be changed by the action $tOn(P)$ or $tOff(P)$. In addition, there is an action $patch(P)$—a special action that could help restore P to the value *true* if the patch of P is available (`availablePatch(P)` holds). Furthermore, when `escam` detects more than 11 passengers, it sets the property *overloaded* to be true. Actions and constraints are translated to ASP rules accordingly following Table 2. In these rules, T is an integer ranging between 0 and n, $h^*(l,t)$ stands for $h(l,t)$ if l is a fluent and for $-h(f,t)$ if l is a negated fluent $\neg f$.

Statement type	**ASP Translation**
(0) *Action declaration*	$\texttt{action(a)}$.
(1) *Executability condition*	$\texttt{exec(a,T)} :- \texttt{step(T)}, \texttt{h}^*(\texttt{p}_1,\texttt{T}),\ldots,\texttt{h}^*(\texttt{p}_n,\texttt{T})$.
(2) *Dynamic law*	$\texttt{h}^*(\texttt{f},\texttt{T}+1) :- \texttt{step(T)}, \texttt{occurs(a,T)}, \texttt{h}^*(\texttt{p}_1,\texttt{T}),\ldots,\texttt{h}^*(\texttt{p}_n,\texttt{T})$.
(3) *State constraint*	$\texttt{h}^*(\texttt{f},\texttt{T}) :- \texttt{step(T)}, \texttt{h}^*(\texttt{p}_1,\texttt{T}),\ldots,\texttt{h}^*(\texttt{p}_n,\texttt{T})$.
(4) *Inertial axiom*	$\texttt{h(f,T+1)} :- \texttt{step(T)},\texttt{h(f,T)}, \texttt{not } -\texttt{h(f,T+1)}$. $-\texttt{h(f,T+1)} :- \texttt{step(T)}, -\texttt{h(f,T)}, \texttt{not h(f,T+1)}$.

Table 2: Formulas of an action a

Listing 1.4: $\Pi_n(\mathscr{S}_{Ele})$

```
action(tOn(X)) :- prop(X).   action(tOff(X)) :- prop(X).
action(patch(X)) :- prop(X).
h(im(ec,halting),T) :- h(im(escam,overloaded),T), step(T).
exec(tOn(X),T)  :- -h(X,T), prop(X), step(T).
exec(tOff(X),T) :- h(X,T), prop(X), step(T).
exec(patch(X),T):- prop(X), availablePatch(X),-h(X,T), step(T).
h(X,T+1)  :- occurs(tOn(X),T), step(T).
-h(X,T+1) :- occurs(tOff(X),T), step(T).
h(X,T+1)  :- occurs(patch(X),T), step(T).
...
```

3.2 Queries Related to Trustworthiness

Given a CPS theory $\Delta = (\mathscr{S}, I)$, [11] list several questions related to the trustworthiness of Δ and its components. We are interested in the following questions:

- *Question #1* (**Q_1**): *What is the most/least trustworthy of component in Δ?* This is one of most important questions in CPSF, as discussed in [11].
- *Question #2* (**Q_2**): *What can be done to mitigate an issue?*
- *Question #3* (**Q_3**): *Which mitigation strategies have the best chance to succeed?* This question is a generalization of (**Q_2**).

3.3 Queries Answering Using $\Pi(\Delta, n)$

In this section, we use $\Pi(\Delta, n)$ to answers $\mathbf{Q}_1 - \mathbf{Q}_3$. For each query $\mathbf{Q}_i$, we develop a program $\Pi_i(\Delta)$ such that answer sets of $\Pi(\Delta, n) \cup \Pi_i(\Delta)$ are solutions of the query.

Most/Least Trustworthy Components. A component $c \in C$ might be related to many concerns through the properties in $R(c)$, whose truth values depend on the state s of the system. If a property $p \in R(c)$ is false in s, it might negatively affect a concern and affect the trustworthiness of the system. Likewise, if p is true in s, it will positively affect the concern and helps strengthen the trustworthiness of the system.

