Cyber-Physical System

Paper #XXX

ABSTRACT

KEYWORDS

CPS; Action and Changes; ASP

1 INTRODUCTION

- Cyber-physical systems (CPS) combine cyber capabilities (computation and/or communication) with physical capabilities (motion or other physical processes).
- Cars, aircraft, and robots are prime examples, because they move physically in space in a way that is determined by discrete computerized control algorithms
- CPS framework provides the taxonomy and methodology for designing, building, and assuring cyber-physical systems that meet the expectations and concerns of system stakeholders.
- Ontology reasoning: Answer Set Programming

2 BACKGROUND

2.1 A CPS Framework Ontology

At the core of this approach is an ontology of the CPS Framework and of a CPS of interest. An ontology is a formal, logic-based representation that supports reasoning by means of logical inference. In this paper, we adopt a rather broad view of this term: by ontology, we mean a collection of statements in a logical language that represent a given domain in terms of *classes* (i.e., sets) of objects, *individuals* (i.e., specific objects), relationships between objects and/or classes, and logical statements over these relationships. In the context of the trustworthiness of CPS, for instance, an ontology might define the high-level concept of "Concern" with its refinement of "Aspect." All of these will be formalized as classes and, for Aspect, subclasses. Specific concerns will be represented as individuals: *Trustworthiness* as an individual of class Aspect, *Security* and *Cybersecurity* of class Concern. Additionally, a relation "has-subconcern" might be used to associate a concern with its sub-concerns. Thus, Aspect "has-subconcern" *Security*, which in turn "has-subconcern" *Cybersecurity*. By introducing a property "satisfied," one could also indicate which concerns are satisfied. Inference can then be applied to propagate "satisfied" and other relevant properties and relations throughout the ontology. For example, given a concern that is not "satisfied," one can leverage relation "has-subconcern" to identify the concerns that are not satisfied, either directly or indirectly, because of it. In practice, it is often convenient to distinguish between the factual part, Ω, of the ontology (later, simply called "ontology"), which encodes the factual information (e.g., *Trustworthiness* "has-subconcern" *Security*), and the *axioms*, Λ, expressing deeper, often causal, links between relations (e.g., a concern is not satisfied if any of its sub-concerns is not satisfied). Further, when discussing reasoning tasks, we will also indicate, separately, the set $\mathcal{Q}$ of axioms encoding a specific reasoning task or query.

2.2 Reasoning with Ontology and Answer Set Program

By leveraging a logic-based representation of a domain of interest, one can apply inference and draw new and useful conclusions in a principled, rigorous way. In essence, our approach is agnostic to any specific choice of logical language and inference mechanisms. Axioms expressed in the used logical language formalize the queries one is interested in answering, the type of reasoning that can be carried out, and any additional contextual information. Thus, given an ontology Ω, a set of axioms Λ, and an inference relation $\vDash$, we say that Δ is an answer to the (implicit) query iff

$$\Omega \cup \Lambda \vDash \Delta.$$

where $\cup$ denotes the union of two sets. For instance, in the language of propositional logic, given knowledge that some proposition p is true and that p implies some other proposition q, one can infer that q is also true, i.e.:

$$\{p, p \supset q\} \vDash \{q\}.$$

In the context of cybersecurity, p might be true when a cyberattack has occurred and $p \supset q$ might formalize an expert's knowledge that, whenever that cyberattack occurs, a certain system becomes inoperative (proposition q). The logical inference represented by symbol $\vDash$ allows to draw the conclusion that, as a result of the cyberattack, the system is now inoperative. For increased flexibility of representation, we use here a non-monotonic extension of propositional logic, called Answer Set Programming (ASP) [1, 2].

An ASP program is built on a signature consisting of a set of constants $\mathcal{C}$, a set of variables $\mathcal{X}$ and a set of predicate symbols $\mathcal{P}$. The definitions of terms and atomic formulae (atoms) are the traditional ones. An answer set program is a set of rules of the form: a_0: $-a_1, ..., a_m$, `not` $a_{m+1}, ...,$ `not` a_n, where $0 \leq$ m $\leq$ n, each a_i is an atom, and `not` represents *negation-as-failure (naf)*. A naf-literal has the form `not` a, where a is an atom. Given a rule of this form, the left and right hand sides are called the *head* and *body*, respectively. A rule may have either an empty head or empty body, but not both. Rules with an empty head are called *constraints*—the empty head is implicitly assumed to represent `False`; rules with an empty body are known as *facts*. The language allows the use of variables, and they are simply viewed as placeholders for any element of $\mathcal{C}$; thus a rule with variables (non-ground) is simply a syntactic sugar for the set of rules obtained by consistently replacing each variable with any element of $\mathcal{C}$ (ground rules).

