Reasoning about Truthfulness of Agents Using Answer Set Programming

Tran Cao Son and **Enrico Pontelli**
Computer Science Department
New Mexico State University

Michael Gelfond
Computer Science Department
Texas Tech University

Marcello Balduccini
Computer Science Department
Drexel University

Abstract

This paper describes a declarative framework for representing and reasoning about truthfulness of agents using Answer Set Programming. The paper illustrates how, starting from observations, knowledge about the actions of the agents, and the normal behavior of agents, one can evaluate the statements made by agents against a set of observations over time. The paper presents an ASP program for computing the truthfulness of statements of agents over time. It also introduces the notions of a supporter (denial) of a statement and discusses possible ways for computing minimal supporter (denial). The paper presents an application of the proposed framework, specifically in detecting man-in-the-middle attacks targeting computer and cyber-physical systems. Finally, the paper briefly relates the proposed framework to works on trust and reputation of agents and discusses possible extensions.

Introduction

The recent developments in the field of AI enable the development of agents that can replace humans in many tasks. Increasingly, organizations are using web-bots for interacting with clients in various capacities, e.g., in providing information about the company or making offers. It is reasonable to believe that this trend will continue and grow, as long as the Internet exists. Unfortunately, not every business on the Internet is as honest as one would hope. Stories about businesses that cheat people of goods or services or wrongly advertise their services are not uncommon. This leads to the development of businesses that allow people to rate companies (e.g., `expedia.com`, `yelp.com`, or Angie's List) or defend the reputation of a company or an entity (e.g., `reputation.com`).

Example 1 *The GIZMODO website carries the following story*[1]*: A man named Tin Le from Australia managed to fool the world's media by posting his screenshot of a passport with the name "Phuc Dat Bich" and claiming that he was denied access to a social network because of his name. It turned out to be a hoax. Yet, before he acknowledged that it was a hoax, BBC and Sky News reported the story as true. Observe that the story was reported as a true story even if requests for interview had been denied.*

The above example displays a typical situation in which some pieces of information were disseminated as true and was found later to be false. It would have been better if the news outlets had verified the truthfulness of the statement before disseminating it. In fact, if they had just applied some common-sense by considering that "*normally, a statement cannot be considered as true if it is not verified*", then they would not have reported the hoax. The example also calls for the development of systems capable of determining whether a given statement is truthful or not.

In this paper, we are interested in reasoning about the truthfulness of agents. We will judge agents by what they do or what we observe rather than by what they say. Our observations are made at different time instances along a linear time line. We assume that whatever observed is true at the time it is observed and will stay true until additional information indicates otherwise. Furthermore, we will need to judge agents even when we do not have complete information about them. This means that reasoning about the truthfulness of agents is a non-monotonic task that is often done under incomplete information. The next example illustrates these issues.

Example 2 *When we first met at an event, John said that he comes from a poor family (t_0). We later learn that John attends the college in town that is famous for its high tuition (t_1). Much later, we learn that John attends the college because he obtained a full ride scholarship from the school due to his financial hardship (t_2).*

This story spans over three time instances: t_0, t_1, and t_2. At each time instance, we learn (observe) some facts or receive some statements and they affect our belief in John's statement as follows.

- *t_0: John says that his family is poor ($poor$). It is likely that we would believe John when he says that his family is poor. This is because we have nothing to conclude otherwise.*
- *t_1: We observe the fact that John attends an expensive college ($in\_college$).* Since students attending the college are normally from rich families *(default d_1), this might be a reason for us to conclude that John has lied to us.*

[1] `http://gizmodo.com/phuc-dat-bich-is-a-massive-phucking-faker-1744588099?utm_campaign=socialflow_gizmodo_facebook&utm_source=gizmodo_facebook&utm_medium=socialflow`

We indicate that the default d_1 is the reason to draw the conclusion.

- t_2*: We observe the fact that John has a full ride scholarship due to his financial hardship* ($has\_scholarship$). Since a student's hardship is usually derived from the family's financial situation *(default d_2), this fact would at least allow us to withdraw the conclusion made at the time instance t_1 that John is a liar. It is still insufficient for us to conclude that John's family is poor.*
  The situation might be different if, for example, we have a preference among defaults. In this example, if we are inclined to believe in the conclusion of d_2 more than that of d_1, then we would believe that John's family is poor and thus restore our trust in John's statement.

The main contribution of this paper is the formalization of a model to represent and reason about truthfulness of agents using Answer Set Programming. The paper illustrates how, starting from

- a collection of observations about the (possibly evolving) state of the world,
- observations about (some of) the actions performed by the agent, and
- the observer's own bias about the *normal* behavior of agents

an observer can draw conclusions about the truthfulness of statements made by the observed agent. We illustrate the framework using examples and discuss possible extensions that need to be considered. We also discuss a potential application of the proposed framework.

Background: Answer Set Programming (ASP)

Answer Set Programming (ASP) (Baral 2003) is the language of logic programs under the answer set semantics (Gelfond and Lifschitz 1990). A logic program Π is a set of rules of the form

$$c_1 \mid \ldots \mid c_k \leftarrow a_1, \ldots, a_m, \; not\; a_{m+1}, \ldots, \; not\; a_n \quad (1)$$

where $0 \leq m \leq n$, $0 \leq k$, each a_i or c_j is a literal of a propositional language[2] and not represents *negation-as-failure*. A negation-as-failure literal (or naf-literal) is of the form $not\; a$ where a is a literal. For a rule of the form (1), the left and right hand sides of the rule are called the *head* and the *body*, respectively. Both the head and the body can be empty. When the head is empty, the rule is called a *constraint*. When the body is empty, the rule is called a *fact*.