Assume that all concerns and properties are equally important, we could say that a component $c \in C$ is less trustworthy than a component $c' \in C$ if the number of concerns negatively associated to c is greater than the number of concerns negatively associated to c. Alternatively, we can characterize the trustworthiness of a component by the numbers of concerns that are positively or negatively affected by the component and use them in evaluating the components. In this paper, we adopt this second alternative. Since actions change the truth values of properties, we will define these numbers wrt. a state, i.e., for each $x \in C$ and a state s of $\mathscr{S}$, we define $twc^+(x,s)$ ($twc^-(x,s)$) as the number of concerns in Ω that are positively (negatively) impacted by component x in a state s.

Note that the direct relationship between a concern c and a property p in Ω is translated into `addBy(c,p)` in $\Pi(\Omega)$. We say that a property p addresses a concern c if (*i*) p directly addresses c; or (*ii*) there exists some subconcern c' of c that is addressed by p. As such, a concern c is positively (negatively) impacted by a component x at the step t if x is related to a property p that addresses c and p holds (does not hold) at t. Formally, for a state s of $\mathscr{S}$: $twc^+(x,s) = \Sigma_{p \in R(x), p \in s} |\{c \mid addBy(c,p)\}|$ and $twc^-(x,s) = \Sigma_{p \in R(x), p \notin s} |\{c \mid addBy(c,p)\}|$. Next, we propose an ordering among the components using the two numbers. Let $\delta(x,s) = twc^+(x,s) - twc^-(x,s)$.

Definition 1. *For a CPS $\mathscr{S} = (C, A, F, R)$, $c_1, c_2 \in C$, and state s of $\mathscr{S}$,*

- *c_1 is* more trustworthy *than c_2 in s, denoted by $c_1 \succ_s c_2$ (or c_2 is* less trustworthy *than c_1, denoted by $c_2 \prec_s c_1$), if $\delta(c_1,s) > \delta(c_2,s)$; and*
- *c_1 is* as trustworthy as *c_2 in s, denoted by $c_1 \sim_s c_2$, if $\delta(c_1,s) = \delta(c_2,s)$.*

$c_1 \succeq_s c_2$ denotes that $c_1 \succ_s c_2$ or $c_1 \sim_s c_2$. c is the most (least) trustworthy component of $\mathscr{S}$ in s if $c \succeq_s c'$ ($c' \succeq_s c$) for every $c' \in C$.

Proposition 1. *Let $\mathscr{S} = (C, A, F, R)$ 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.*

Proposition 1 shows that $\succeq_s$ has min/maximal elements, i.e., least/most trustworthy components of a system always exist. The program $\Pi_1(\mathscr{S})$ for computing these components is listed below.

Listing 1.5: $\Pi_1(\mathscr{S})$: Computing the most/least trustworthy component

```
1 addBy(C,P) :- prop(P), addBy(O,P), subCo(C,O).
2 tw_p(P,N)  :- N=#count{C:addBy(C,P)}, prop(P).
3 pos(X,P,T) :- comp(X), prop(P), step(T), rel(X,P), h(P,T).
```

```
4  neg(X,P,T) :- comp(X), prop(P), step(T), rel(X,P),-h(P,T).
5  twcp(X,TW,T) :- TW=#sum{N,P:tw_p(P,N), prop(P), pos(X,P,T)},
6          comp(X), step(T).
7  twcn(X,TW,T) :- TW=#sum{N,P:tw_p(P,N), prop(P), neg(X,P,T)},
8          comp(X), step(T).
9  d(X,D,T) :- comp(X), step(T), twcp(X,TWp,T), twcn(X,TWn,T),
10         D=TWp-TWn.
11 most(X,T) :- comp(X), step(T), d(X,M,T), M == #max{N:d(_,N,T)}.
12 least(X,T):- comp(X), step(T), d(X,M,T), M == #min{N:d(_,N,T)}.
```