A set of ground atoms X satisfies the body of a ground rule if $\{a_{m+1}, ..., a_n\} \cap \mathrm{X} = \emptyset$ and $\{a_1, ..., a_m\} \subseteq X$. A *constraint* is *satisfied* by X if its body is not satisfied by X. X is a model of a rule if either it does not satisfy the body or $a_0 \in X$. X is a model of a program if it satisfies all of its rules.

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 2020, Auckland, New Zealand

https://doi.org/doi

If a ground program Π does not contain any naf-literals, then the semantics of Π is given by its unique subset-minimal model. Given a ground program Π and a set of ground atoms X, the *reduct* of Π w.r.t `X` (denoted by Π^X) is the program obtained from Π by: (*i*) deleting all the rules that have a naf-literal `not` a in the body and $a \in X$, and (*ii*) removing all remaining naf-literals.

A set of ground atoms X is an *answer set* of a program Π if X is the subset-minimal model of Π^X. Several syntactic extensions have been introduced to facilitate the development of program. For example, *choice atoms* have the form $l\,\{b_1,...,b_k\}\,u$, where each b_j is an atom, and l and u are integers such that $l \leq u$. A set of atoms X satisfies a choice atom $l\{b_1,...,b_k\}u$ if $l \leq |X \cap \{b_1,...,b_k\}| \leq u$. Non-ground versions of choice atoms allow the use of syntax analogous to that of intensional sets, e.g., $l\,\{p(X)\colon q(X)\}\,u$.

2.3 Representing Ontology in ASP

The entities of the CPS Ontology (classes, instances and properties) are represented in ASP using unary predicate `class` and the binary predicate `subClass`. The predicate `class(X)`, `subClass(X,Y)` say that `X` is a class, `X` is a subclass of `Y`, respectively. To reason about subclass relationship, the encoding contains the rule `subClass(X,Y) :- subClass(Z,Y), subClass(X,Z).`

The predicate `aspect(I)`, `concern(I)`, `property(I)` denotes that `I` is an individual of class aspect, concern and property, respectively. For example, in CPS ontology, *concern*, *aspect* and *property* are classes and *aspect* is subclass of *concern*. There are nine aspects (highest level concerns) (e.g., *trustWorthiness*, *business*, *timing*, etc.) and they are individuals of class *aspect*. In addition, there are many instances of class *concern* such as: *security*, *reliability*, *functionality*, *cyber-security*, *physical-security* etc. In which, *cyber-security* and *physical-security* are sub-concerns of *security*. Concern *security*, *reliability* are sub-concerns of *trustWorthiness*. The predicate `subconcern(I,J)` denotes that concern `J` is sub-concern of concern `I`. All above entities and their relations are described in ASP by the set of facts:

Listing 1: ASP representation aspects and concerns in Ontology Ω

```
class(concern). class(aspect). class(property).
subClass(aspect,concern).
aspect(trustWorthiness). aspect(business). aspect(timing).
concern(security). concern(reliability). concern(functionality).
concern(cyber-security). concern(physical-security).
subconcern(trustWorthiness,security). subconcern(trustWorthiness,reliability).
subconcern(security,cyber-security). subconcern(security,physical-security).
```

In the current system, the CPS Ontology is queried by `SparQL` language to retrieve all RDF triplestores. Each RDF triplestore is a tuple `(S,P,O)` (`S`: Subject, `P`: Predicate or Verb, `O`:Object) that is encoded in ASP by atom of the form `input(S,P,O)`.

Listing 2: ASP reasoning RDF triplestores in Ontology Ω

```
property(P):-input(P,"rdf:type","cpsf:Property").
concern(C) :- input(C,"rdf:type","cpsf:Concern").
aspect(A) :- input(A,"rdf:type","cpsf:Aspect").
subconcern(S,O):- input(S,"cpsf:includesConcern",O).
% Aspect is a concern and is subconcern of "all"
concern(A) :- aspect(A).
concern(all).
subconcern(all,A) :- aspect(A).
relevantToFunc(C,F) :- input(C,"cpsf:relevantToFunc",F).
relevantToFunc(C1,F) :- relevantToFunc(C2,F), subconcern(C1,C2).
```

Properties of CPS are represented by individual of class `Property`. The relation between concern C and property P is represented by the predicate `addressedBy(C,P)` which denotes that concern C is address by property P.