For a rule r of form (1), $H(r)$ and $B(r)$ denote the left and right hand side of $\leftarrow$, respectively; $head(r)$ denotes $\{c_1, \ldots, c_k\}$; and $pos(r)$ and $neg(r)$ denote $\{a_1, \ldots, a_m\}$ and $\{a_{m+1}, \ldots, a_n\}$. For a program Π, $lit(\Pi)$ denotes the set of literals occurring in Π.

Consider a set of ground literals X. X is *consistent* if there exists no atom a such that both a and $\neg a$ belong to X. The body of a rule r of the form (1) is *satisfied* by X if $neg(r) \cap X = \emptyset$ and $pos(r) \subseteq X$. A rule of form (1) with nonempty head is satisfied by X if either its body is not satisfied by X or $head(r) \cap X \neq \emptyset$. A constraint is *satisfied* by X if its body is not satisfied by X.

For a consistent set of ground literals S and a program Π, the *reduct* of Π w.r.t. S, denoted by Π^S, is the program obtained from Π by deleting **(i)** each rule that has a naf-literal $not\; a$ in its body with $a \in S$, and **ii** all naf-literals in the bodies of the remaining rules.

S is an *answer set (or a stable model)* of Π (Gelfond and Lifschitz 1990) if it satisfies the following conditions: **(i)** If Π does not contain any naf-literal (i.e., $m = n$ in every rule of Π) then S is a minimal consistent set of literals that satisfies all the rules in Π; and **(ii)** If Π does contain some naf-literal ($m < n$ in some rules of Π), then S is an answer set of Π if S is the answer set of Π^S. Note that Π^S does not contain naf-literals; hence its answer set is defined in the first item. A program Π is said to be *consistent* if it has an answer set. Otherwise, it is inconsistent.

To increase the expressiveness of ASP and simplify its use in applications, the language has been extended with several constructs such as

- Weight constraint atoms (e.g., (Niemelä, Simons, and Soininen 1999)) of the form:

$$l\; [a_1 = w_1, \ldots, a_n = w_n, \\ not\; b_{n+1} = w_{n+1}, \ldots, b_{n+k} = w_{n+k}]\; u \quad (2)$$

  where a_i and b_j are literals and l, u, and w_j's are integers, $l \leq u$.

- Aggregates atoms (e.g, (Faber, Leone, and Pfeifer 2004; Pelov, Denecker, and Bruynooghe 2004; Son and Pontelli 2007)) of the form:

$$f(S)\; op\; v \quad (3)$$

  where S is a set-literal, $f \in \{\text{SUM}, \text{COUNT}, \text{MAX}, \text{MIN}\}$, $op \in \{>, <, \geq, \leq, =\}$, and v is a number; a set-literal is of the form (*i*) $\{\vec{X} \mid p(\vec{W})\}$ where $\vec{X}$ is a vector of variables, $\vec{W}$ is vector of parameters and constants such that each variable in $\vec{X}$ also occurs in $\vec{W}$; or (*ii*) $\{\!\{\vec{X} \mid \vec{Y}.p(\vec{W})\}\!\}$ where $\vec{X}$ and $\vec{Y}$ are vectors of variables, $\vec{W}$ is vector of parameters and constants such that each variable in $\vec{X}$ or $\vec{Y}$ also occurs in $\vec{W}$.

The semantics of logic programs with such atoms has been defined (e.g., (Faber, Leone, and Pfeifer 2004; Niemelä, Simons, and Soininen 1999; Pelov, Denecker, and Bruynooghe 2004; Son and Pontelli 2007). Standard syntax for these types of atoms has been proposed and adopted in most state-of-the-art ASP-solvers such as CLASP (Gebser et al. 2007) and DLV (Citrigno et al. Sep 1997).

A Framework for Reasoning about Truthfulness of Agents

In this section, we develop a framework for representing and reasoning about the truthfulness of (statements made by) agents. We assume that

- We can observe the occurrences of the agents' actions and the properties of the world over time (e.g., we observe that John buys a car, John is a student, etc.);

[2]Rules with variables are viewed as a shorthand for the set of their ground instances.

- We have adequate knowledge about the agents' actions and their effects (e.g., the action of buying a car requires that the agent has money and its execution will result in the agent owning a car);
- We have a repertoire of common-sense knowledge about normal behaviors (e.g., a person attending an expensive school normally comes from a rich family, a person obtaining scholarship due to his parent financial status usually comes from a poor family); and
- We consider a statement asserting that a proposition p is true (resp. false) to be truthful with respect to a logical theory T, if $T \models p$ (resp. $T \models \neg p$) where $\models$ is the entailment relation defined for T.

We will develop our framework in two steps. We start with a framework that does not take into consideration action occurrences. Later, we will add observations about action occurrences.

No Observations about Action Occurrences

We assume that we are working with a propositional language $\mathcal{L}$, called a *signature* whose elements are called fluents. Fluent literals (or literals) and formulae are defined as usual. For a fluent literal l, $\bar{l}$ denotes its negation. We allow agents to make statement about literals.

Definition 1 *A* statement *about a fluent literal P is of the form*

$$statement(P, S) \tag{4}$$

where S is a non-negative integer.

For example, John said that he is sick. This can be stated as

$$statement(sick, 0)$$

says that $sick$ is true at time step 0.

Another example is that John said that he is poor. This can be stated as

$$statement(poor, 0)$$

states that $poor$ is true at time step 0.

We can make observations about fluent literals.

Definition 2 *An observation about a fluent literal P is of the form*

$$observed(P, S) \tag{5}$$

where S is a non-negative integer.