In $\Pi_1(\mathscr{S})$, $tw\_p(p,n)$ says that p impacts n concerns. $pos(x,p,t)$ ($neg(x,p,t)$) states that x has a property p which positively (negatively) impacts the concerns related to it at the step t. $twcp(x,tw,t)$ ($twcn(x,tw,t)$) states that the number of concerns positively (negatively) impacted by x at step t is tw. $\#\mathtt{count}\{\mathtt{C}:\mathtt{addBy}(\mathtt{C},\mathtt{P}),\mathtt{prop}(\mathtt{P})\}$ is an aggregate atom and encodes the cardinality of the set of all concerns addressed by P. Similarly, $\#\mathtt{sum}\{...\}$, $\#\mathtt{max}\{...\}$, and $\#\mathtt{min}\{...\}$ are aggregate atoms and are self-explanatory.

Proposition 2. *For a CPS theory $\Delta = (\mathscr{S}, I)$ and an answer set S of $\Pi(\Delta,n) \cup \Pi_1(\mathscr{S})$, $twc^+(x,s_t) = k$ iff $twcp(x,k,t) \in S$ and $twc^-(x,s_t) = k$ iff $twcn(x,k,t) \in S$ where $s_t = \{h(f,t) \mid h(f,t) \in S\}$. Furthermore, x is a most (least) trustworthy component in s_t iff $most(x,t) \in S$ ($least(x,t) \in S$).*

The proposition confirms that the program correctly computes the values of $twc^+(x,s)$ and $twc^-(x,s)$ as well as the most (least) component of $\mathscr{S}$ in a state. Its proof follows immediately from the definition of the predicate *addBy* 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 $\Pi(\Delta) \cup \Pi_1(\mathscr{S})$ and use Proposition 2.

Example 2. For Δ_{Ele}, we can easily see that (from Figure 1) $tw\_p(\text{control\_speed}, 6)$ and $tw\_p(\text{flash\_warning}, 4)$, etc. belong to any answer set of $\Pi(\Delta_{Ele}) \cup \Pi_1(\mathscr{S}_{Ele})$. Similar atoms are present to record the number of concerns affected by different properties. Furthermore, $twcp(cp,15,0)$, $twcn(cp,0,0)$, $twcp(er,16,0)$, $twcn(er,6,0)$, $twcp(pf,28,0)$, $twcn(pf,6,0)$, $twcp(ec,32,0)$, $twcn(ec,0,0)$, $twcp(escam,13,0)$, and $twcn(escam,3,0)$ belong to any answer set of $\Pi(\Delta_{Ele}) \cup \Pi_1(\mathscr{S}_{Ele})$: EC is the most trustworthy component; ER and $ESCam$ are 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. On the other hand, $twc^+(x,s)$ and $twc^-(x,s)$ can be used in different ways such as via the lexicographic order to define an ordering that prefers the number of positively impacted concerns over the negatively ones (or vise versa). Any ordering based on $twc^+(x,s)$ and $twc^-(x,s)$ could easily be implemented using ASP. Last but not least, 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., [9]).

Generating Mitigation Strategies. Let $\mathscr{S} = (C,A,F,R)$ be a CPS domain and s be a state of $\mathscr{S}$. A concern c is *satisfied* in a state s if all properties addressing c are true

in s and all sub-concerns of c are *satisfied* in s; otherwise, it is unsatisfied. Mitigating an issue is therefore equivalent to identifying a plan that suitably changes the state of properties related to it. The mitigation problem can then be defined as follows:

Definition 2. *Let $\mathscr{S} = (C, A, F, R)$ be a CPS system and s be a state of $\mathscr{S}$. Let Σ be a set of concerns. A mitigation strategy addressing Σ is a plan $[a_1, \ldots, a_k]$ whose execution in s results in a state s' such that for every $c \in \Sigma$, c is satisfied in s'.*