Listing 3: The relation between Properties and Concerns in Ω

```
addressedBy(C,P):- concern(C), property(P), input(C,"cpsf:addressedByProperty",P), ).
addressedBy(C,P):-addressedBy(C1,P), subconcern(C,C1).
```

In ASP representation, each *configuration* (e.g, in listing 4) includes a set of facts in the ASP form `obs(conf,true)` which denote that in the initial state of CPS system, the fluent `conf` holds at step 0 (Line 1-7). Line 8 denotes that a property X was impacted negatively then it does not hold. In CPS ontology, the terminology *Configuration* presents the current state of CPS system and each individual of class *Configuration* represents a different configuration feature.

Listing 4: ASP reasoning Configurations (Initial State) in Ω

```
1 obs(sat("encrypted_mem_input1"),true).
2 obs(sat("encrypted_mem_sam"),true).
3 obs(sat("sec_boot_input1"),true).
4 obs(sat("sec_boot_sam"),true).
5 obs(sat("slow_mode_input1"),true).
6 holds(F,0) :- fluent(F), obs(F,true).
7 -holds(F,0) :- fluent(F), obs(F,false).
8 -holds(X,T) :- impacted(neg,X,T).
```

Listing 5: An example of negative impaction of a property

```
1 impacted(neg,sat("consist_reading_freq_input1"),S) :- step(S), holds(sat("using_basic_input1"),S),holds(sat("encrypted_mem_input1"),S), -holds(sat("slow_mode_input1"),S).
```

Constraints, dependencies, trade-offs. An additional feature of our model is the ability to establish causal links between concerns, properties, configurations, and actions. This is accomplished by the reasoning over statements. Table 1 lists types of statements, their syntactic expressions as judgments, and their corresponding encodings for the ASP reasoner. The logical encodings of the statements are used to implement reasoning capabilities discussed later in the paper. Γ and π range over (sets of) propositions and a over actions. For example of a property dependency statement, Rule 1 in listing 5 encodes that when the basic type of device `input1` is in use

Statement type	Syntax	Encoding for reasoner
Property dependency	Γ impacts$_{\text{pos}}$ π Γ impacts$_{\text{neg}}$ π	$impacted(pos/neg, \pi, S) \leftarrow holds(\Gamma, S)$
Default property value	σ defaults $true$ σ defaults $false$	defaults(σ, $truefalse$)
Effects of actions	a causes π if Γ	$holds(\pi, S+1) \leftarrow holds(\Gamma, S), occurs(a, S)$
Triggered actions	Γ triggers a	$occurs(a, S) \leftarrow holds(\Gamma, S)$

Table 1: Constraints, dependencies, and trade-offs where Γ, π range over (sets of) propositions and a over actions

and input1 is using encrypted memory and fast mode impacts negatively the maintainability of a consistent reading frequency.

2.4 A Use Case

For sake of illustration, in a lane keeping/assist (LKAS) use case centered around 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. The camera and the SAM may use encrypted memory and secure boot. Safety mechanisms in the navigation system cause it to shut down if issues are detected in the input received from the SAM. This use case is chosen because it encompasses major component types of a CPS, and lends itself to various non-trivial investigations. Through this use case, we will highlight the interplay among trustworthiness concerns, as well as their ramifications on other CPS aspects, such as the functional aspect.

Assuming that the camera 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. It is assumed that two camera models exist, a basic one and an advanced one. Either type of camera can be used when realizing the CPS. Due to assumed technical limitations, the basic camera is likely to drop frames if it attempts to record at 50 fps while using encrypted memory.

The decomposition of a CPS identifies resources that may satisfy properties. Suppose that CAM is a camera, a subsystem of an autonomous car, and that SAM is a memory sub-system of CAM; we will examine this system in more detail later. Then cam_mem[encr], e.g., is a Boolean predicate that is true if the memory mem of camera CAM uses encryption. Properties thus have form SystemPath[prop] where SystemPath identifies a system component or part, with sub-components indicated by the underscore symbol, and prop a property that this part may enjoy. Properties SystemPath[prop] also have a semantic context ConcernPath that articulates which (sub)concern of an aspect this property is trying to address. In our semantics below, a property may be either true or false (i.e., satisfied or non-satisfied). These truth values in turn influence the satisfaction of concerns and aspects.