Note that $observed$ is understood in a broad sense: it might be that we learn some facts about the fluent literal, or we actually observed it (via the execution of a sensing action), etc.

For example, if we observe that John is not sick, we use

$$observed(\neg sick, 0).$$

Our reasoning module consists of a collection of rules and defaults, possibly with priorities among defaults.

- A *rule* is of the form

$$rule(r, head, body) \tag{6}$$

where r is the name of the rule, $head$ is a fluent literal, and $body$ is a collection of literals. A rule (6) says that whenever $body$ holds then $head$ must also hold.

- A *default* is of the form

$$default(d, head, body) \tag{7}$$

where d is the name of the default, $head$ (or *conclusion*) is a fluent literal, and $body$ is a collection of literals. A default (7) says that, normally, if $body$ holds then $head$ also holds.

- A *preference* between two defaults is expressed by

$$prefer(d_1, d_2, body) \tag{8}$$

where $body$ is a set of fluent literals and d_1 and d_2 are names of two defaults. This says that the default d_1 is preferred to d_2 whenever $body$ holds.

Definition 3 *A* knowledge base *(about an agent) over a propositional language $\mathcal{L}$ is a pair $KB = \langle O, RD \rangle$ where*

- *O is a set of observations.*
- *RD is a collection of rules, defaults, and preferences between defaults.*

We will next define the notion of consistent knowledge bases. We need some extra notations:

- A set of observations O is *consistent* if, for each time step s and proposition p, we have that $\{observed(p, s), observed(\neg p, s)\} \not\subseteq O$; and
- A set of rules and defaults RD is *consistent* if for any set of literals X, the logic program

$$RD_\pi = \{head \leftarrow body \mid rule(r, head, body) \in RD\} \cup X$$

is consistent.

Definition 4 *A $KB = \langle O, RD \rangle$ is* consistent *if O and RD are consistent.*

Our goal is to identify whether or not a particular statement $statement(p, s)$ is true at a time step t for $t \geq s$ given a KB. We will develop a program $\Pi(KB)$ to answer this question. We assume a finite horizon of time steps, denoted by $step(0), \ldots, steps(k)$. We use $h(P, S)$ to encode that literal P is true at step S. Furthermore, to simplify the use of $\Pi(KB)$ with current answer set solvers, we will encode a rule of the form (6) by the atom $rule(r, head, body\_name)$ and the set of atoms $\{member(l, body\_name) \mid l \in body\}$; likewise, a default of the form (7) by the atom $default(r, head, body\_name)$ and the set of atoms $\{member(l, body\_name) \mid l \in body\}$; and a preference of the form (8) by the atom $prefer(d_1, d_2, body\_name)$ and the set of atoms $\{member(l, body\_name) \mid l \in body\}$. The main difficulty lies in the interaction between observations, defaults, etc. In doing so, we apply the following principles:

(O)ptimistic: The most recent observation reflects the true state of the world.

(SK)eptical: When conflicting conclusions can be drawn then believe in none!

$\Pi(KB)$ contains the set of facts encoding O and RD as described above. We next describe the rules in $\Pi(KB)$.

- For reasoning about observations of the form (5), we have the following rules:

$$h(P,S) \leftarrow \begin{array}{l} observed(P,S_1), step(S), S_1 \leq S, \\ not\ \ not\_most\_recent(P,S_1,S). \end{array} \quad (9)$$

$$\begin{array}{l} not\_most\_recent(P,S_1,S) \leftarrow step(S_1), \\ \qquad observed(\overline{P},S_2), step(S), \\ \qquad S_1 < S_2, S_2 < S. \end{array} \quad (10)$$

  This rules encode the principle **(O)**. Rule (10) states that $not\_most\_recent(P,S_1)$ is true if there exists a more recent observation of $\neg P$. Rule (9) indicates that an observation stays true if every conflicting observation is 'older'.

- For reasoning about rules of the form (6), we have:

$$\begin{array}{l} h(H,S) \leftarrow step(S), rule(D,H,M), \\ \qquad NC = \#count\{L : member(M,L)\}, \\ \qquad \#count\{L : member(M,L), h(L,S)\} == NC. \end{array} \quad (11)$$

  This rule is straightforward. It says that if the body of a rule is satisfied then the head of the rule must be true.

- For reasoning about defaults of the form (7) we have the following rules:

$$\begin{array}{rl} applicable(D,H,S) \leftarrow step(S), & (12) \\ default(D,H,M), & \\ NC = \#count\{L : member(M,L)\}, & \\ \#count\{L : member(M,L), h(L,T)\} == NC. & \\ ab(D,H,S) \leftarrow step(S), & (13) \\ applicable(D,H,S), & \\ applicable(D_1,\overline{H},S), & \\ not\ defeated(D_1,\overline{H},S). & \\ defeated(D,H,S) \leftarrow step(S), & (14) \\ applicable(D,H,S), & \\ h(\overline{H},S). & \\ defeated(D,H,S) \leftarrow step(S), & (15) \\ applicable(D,H,S), & \\ applicable(D_1,\overline{H},S), & \\ prefer(D_1,D,S). & \\ h(H,S) \leftarrow applicable(D,H,S), & (16) \\ not\ ab(D,H,S), & \\ not\ defeated(D,H,S). & \end{array}$$

  Rule (12) defines when a default is *applicable*, i.e., when its body is satisfied. The interaction between defaults and rules in the knowledge base are dealt with using rules (13)—(15). (14) dismisses the applicability of (i.e., *defeats*) a default if the complement of its conclusion is already established. On the other hand, rule (15) expresses that a default is defeated if there is a more preferred default with conflicting conclusion that is applicable. Rule (13) enforces the principle **(SK)**. It states that a default d should be blocked if there is another default with the conflicting conclusion ($\overline{h}$), which is not defeated. Finally, rule (16) enforces the application of any default that is applicable and not otherwise blocked.