As planing can be done using ASP, the mitigation problem can be solved using the following code:

Listing 1.6: $\Pi_2(\mathscr{S})$: Computing Mitigation Strategy for concern c

```
1  -h(sat(C),T) :- addBy(C,P),-h(P,T).
2  -h(sat(X),T) :- subCo(X,Y), not h(sat(Y),T).
3  -h(sat(X),T) :- subCo(X,Y),-h(sat(Y),T).
4  h(sat(C),T)  :- not -h(sat(C),T), concern(C).
5  1{occurs(A,T):action(A)}1 :- step(T), T<n.
6  :- occurs(A,T), not exec(A,T).
7  :- not h(sat(c),n), concern(c).
```

The first four rules are for reasoning about the satisfaction of concerns (see also [2]): $\texttt{h}(\texttt{sat}(\texttt{C}),\texttt{T})$ states that concern C is satisfied at the time step `T`. The first rule states that C is not addressed if some of its properties is false. The next two rules propagate the unsatisfaction of a concern from its subconcern. Finally, a concern is satisfied if it cannot be proven that it is unsatisfied. In line 5, the rule containing the atom $1\{\texttt{occurs}(\texttt{A},\texttt{T}) : \texttt{action}(\texttt{A})\}1$ —a *choice atom*—is used to generate the action occurrences and says that at any step T, exactly one action must occur. The second to last 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 $\Pi_2(\mathscr{S})[\Sigma]$ be the program obtained from $\Pi_2(\mathscr{S})$ by replacing its last rule with the set $\{:\!-\texttt{not}\ \ \texttt{h}(\texttt{sat}(\texttt{c}),\texttt{n}).\mid \texttt{c} \in \Sigma\}$. Based on the results in answer set planning, we have:

Proposition 3. *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 $\Pi(\Delta, n) \cup \Pi_2(\mathscr{S})[\Sigma]$ has an answer set S such that occurs$(a_i, i) \in S$ for every $i = 0, \ldots, n-1$.*

We conclude the section with a brief discussion on possible changes to $\Pi_2(\mathscr{S})$ that might be useful in certain situations and can easily be implemented in ASP. Observe that the execution of an action might change the state of some properties between step 0 and n or might result in some concerns becoming unsatisfied. To prevent this, the following rule can be added to $\Pi_2(\mathscr{S}[\Sigma])$: $:\!-\texttt{h}(\texttt{sat}(\texttt{C}),0), -\texttt{h}(\texttt{sat}(\texttt{C}),\texttt{T}), \texttt{T} > 0$. which says that if C is satisfied at 0 then it should not be unsatisfied at any step > 0.

Example 3. Consider again Δ_{Ele}. Assume that `control_speed` and `detect_up_11` are initially false (figure 1), leading to many unsatisfied concerns (e.g., `Safety`, `Privacy`) and affecting the `Trustworthiness` of the system. In this situation, `tOn(control_speed)`, `patch(control_speed)` and `tOn(detect_up_11)` can be used to repair these properties (action `patch(detect_up_11)` is not executable). $\Pi_2(\mathscr{S})$ have four mitigation strategies of length 2:

- $\alpha_1 = \texttt{tOn}(\texttt{detect\_up\_11}) \cdot \texttt{tOn}(\texttt{control\_speed})$

- $\alpha_2 = \texttt{tOn}(\texttt{detect\_up\_11}) \cdot \texttt{patch}(\texttt{control\_speed})$
- $\alpha_3 = \texttt{tOn}(\texttt{control\_speed}) \cdot \texttt{tOn}(\texttt{detect\_up\_11})$
- $\alpha_4 = \texttt{patch}(\texttt{control\_speed}) \cdot \texttt{tOn}(\texttt{detect\_up\_11})$

Best Mitigation Strategies. Mitigation strategies computed in the previous subsection assumed 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 **success_with** v **if** X 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 $pr : A \times States \rightarrow [0,1]$ 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. Such concepts can easily be added to the CPS ontology and information about the actions in the theory can easily be added to the theory specification.