3 FORMALIZATION

3.1 Formalization CPS ontology

The formalization of a CPS is organized along multiple levels: (L1) aspects and concerns; (L2) properties; (L3) CPS configuration; (L4) actions; (L5) constraints, dependencies and trade-offs; and (L6) satisfaction axioms. Level L1 and L6 form the *CPS-independent specification*, since aspects and concerns are independent of the specific CPS being modeled. Levels L2-L5 comprise the *CPS-dependent specification*, as the information included in them depends on the CPS being modeled. Furthermore, levels L1 and L2 formalize the concepts from the definition of the CPS Framework. Levels L3-L5 extend the CPS Framework in order to provide details needed for reasoning about the behavior of a CPS of interest. Level L6 provides the semantics of the formalization. The ASP representation for 6 levels of CPS ontology is described in section 2.3

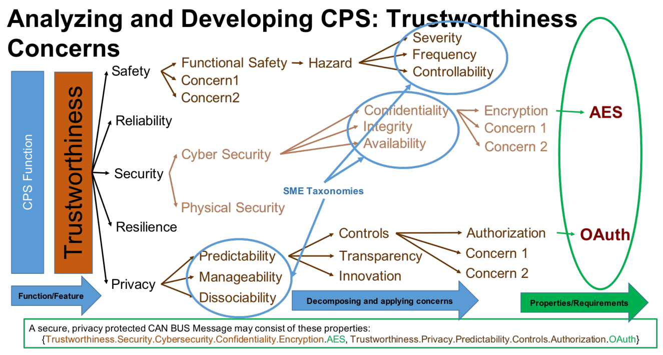
3.2 A Physical CPS System

Definition 3.1. A physical CPS system S is a tuple (C, A, F, R) where:

- C is a set of physical components.
- A is a finite set of actions that can be execute over CPS system.
- F is a finite set of fluent literals.
- R is a set of relations that map each physical component $c \in C$ with a set of physical component properties that are defined in CPS Ontology.
  For any $r \in R$, $r : C \longrightarrow 2^{P_C}$. In which, we have 2^{P_C} is power set of P_C, P_C is set of properties which related to physical components, P is set of all properties that are defined in CPS ontology 2.1 and $P_C \subset P$.

Intuitively, the CPS problem related the example in use case 2.4 can be specified by $S_{auto} = (C_{auto}, A_{auto}, F_{auto}, R_{auto})$ where:

- $C_{auto} = \{$SAM, CAM$\}$.
- A_{auto} is set of finite actions which consists of:
  - switMem(X,encrypted), switMem(X,unencrypted): denote actions to switch component X using encrypted or unencrypted memory respectively.
  - switReMod(X,50fps), switReMod(X,25fps): denote actions to switch component X using 50 fps and 25 fps recording modes respectively.
  - switModel(X,basic), switReMod(X,advanced): denote actions to switch component X using basic and advanced models respectively.
  - etc. (should be more)
- F_{auto} is a finite set of fluent literals which consists of:
  - useMem(X,encrypted), useMem(X,unencrypted): denote that component X is using encrypted or unencrypted memory respectively.
  - recordMode(X,50fps), recordMode(X,25fps): denote that component X is recording video with 50 fps and 25 fps modes respectively.
  - useModel(X,basic), useModel(X,advanced): denote that component X is working on basic and advanced models respectively.
  - etc. (should be more)



A secure, privacy protected CAN BUS Message may consist of these properties:
{Trustworthiness.Security.Cybersecurity.Confidentiality.Encryption.AES, Trustworthiness.Privacy.Predictability.Controls.Authorization.OAuth}

Figure 1: Decomposition tree for Trustworthiness Concerns

- R_{auto} = {`CAM` ↦ {`reMode_50fps`, `model_basic`, ...}, `SAM` ↦ {`mem_encrypted`, `boot_sec`, ...} }
  In which, `reMode_50fps`, `reMode_25fps`, `model_basic`, `mem_encrypted`, `boot_sec`, `etc`. are defined in CPS Ontology properties 2.1.