- For reasoning about preferences of the form (8), we have the following rule:

$$\begin{array}{l} prefer(D_1,D_2,S) \leftarrow step(S), \\ \quad prefer(D_1,D_2,M), \\ \quad NC = \#count\{L : member(M,L)\}, \\ \quad \#count\{L : member(M,L), h(L,S)\} == NC. \end{array} \quad (17)$$

  This rule defines when a preference among two defaults can be applied.

In summary, for a $KB = \langle O, RD\rangle$, $\Pi(KB)$ consists of rules (9)–(17) and the set of facts encoding RD and O.

Example 3 *The story in Example 2 can be represented by the* $KB_1 = \langle O_1, RD_1\rangle$ *where*

$$O_1 = \left\{ \begin{array}{l} observed(in\_college,1). \\ observed(has\_scholarship,2). \end{array} \right\}, \text{and}$$

$$RD_1 = \left\{ \begin{array}{l} default(d_1, \neg poor, [in\_college]). \\ default(d_2, poor, [has\_scholarship]). \end{array} \right\}$$

The statement in Example 2 can be represented by

$$statement(poor, 0).$$

Consider the program $\Pi(KB_1)$ *with different time steps:*

- *For* $k = 0$*: since there is no observation at this step, the body of* (9) *is not true and thus no default is applicable. The KB does not have any rule (of the form* (6)*). As such, no answer set of* $\Pi(KB_1)$ *contains any atom of the form* $h(l, 0)$ *where* l *is a literal.*
- *For* $k = 1$*: given* $observed(in\_college, 1)$*, it is easy to see that* (9) *will result in* $h(in\_college, 1)$ *being in every answer set of* $\Pi(KB_1)$*. This will lead to the applicability of default* d_1*.* d_2 *is not applicable. As such, every answer set of* $\Pi(KB_1)$ *contains* $h(in\_college, 1)$ *and* $h(\neg poor, 1)$*.*
- *For* $k = 2$*: it is easy to see that* (9) *will result in* $h(in\_college, 1)$*,* $h(in\_college, 2)$*, and* $h(has\_scholarship, 2)$ *being in every answer set of* $\Pi(KB_1)$*. This will lead to the applicability of default* d_1 *at the steps 1 and 2.* d_2 *is not applicable at step 1 but it is applicable at step 2. As such, every answer set of* $\Pi(KB_1)$ *contains also* $h(\neg poor, 1)$*. However, none of the answer set would contain* $h(poor, 2)$ *or* $h(\neg poor, 2)$*.*

Given a $KB = \langle O, RD\rangle$ over $\mathcal{L}$, an integer k, and a $statement(l, s)$, we are interested in determining the truthfulness of the statement is true at time steps $s \leq t \leq k$. This is defined as follows.

Definition 5 *Let* $KB = \langle O, RD\rangle$ *be a knowledge base and a statement over a literal* l *at the time step* s*,* $statement(l, s)$*. We say that*

- $statement(l, s)$ *is* true *w.r.t.* KB *at time step* $t \geq s$*, denoted* $KB \models +statement(l, s)@t$*, if for every answer set* A *of* KB*,* $h(l, t) \in A$*.*
- $statement(l, s)$ *is* false *w.r.t.* KB *at time step* $t \geq s$*, denoted* $KB \models -statement(l, s)@t$*, if for every answer set* A *of* KB*,* $h(\bar{l}, t) \in A$*.*
- $statement(l, s)$ *is* unknown *w.r.t.* KB *at time step* $t \geq s$*, denoted by* $KB \not\models \pm statement(l, s)@t$*, if* $KB \not\models +statement(l, t)$ *and* $KB \not\models -statement(l, t)$*.*

Intuitively, $KB \models +statement(l, s)@t$ (resp. $KB \models -statement(l, s)@t$) says that at the time step t the statement is true (resp. false). $KB \not\models \pm statement(l, s)@t$ states that there is no information that supports or denies the statement at the time step t. Observe that this entailment can be computed in ASP by the following rules:

$$true(H, T) \leftarrow statement(H, S), \quad (18)$$
$$step(T), T \geq S, h(H, T).$$
$$false(H, T) \leftarrow statement(H, S), \quad (19)$$
$$step(T), T \geq S, h(\overline{H}, T).$$
$$unknown(H, T) \leftarrow statement(H, S), \quad (20)$$
$$step(T), T \geq S,$$
$$not\ h(H, T),\ not\ h(\overline{H}, T).$$

Let $\Pi_Q = \Pi(KB) \cup \{(18)–(20)\}$. It can be shown that $KB \models +statement(l, s)@t$, $KB \models -statement(l, s)@t$, and $KB \models \pm statement(l, s)@t$ correspond to $\Pi_Q \models true(l, t)$, $\Pi_Q \models false(l, t)$, and $\Pi_Q \models unknown(l, t)$, respectively. This means that we can compute the truthfulness of a statement by making two calls to an ASP-solver. For example, if the program

$$\Pi_Q \cup \{\leftarrow\ not\ false(l, t),\ not\ unknown(l, t).\}$$

does not have an answer set and the program

$$\Pi_Q \cup \{\leftarrow\ not\ true(l, t)\}$$

has an answer set then we can conclude that $\Pi_Q \models true((l, t)$. It is easy to see that the following holds for KB_1:

- $KB_1 \not\models \pm statement(poor, 0)@0$;
- $KB_1 \models -statement(poor, 0)@1$; and
- $KB_1 \not\models \pm statement(poor, 0)@2$.