In this setting, the execution of a sequence of actions (or a strategy) $[a_0, \ldots, a_{n-1}]$ 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}. Problem $\mathbf{Q_3}$ focuses on identifying strategies with the maximal probability of success. Due to the space limitation, we will only briefly discuss how this problem can be addressed. Let $\Pi_3(\mathscr{S})$ be the program $\Pi_2(\mathscr{S})$ extended with the following rules:

- for each statement a **success_with** v **if** $p_1, \ldots, p_n$, the two rules:
  $\texttt{pr}(\texttt{a}, \texttt{v}, \texttt{T}) :- \texttt{h}^*(\texttt{p}_1, \texttt{T}), \ldots, \texttt{h}^*(\texttt{p}_\texttt{n}, \texttt{T}).$
  $\texttt{dpr}(\texttt{a}, \texttt{T}) :- \texttt{h}^*(\texttt{p}_1, \texttt{T}), \ldots, \texttt{h}^*(\texttt{p}_\texttt{n}, \texttt{T}).$
  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: $\texttt{pr}(\texttt{A}, 1, \texttt{T}) :- \texttt{exec}(\texttt{A}, \texttt{T}), \texttt{not dpr}(\texttt{A}, \texttt{T}).$
  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:
  $\texttt{prob}(1, 0).$
  $\texttt{prob}(\texttt{U} * \texttt{V}, \texttt{T} + 1) :- \texttt{prob}(\texttt{U}, \texttt{T}), \texttt{occurs}(\texttt{A}, \texttt{T}), \texttt{pr}(\texttt{A}, \texttt{V}, \texttt{T}).$ 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.

We have that if $[a_0, \ldots, a_{n-1}]$ and S is an answer set of $\Pi(\Delta) \cup \Pi_3(\mathscr{S}) \cup \{occurs(a_i, i) \mid i = 0, \ldots, n-1\}$ then $prob(\Pi_{i=0}^{n-1} pr(a_i, s_i), n) \in S$. To compute the best strategy, we add the rule $\texttt{\#maximize}\{\texttt{V} : \texttt{prob}(\texttt{V}, \texttt{n})\}.$ to $\Pi_3(\mathscr{S})$.

Example 4. Continue with Example 3. We assume that the probability of success of $\texttt{tOn}(\texttt{detect\_up\_11})$, $\texttt{tOn}(\texttt{control\_speed})$, and $\texttt{patch}(\texttt{control\_speed})$ are 0.8, 0.7, 0.3 in every state, respectively. In this case, the strategies α_1 and α_3 have the maximal probability to succeed.

4 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 that is being built for use by CPS designers, managers and operators. We also include preliminary considerations on performance aspects.

The decision-support system relies on an implementation of the translator and of the different modules for answering queries described in Sec. 3, and comprises a reasoning component and a visualization component. Figure 4 shows the reasoning component at work on two queries related to the elevator example. Notice how the user can ask the system to reason about satisfaction of concerns and to produce mitigation plans. 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 into specific details. For instance, the sunburst visual from Figure 3 provides a view of the CPS from Figure 1 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.

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[10], the SPARQL reasoner[11] and the Clingo ASP solver. This 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*.

[10] 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.