3.3 Truthworthiness Problem Definition

Truthworthiness is one of nine highest level concerns (*aspects*) and a critical concern stakeholders have about CPS and the Internet of Things (IoT) and their development.There, *trustworthiness* is captured as a high-level concern encompassing safety, security, privacy, resilience, and reliability. While there are many efforts, in multiple sectors, to study these characteristics of systems they are typically considered separately and in isolation. This can result in work, intended to address one of these concerns, adversely impacting work to address one or more of the others. Thus CPS/IoT trustworthiness relies on an integrated, concern-driven approach that takes into account the interactions between the cyber and physical elements of systems. There are some important *Truthworthiness* queries which should be answered in CPS system:

- Is the Truthworthiness aspect satisfied?
- What is the most vulnerability in CPS system?
- If there exists the vulnerability in CPS system, how to fix it?

3.4 Probability of success of mitigation strategies

Definition 3.2.

3.5 Likelihood of Concerns Satisfaction

Definition 3.3.

3.6 Vulnerability in CPS System

Definition 3.4.

PROPOSITION 3.5.

4 IMPLEMENTATION

4.1 Query 1: Is the Truthworthiness aspect satisfied?

Listing 6: Reasoning for the satsifaction of aspect

```
1  -holds(sat(C),T):-addressedBy(C,π),not holds(π,T)
       .
2  -holds(sat(C),T) :-addressedBy(C,π),-holds(π,T).
3  -holds(sat(C1),T) :-subconcern(C1,C2),not holds(
       sat(C2),T).
4  -holds(sat(C1),T) :-subconcern(C1,C2),-holds(sat(
       C2),T).
5  holds(X,T) :- defaults(X,true), not -holds(X,T).
```

In this listing, T denotes a discrete step in the evolution of the CPS. The inclusion of a step argument makes it possible to analyze the evolution of CPS over time in response to possible events. Lines 1-2 states that a concern C is not satisfied if any of the properties that address it does not hold. This ensures that the lack of satisfaction of a property π is propagated to the concern(s) that are addressed by π according to the `addressedBy/2` statements. Line 3-4 encodes that the lack of the satisfaction is then propagated up the concern tree according to the concern-concern relation specified by the `subconcern/2` statements. The specification of the notion of satisfaction is completed by `defaults` statements saying that all properties and concerns are satisfied by default (Line 5), which embodies the semantics of the `defaults` statement in Table 1.

The probabilistic component was added to the model, supporting calculation of (**1**) *probability of success of mitigation strategies* and (**2**) *likelihood that concerns are satisfied* .

In probability of success of mitigation strategies, the fluent `prob_of_state(prob)` models the propagation by the model to the successor state and the statement `holds(prob_of_state(prob),S)` means that at step S of the evolution the system, the probability in the current state described by the fluent is `prob`. The statement `do(a,S)` denotes that an action a is executed at step S of CPS evolution. The fluent `prob_success(a,`$prob_a$`)` denotes that an action a has probability $prob_a$ of success.

Listing 7: Probability of success of mitigation strategies

```
1  holds(prob_of_state(100),0).
2  holds(prob_of_state(P2),S2) :- step(S),step(S2),
       S2=S+1, holds(prob_of_state(P),S),do(A,S),
       prob_success(A,PSucc),P2a=P*PSucc,P2=P2a/100.
3  known_prob_success(S) :- step(S),do(A,S),
       prob_success(A,PSucc).
4  unknown_prob_success(S) :- step(S),not
       known_prob_success(S).
5  holds(prob_of_state(P),S2) :- step(S), step(S2),
       S2=S+1, holds(prob_of_state(P),S),
       unknown_prob_success(S).
```

Listing 7 provides set of rules to generate the probability of success of actions. Line 1 denotes that the probability of CPS system state at step 0 is 100. Line 2 provides the probability calculation of CPS state at step S_2 based on the probability of system at previous step S and the probability of success of action A that is executed in step S. This calculation has been formalized in definition 3.2. Line 3-4 provides the availability information of the probability of success for action a executed at step S. Based on them, line 5 encodes the calculation for the probability of CPS state at step S_2 in the case that there is no action a or the probability of success of action a is unknown in previous step S. The ASP encoding in listing 7 is able to calculate the probabilities of success of mitigation strategies based on multiple solution actions which attached with probabilities of success. Based on this result, we can select the best mitigation strategy with highest probability of success.