Proposition 1 *For each consistent* KB*, there exists no answer set of* $\Pi(KB)$ *that contains* $h(l, t)$ *and* $h(\bar{l}, t)$ *for some literal* l *and time step* t*.*

For the sake of our discussion, let us consider $KB_2 = \langle O_1, RD_1 \cup \{prefer(d_2, d_1, [])\}\rangle$. It is easy to see that every answer set of $\Pi(KB_2)$ contains $defeated(d_1, \neg poor, 2)$. As such, we have that

- $KB_2 \not\models \pm statement(poor, 0)@0$;
- $KB_2 \models -statement(poor, 0)@1$; and
- $KB_2 \models +statement(poor, 0)@2$.

With Observations about Action Occurrences

We will now extend our signature with a set of actions $\mathcal{A}$ that can be observed. We assume that each action is associated with a set of literals, called its *preconditions*, and a set of effects (Gelfond and Lifschitz 1998). This information is encoded in statements of the following form:

$$executable(a, body) \quad (21)$$

and a set of effects of the following form

$$causes(a, p, body) \quad (22)$$

where a is an action, p is a literal, $body$ is a set of literals. We will represent an *action occurrence observation* (or *action occurrence*) by a statement of the form

$$occurred(a, s) \quad (23)$$

where $a \in \mathcal{A}$ and s is a non-negative integer. The notion of a knowledge base is extended to allow action occurrence observations in a straightforward way: a knowledge base is a tuple $KB = \langle O, RD\rangle$ where O is a set of observations and action occurrences and RD is a collection of rules, defaults, preferences, and action descriptions. To reason with action occurrences, we add to the rules developed in the previous subsection the rules to deal with action occurrences in O. We will also use the encoding of a set of literals used for rules and defaults in the previous section. Intuitively, an action occurrence can be viewed as a set of observations and can be encoded easily as follows.

$$observed(L, S) \leftarrow step(S), occurred(A, S), \quad (24)$$
$$executable(A, M), member(L, M).$$

$$observed(L, S+1) \leftarrow step(S), occurred(A, S),$$
$$causes(A, L, M),$$
$$NC = \#count\{L : member(M, L)\},$$
$$\#count\{L : member(M, L), h(L, T)\} == NC. \quad (25)$$

Abusing the notation, we will use $\Pi(KB)$ to denote the program consisting of the facts representing O and RD and the rules developed in the previous section and (24)—(25). The definition of entailment of a statement at a time step is extended to knowledge bases with action occurrences in a trivial way.

Supporters and Denials

The previous section shows that we can compute the truthfulness of a statement given a set of observations and a set of defaults, preferences, and rules. In this section, we will define the notion of a supporter (resp. a denial) of a statement. Intuitively, a supporter (denial) will help an observer in determining the truthfulness of a given statement with respect to her knowledge base by identifying the set of observations that the observer needs to have. Given a $KB = \langle O, RD\rangle$, we will use now to denote a specific time step such that for every $observed(l, t)$ or $occurred(a, t)$ in O, $now > t$. For a consistent set of literals L, let

$$OBS(L) = \{observed(l, now) \mid l \in L\}.$$

The notion of a supporter (denial) is defined next.

Definition 6 *Let $KB = \langle O, RD\rangle$ be a knowledge base. Let L be a consistent set of literals.*

- *L is said to be a* supporter *of a literal l with respect to KB if $KB \cup OBS(L) \models +statement(l,0)@now$; and*
- *L is said to be a* denial *of a literal l with respect to KB if $KB \cup OBS(L) \models -statement(l,0)@now$*

where $KB \cup OBS(L) = \langle O \cup OBS(L), RD\rangle$.

L is a minimal supporter (or denial) *of l with respect to KB if it is a subset minimal supporter (or denial) of l with respect to KB.*

The intuition behinds the above definition is clear: a supporter (denial) of a literal l is the set of literals whose truth values need to be established before the truthfulness of a statement about l can be determined. Of course, if an observer can observe every literal in $\mathcal{L}$ then she would be able to determine the truthfulness of a statement about l. However, it is desirable to consider minimal supporters (denials). We next discuss possible way to compute minimal supporters (denials) of a literal l.

Computing Minimal Supporters/Denials

A minimal supporter/denial of a literal l with respect to KB can be computed using $\Pi(KB)$ and a few extra rules that generate observations and evaluate the truth value of l at the step now.

$$\{observed(P, now); observed(\neg P, now)\}1 \leftarrow fluent(P). \tag{26}$$

$$nr\_observed(NC) \leftarrow NC = \#count\{L : observed(L, now)\}. \tag{27}$$

$$\#minimize\{NC : nc\_observed(NC)\}. \tag{28}$$

Let $P(l)$ be the program consisting of $\Pi(KB)$, the rules (26)-(28), and the following rule:

$$\leftarrow \ not\ h(l, now). \tag{29}$$

Let $N(l)$ be the program consisting of $\Pi(KB)$, the rules (26)-(28), and the following rule:

$$\leftarrow \ not\ h(\bar{l}, now). \tag{30}$$

It is easy to see that the following holds:

Proposition 2 *For a $KB = \langle O, RD\rangle$ and a literal l,*

- *For every answer set A of $P(l)$, the set $L = \{l \mid observed(l, now) \in A\}$ is a minimal supporter of l with respect to KB.*
- *For every answer set A of $N(l)$, the set $L = \{l \mid observed(l, now) \in A\}$ is a minimal denial of l with respect to KB.*

Observe that $P(l)$ (resp. $N(l)$) computes cardinality minimal supporters (resp. denials) of l with respect to KB. Since minimal cardinality implies subset minimality, the proposed approach is sound, but not vice versa. As such, $P(l)$ (resp.$N(l)$) does not compute all minimal supporters (resp. denials).