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

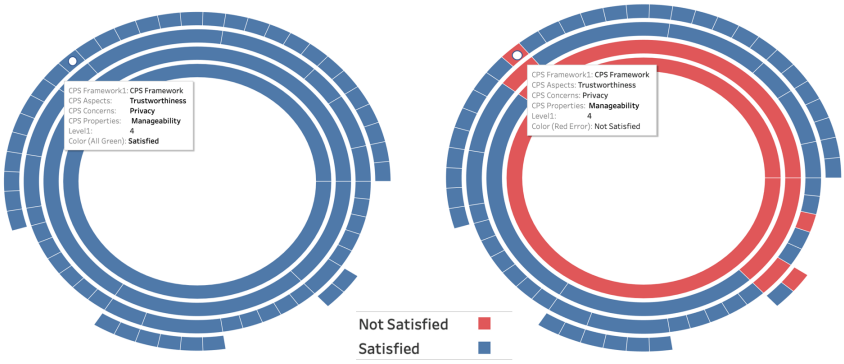

Fig. 3: Visualization component

In this preliminary phase of our investigation on scalability, all reasoning engines have exhibited similar performance, as exemplified by Table 3. The table summarizes the results of question-answering experiments on the Lane Keeping/Assist System (LKAS) [2] domain and on the Elevator domain. The reasoning tasks considered are for answering questions $\mathbf{Q}_1 - \mathbf{Q}_3$ discussed earlier. While the results show that the naïve engine is marginally better than the others, the differences are quite negligible, all within 10%.

Query	LKAS				Elevator			
	Naïve	Clingo -Python	DLVHex2	Hexlite	Naïve	Clingo -Python	DLVHex2	Hexlite
$\mathbf{Q}_1$	1.827s	2.013s	1.82s	1.831s	1.853s	2.03s	1.795s	1.88s
$\mathbf{Q}_2$	1.91s	2.15s	1.913s	1.924s	1.933s	2.076s	1.941s	2.02s
$\mathbf{Q}_3$	2.02s	2.33s	2.031s	2.027s	2.051s	2.253s	2.058s	2.181s

Table 3: CPS domains Querying, Extracting and Reasoning Summary

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., [3]). We obtained positive indications on this from preliminary experiments we conducted on ontologies featuring up to 150K triples, 85 classes, 61K individuals, 30 object properties, 40 data properties, and 45 subclass relations. In these experiments, running our system consistently took a minute or less. (We omit the details due to space considerations, since it is not the focus of this paper.) A thorough analysis will be the subject of a separate paper.

5 Conclusions, Related Work, and Discussion

This paper discusses three important problems related to the trustworthiness of CPS and their solutions using hybrid ontology-ASP reasoning. Specifically, for each problem, the paper presents a mechanism for answering it and proves relevant properties. To the best of our knowledge, this is the first attempt at a mathematically precise solution of issues in the CPSF, addressing the need for tools for evaluating the trustworthiness of CPS.

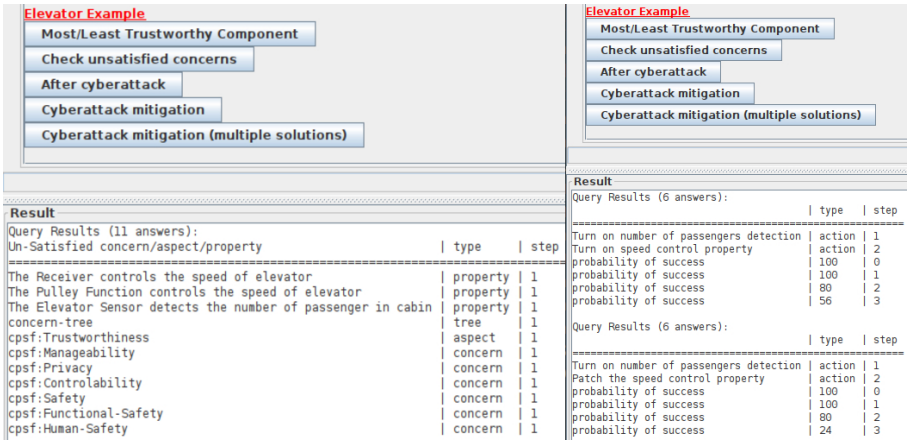

Fig. 4: Reasoning component

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) [18]. 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 [3,17,5]. 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 [1]. But these methods seem to struggle with supporting richer logics and "what-if" explorations. For relevant work in this direction, see [13,6].