Listing 8: Likelihood of Concern Satisifaction

Listing 9: Output of Query 1

```
last_step(S) :- step(S),S2=S+1,not step(S2).
output(C,concern,F,S) :- last_step(S), -holds(sat
     (C,F),S),concern(C),not aspect(C), C!=all.
output(C,aspect,F,S) :- last_step(S), -holds(sat(
     C,F),S),aspect(C).
output(D,property,"-",S) :- last_step(S), -holds(
     sat(A),S), atomic_statement(P,A), descr(P,D).
output("concern-tree",tree,F,S) :- last_step(S),
     -holds(sat(all,F),S).
output(D,action,"-",S) :- step(S), action(A), do(
     A,S), action_descr(A,D).
output("probability of success",P,"-",S) :- step(
     S), holds(prob_of_state(P),S).
```

The listing 9 provides ASP encoding to generate the answers (output) for CPS questions such as *Is the*

Listing 10: Optimization to select the best mitigation strategy

```
#maximize{P : holds(prob_of_state(P),S)}.
```

4.2 Query 2: What is the most vulnerability in the system?

Listing 11: Example for Ontology and Physical CPS System

```
% --- CPS Ontology Example ---
% Concerns
concern(a1). concern(a2). concern(a3). concern(b0
     ). concern(b1). concern(b2). concern(b3).
% Concerns-Concern relation
subconcern(b1,a1).
subconcern(b0,b1).
% Property
property(x).
property(y).
property(z).
property(t).
% Property-Concern relation
addressedBy(a1,x).
addressedBy(a2,x).
addressedBy(a3,x).
addressedBy(a3,y).
addressedBy(b1,y).
addressedBy(b1,z).
addressedBy(b2,z).
addressedBy(b3,z).
addressedBy(a2,z).
addressedBy(b1,t).
addressedBy(b2,t).
addressedBy(b3,t).
addressedBy(a2,t).
```

```
% --- Physical CPS System Example ---
% Component
component(c1).
component(c2).
component(c3).
% Relation Component-Property \in R
relation(c1,x).
relation(c1,y).
relation(c2,y).
relation(c2,x).
relation(c2,z).
relation(c3,z).
relation(c3,t).
```

Listing 12: Reasoning for Truthworthiness aspect

```
% --- Configuration ---
sol(addr).
%sol(all).
% --- State of CPS System ---
step(0).
% --- Reasoning ---
% -Step 1: Generate from Ontology and physical
     CPS system to define component and component
     -> a set of property
property(P):-input(P,"rdf:type","cpsf:Property").
component(C):-input(C,"rdf:type","cpsf:Component
     ").
holds(P,0) :- obs(P,true), property(P).
compHoldProp(C,P,S):- relation(C,P), component(C)
     , property(P), holds(P,S), step(S).
% -Step 2 - S1 sol(addr): Compute for number of
     relations based on the number of addresses
     between property and concern ONLY
count_relations(P,N) :- N = #count{C :
     addressedBy(C,P)}, property(P), sol(addr).
% Step 3: Compute Truthworthiness value of a
     component in CPS system based on total number
      of relations of its property.
truthworthiness_comp(C,TW,S) :- component(C),
     step(S), TW = #sum{N,P : count_relations(P,N)
     , property(P), compHoldProp(C,P,S)}.
% Step 4: Generate the component in S which has
     the highest TWvalue
higher_TW(C1,C2,S) :- truthworthiness_comp(
     C1,TW1,S), truthworthiness_comp(C2,TW2,S),
     step(S), TW1 > TW2.
not_highest_TW(C2,S) :- component(C1),component(
     C2) , higher_TW(C1,C2,S), step(S).
highest_TW(C,S) :- not not_highest_TW(C,S), step(
     S).

%===MORE===
% Step 2 - S2 sol(all): Compute for number of
    relations based on the number of addresses +
    relations between property and concern AND
    addressed concern with parent concerns and go
     up higher in concern tree.
addrForConcern(P,C) :- property(P), addressedBy(
    C,P), sol(all).
addrForConcern(P,C) :- addrForConcern(P,C1),
    subconcern(C,C1), sol(all).
count_relations(P,N) :- N= #count{C :
    addrForConcern(P,C)}, property(P), sol(all).
```

4.3 Query 3: If there exists the vulnerability, how to fix it?

Listing 13: Reasoning for Truthworthiness aspect

5 EXPERIMENT EVALUATION

6 CONCLUSIONS

REFERENCES

[1] Tran Cao Son, Chitta Baral, Nam Tran, and Sheila McIlraith. 2006. Domain-Dependent Knowledge in Answer Set Planning. *ACM Transactions on Computational Logic* 7, 4 (2006).

[2] Tran Cao Son and Enrico Pontelli. 2006. Planning with Preferences using Logic Programming. *Theory and Practice of Logic Programming* 6 (2006), 559–607. Issue 05.