A CR-Prolog Based Encoding

We explore the use of CR-Prolog to address the issue of completeness in the computation of the minimal supporters and denials. CR-Prolog extends ASP by introducing an additional type of rules, called *consistency restoring rules* (or *cr-rules*), of the form

$$r: \quad c \stackrel{+}{\leftarrow} a_1, \ldots, a_m, \ not\ a_{m+1}, \ldots, \ not\ a_n \tag{31}$$

where r is the name of the rule and c and a_j's are literals as in the rule (1). Observe that a cr-rule can be viewed as a normal rule by dropping its name and replacing the connective $\stackrel{+}{\leftarrow}$ with $\leftarrow$. As such, we refer to $head(r)$, $pos(r)$, and $neg(r)$ for a cr-rule r in the same way we refer to different elements of a normal rule.

A CR-program P is given by a pair (P^r, P^c) where P^r is a set of rules of the form (1) and P^c is a set of rules of the form (31). Let C be a subset of P^c. By $P^r \cup C$ we denote the program consisting of rules in P^r and the cr-rules in C viewed as normal rules.

Answer sets of a CR-program P are defined as follows. If P^r is consistent, then any answer set of P^r is an answer set of P. Otherwise, an answer set of P is an answer set of $P^r \cup C$ where C is a minimal subset of P^c such that $P^r \cup \{c \leftarrow body(r) \mid c \stackrel{+}{\leftarrow} body(r) \in C\}$ is consistent.

Let $\Pi^+(KB)$ (resp. $\Pi^-(KB)$) be the program consisting of $\Pi(KB)$, the rule (29) (resp. the rule (30)). Let Π^C be the set of cr-rules

$$observed(P, now) \stackrel{+}{\leftarrow} fluent(P). \tag{32}$$

$$observed(\neg P, now) \stackrel{+}{\leftarrow} fluent(P). \tag{33}$$

We can show the following

Proposition 3 *For a $KB = \langle O, RD\rangle$ and a literal l,*

- *A set of literals L is a minimal supporter of l with respect to KB iff there exists an answer set A of $(\Pi^+(KB), \Pi^C)$ such that $L = \{l \mid observed(l, now) \in A\}$.*
- *A set of literals L is a minimal denial of l with respect to KB iff there exists an answer set A of $(\Pi^-(KB), \Pi^C)$ such that $L = \{l \mid observed(l, now) \in A\}$.*

Applications

One interesting application of our framework is the detection of *Man-in-the-Middle* (MITM) attacks targeting computer and cyber-physical systems. In a MITM attack, the attacker secretly places itself as an intermediary between two communicating parties, relaying the information between them. By intercepting the communications, the attacker may steal valuable information, or even alter the information exchanged between the parties and fool them into performing unintended or undesirable actions.

For example, a MITM attack was used in Stuxnet[3], a sophisticated malicious software (malware) that targeted certain models of industrial Programmable Logic Controllers (PLCs). Stuxnet is remarkable in that it is reported to have been successful in impacting industrial systems involved in Iran's nuclear enrichment program. Obviously, its success has major implications on the security and safety of industrial systems world-wide.

[3] https://en.wikipedia.org/wiki/Stuxnet

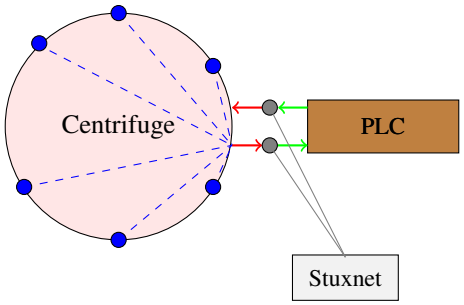

Figure 1: Outline of the MITM attack carried out by Stuxnet: safe commands sent by the PLC (green) are replaced by dangerous ones (red); alarming readings from the centrifuge's sensors (red) are replaced by seemingly safe ones.

The behavior of Stuxnet's MITM component is outlined in Figure 1. The MITM component operated by intercepting the commands sent by a PLC to a connected centrifuge. The malware first increased the speed of the centrifuge above normal levels for a short amount of time, and later slowed it down below normal levels for a longer period of time. It is believed that the resulting stress caused components of the centrifuge to expand and eventually destroy it. Under normal conditions, sensors installed in the centrifuge would have alerted the PLC – and its users – about the abnormal conditions, giving them a chance to shut down the system before damage occurred. However, as part of the MITM attack, Stuxnet also intercepted the sensor readings from the centrifuge, and sent to the PLC fake readings based on previous recordings that indicated that the system was operating normally (this is known as a *replay attack*).

A seriously concerning feature of MITM attacks is their ability to take control of the involved parties' inputs and outputs, making the attack virtually undetectable to the parties. In this section, we show that detection of a MITM attack is indeed possible if the detection task is reduced to that of reasoning about the truthfulness of a communication partner, as long as one has access to some external knowledge that can be used as supporter or denial of the partner's statements.

To demonstrate this, we consider a simplified MITM attack scenario along the lines of Stuxnet's MITM component. For simplicity of presentation, we do not include in the scenario occurrences of actions, but it is not difficult to see that our approach extends in a natural way when occurrences of actions are present

Consider the case of a motor, M, that can be on or off. A sensor mounted on the motor tells whether the motor is overheating (fluent $overheat$). A controller, C, is programmed to turn off the motor if it is found to be overheating.