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 that can deal with millions of atoms, incomplete information, default reasoning, and features that allow ASP to interact with constraint solvers and external systems. In our future works, we continue monitoring to reason about the factors that affect to the *trustworthiness* of a CPS such as the probability that a component crashes, the accessibility of a component and the internal probability of system errors.

The second author is partially supported by NSF grants 1757207, 1812628, and 1914635.

 Portions of this publication and research effort are made possible through the help and support of NIST via cooperative agreements 70NANB18H257 and 70NANB19H102.

References

1. Balas, E.: Disjunctive programming: Cutting planes from logical conditions. In: Nonlinear Programming 2, pp. 279–312. Elsevier (1975)
2. Balduccini, M., Griffor, E., Huth, M., Vishik, C., Burns, M., Wollman, D.A.: Ontology-based reasoning about the trustworthiness of cyber-physical systems. ArXiv **abs/1803.07438** (2018)
3. Balduccini, M., Lierler, Y.: Constraint Answer Set Solver EZCSP and Why Integration Schemas Matter. Journal of Theory and Practice of Logic Programming (TPLP) **17**(4), 462–515 (2017)
4. Baral, C.: Knowledge Representation, reasoning, and declarative problem solving with Answer sets. Cambridge University Press, Cambridge, MA (2003)
5. Baral, C., Gelfond, M., Rushton, N.: Probabilistic reasoning with answer sets. Theory and Practice of Logic Programming **9**(1), 57–144 (2009)
6. D'Iddio, A.C., Huth, M.: ManyOpt: An Extensible Tool for Mixed, Non-Linear Optimization Through SMT Solving. CoRR **abs/1702.01332** (2017), http://arxiv.org/abs/1702.01332
7. Eiter, T.: Answer set programming for the semantic web. In: ICLP. LNCS, vol. 4670, pp. 23–26. Springer (2007)
8. Gelfond, M., Lifschitz, V.: Logic programs with classical negation. In: Warren, D., Szeredi, P. (eds.) Logic Programming: Proceedings of the Seventh International Conference. pp. 579–597 (1990)
9. Gelfond, M., Son, T.C.: Prioritized default theory. In: Selected Papers from the Workshop on Logic Programming and Knowledge Representation 1997. pp. 164–223. Springer Verlag, LNAI 1471 (1998)
10. Gelfond, M., Lifschitz, V.: Action languages. Electron. Trans. Artif. Intell. **2**, 193–210 (1998)
11. Griffor, E., Greer, C., Wollman, D.A., Burns, M.J.: Framework for cyber-physical systems: volume 1, overview (2017)
12. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm: a 25-year Perspective. pp. 375–398 (1999)
13. Mistr, M., D'Iddio, A.C., Huth, M., Misener, R.: Satisfiability modulo theories for process systems engineering. eprints for the optimization community (19 June 2017)
14. Nguyen, T.H., Potelli, E., Son, T.C.: Phylotastic: An experiment in creating, manipulating, and evolving phylogenetic biology workflows using logic programming. TPLP **18**(3-4), 656672 (2018). https://doi.org/10.1017/S1471068418000236
15. Nguyen, T.H., Son, T.C., Pontelli, E.: Automatic web services composition for phylotastic. In: PADL. pp. 186–202 (2018), https://doi.org/10.1007/978-3-319-73305-0_13
16. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**(3,4), 241–273 (1999)
17. Ostrowski, M., Schaub, T.: ASP Modulo CSP: The Clingcon System. Journal of Theory and Practice of Logic Programming (TPLP) **12**(4–5), 485–503 (2012)
18. Roy, A., Kim, D.S., Trivedi, K.S.: Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. Security and Communication Networks **5**(8), 929–943 (2012). https://doi.org/10.1002/sec.299, https://doi.org/10.1002/sec.299