Suppose now that the system is the target of a MITM attack. An attacker, A, manages to place itself between C and M, intercepting the communications between them. A intercepts the output of M's sensor, discarding the sensor reading and always providing C with a reading of $\neg overheat$ independently of the actual state of M. In doing so, the attacker could prevent C from turning off an overheating motor, eventually causing it to become damaged. How can such an attack be detected?

The solution leverages a technique for reasoning about cyber-physical systems and their interaction with the physical environment discussed in (Nedelcu and Balduccini 2015). Suppose that, located near the motor, is a thermostat, T, which generates an acoustic alert every 20 minutes if the temperature of the room is above 90°F in order to ensure that the operators take more frequent breaks. World knowledge furthermore tells us that, during a cold season, it is unlikely for the temperature in the room to be above 90°F.

The thermostat is not related to the functioning of the motor and, thus, it is conceivable that A will not attempt to alter its activities. However, given the proximity of the motor, an alert from the thermostat when the room is not hot can be taken as an indication that M is indeed overheating (and that M's sensor reading may have been tampered with). Let us see how one can draw this conclusion using our framework.

The relevant information can be formalized using the following statements:

$$default(d_1, hot\_room, [alert]).$$
$$default(d_2, \neg hot\_room, [cold\_season]).$$
$$prefer(d_2, d_1).$$
$$rule(r_1, cold\_season, [winter]).$$
$$default(d_3, overheat, [alert, \neg hot\_room]).$$

The first statement, a default, says that, normally, an alert from T indicates that the room is hot. The second statement, also a default, states that, normally, the room is not hot during a cold season. The third statement expresses a preference for d_2 when both d_1 and d_2 are applicable. This captures the intuition that, during the cold season, one will have a tendency to assume that the room is not hot even if an alert is heard. The fourth statement, a rule, encodes the world knowledge that winter is a cold season. Finally, the last statement says that, typically, if an alert is generated by T when the room is not hot, then M is overheating. (The statement is encoded as a default to increase elaboration tolerance, making it possible, for example, to take into account faults in the thermostat or the presence of other heat sources.)

Let us now suppose that we would like to evaluate the truthfulness of M's sensor reading, and suppose that the sensor reports $\neg overheat$. The corresponding statement is:

$$statement(\neg overheat, 0).$$

Next, an alert is generated by T:

$$observed(alert, 1).$$

Let KB_m be the corresponding knowledge base. Clearly, default d_2 is not applicable and d_1 leads us to conclude that the temperature in the room is above 90°F. Thus, the reasoner has no reason to doubt the sensor reading:

$$KB_m \models +statement(\neg overheat, 0)@1$$

Next, the system is informed that it is winter. The updated knowledge base, KB'_m, extends KB_m by the statement:

$$observed(winter, 2).$$

Our framework now yields:

$$KB'_m \models -statement(\neg overheat, 0)@2$$

That is, the sensor reading from M is deemed not truthful, which indicates that the system may be under a MITM attack.

Finally, one can also reason about supporters and denials of $\neg overheat$. Given KB_m and the above fluents, for example, it is not difficult to show that the CR-Prolog encoding given earlier yields $\{\ \}$ as the minimal supporter of $\neg overheat$ and $\{winter\}$ as its minimal denial.

Discussion

Within the scope of the ASP and logic programming community, this work is to the best of our knowledge completely novel. The AI community has explored the issue of computational trust and reputation in several works—please see (Sabater and Sierra 2005) for a survey. The survey focuses predominantly on trust models observed in multi-agent scenarios, taking explicitly into account the observations concerning interactions among agents. The survey provides classification of the models according to different dimensions: conceptual model (cognitive vs. game theoretical), information sources (direct interactions, direct observations, witness information, sociological information, prejudice), visibility (subjective vs. global), granularity (context dependent vs. non-context dependent), model type (trust vs. reputation), type of information exchanged (boolean vs. continuous), and agent behavior's assumptions (honest, biased but not lying, lying). Within such classification, our model focuses on trust (but, as mentioned in our discussion, could accommodate interesting forms of reputation), based on cognitive aspects, it builds on direct observations, but can accommodate sociological biases and prejudice through defaults, it captures subjective visibility, it is context dependent and relies on boolean information.

The survey by Artz and Gil (Artz and Gil 2007) places a greater emphasis on surveying models of trust as models to predict attitude towards future interactions with an agent—with less emphasis on assessing the trustworthiness of a current statement.

Relaxing Consequence

One of the advantages of using a declarative language like ASP is the ability to explore alternative reasoning strategies. The proposed encoding is skeptical in the way it handles the assessment of statements—by considering a statement true when supported by *all* answer sets and false when its negation is supported by all answer sets. There are variations of this approach that could be easily modeled. For example, a trusting observe may want to accept the statement of the agent as long as this is not explicitly contradicted.

Definition 7 *Let $KB = \langle O, RD\rangle$ be a knowledge base and let us consider a statement $statement(l, s)$. We say that:*

- *$KB \models +statement(l, s)@t$ if for every answer set A of KB we have that $h(\bar{l}, t) \notin A$;*
- *$KB \models -statement(l, s)@t$ if every answer set A of KB contains $h(\bar{l}, t)$;*
- *the statement is otherwise unknown.*

This definition, applied to Example 3, allows us to conclude:

- $KB_1 \models +statement(poor, 0)@0$;
- $KB_1 \models -statement(poor, 0)@1$; and
- $KB_1 \models +statement(poor, 0)@2$.

Time-Annotated Knowledge Bases

The components of a knowledge base—as defined in the previous section—relate properties of the world at a single time instance. It is easy to imagine situations in which rules, defaults, and preferences could relate to information in different time points. For example, if the unit of time is months, the information "if a graduate student fails the final examination, he/she needs to wait at least 1 year before re-taking the exam" cannot be represented as a rule in the current framework since the relation between the fluent $take\_exam\_second\_time$ and the fluent $fail\_exam\_first\_time$ spans 12 units of times.

We will next propose a generalization of the proposed framework, called *time-annotated knowledge bases*, for dealing with the above issue. Let l be a literal and t be an integer. l^t is called a *time-annotated literal*. A *time-annotated* rule, default, or preference has the form (6), (7), or (8), respectively, whose $head$ and $body$ are now time-annotated literals and sets of time-annotated literals. For example, the relation between the two fluents $take\_exam\_second\_time$ (ts) and $fail\_exam\_first\_time$ (ff) can be represented by the set of rules:

$$\left\{rule(exame\_time, (\neg ts)^i, [ff^0]) \mid i = 1, \ldots, 11\right\}$$

Given a time-annotated knowledge base KB, the program $\Pi(KB)$ defined in the previous section can be easily modified for computing the truthfulness of statements. In fact, in all the rules of $\Pi(KB)$, $h(L, S)$ is replaced with $h(U, S+T)$ if L is the time-annotated literal U^S. As an example, the rule (11) becomes

$$\begin{array}{l} h(H, S+T) \leftarrow step(S), rule(D, H^T, M), \\ \quad NC = \#count\{L^T : member(M, L^T)\}, \\ \quad \#count\{L^T : member(M, L^T), h(L, T+S)\} == NC. \end{array} \quad (34)$$

A System for Reputation Evaluation

Given that reputation of agents is exhibited on what they say and what they do, the proposed framework can be used for evaluating agents' reputation if we have a rule (or a set of rules) for the evaluation, i.e., what does it mean for an agent to have excellent, good, or bad reputation. A reasonable rule for such a purpose could be devised using the ratio between the truthful and non-truthful statements made by agents, e.g., an agent has an excellent reputation if the ratio between her truthful and non-truthful statements is 99%. We note that this percentage is likely dependent on individual observer, the agent, and the application. For instance, it might be okay to classify a saleman as excellent if the percentage is 70% but this thresthold would be insufficient for a doctor to be called as excellent.

Given a knowledge base KB, a set of statements S made by an agent, and a classification rule of the form "an agent is of the type $xType$ if the ratio between her truthful and non-truthful statements is $y\%$," the program $\Pi_Q(KB)$ can be extended for determining whether or not the agent is of the type $xType$ in the following way.

$$\begin{aligned} reputation(xType, T) \leftarrow\ & step(T), \\ & NT = \#count\{L, T : true(L, T)\}, \\ & NF = \#count\{L, T : false(L, T)\}, \\ & NT > 0, NT * 100 \geq NF * y. \end{aligned} \tag{35}$$

where we assume that y is given by an integer ranging between 0 and 100.

Observe that this rule determines the reputation of the agent at the step T. The rule could be changed to compute the reputation of an agent over the full history considered by the program. Using the same method for computing the truthfulness of agents' statements, we can answer the question of whether or not the agent is of the type $xType$.

Let $\Pi_R = \Pi_Q \cup \{(35)\}$. If the program $\Pi_R \cup \{\leftarrow not\ reputation(xType, t)\}$ has an answer set and the program $\Pi_R(KB) \cup \{\leftarrow reputation(xType, t)\}$ has no answer set then we can conclude that the agent is of the type $xType$ at the time step t.

Final Remark

Let us conclude with a final remark. The proposed framework bears similarity to the ASP frameworks developed to support *diagnosis*. In particular, both frameworks focus on using observations (and an underlying logical theory) to infer the correctness of some entity—a statement in our framework, a system in the case of diagnosis. On the other hand, there is a profound difference between the two frameworks: unlike diagnosis, our framework does not rely on a well elaborated theory of how the agent works. In particular, our frameworks does not make any assumption about completeness of knowledge of the agent (e.g., in the form of a complete model).

Conclusions

In this paper, we presented a framework based on Answer Set Programming to reason about the truthfulness of statements made by an agent; the framework does not assume complete knowledge about the agent being observed and the reasoning process builds on observations about the state of the world and occurrences of actions. We explored the use of the framework in simple scenarios derived from man-in-the-middle attacks.

The framework represents a starting point for the exploration of ASP-based methods to reason about trust and reputation of agents. The literature has explored a variety of models to describe the reputation of agents—our next step will embed such models in ASP, thus offering a knowledge-based approach to study agent's behavior in presence of incomplete knowledge. We will also explore alternative approaches assess truthfulness of statements, reflecting different attitudes of the observer.

References

Artz, D., and Gil, Y. 2007. A Survey of Trust in Computer Science and the Semantic Web. *Journal of Web Semantics* 5:58–71.

Baral, C. 2003. *Knowledge Representation, reasoning, and declarative problem solving with Answer sets*. Cambridge University Press, Cambridge, MA.

Citrigno, S.; Eiter, T.; Faber, W.; Gottlob, G.; Koch, C.; Leone, N.; Mateis, C.; Pfeifer, G.; and Scarcello, F. Sep 1997. The dlv system: Model generator and application frontends. In Bry, F.; Freitag, B.; and D., S., eds., *Proceedings of the 12th Workshop on Logic Programming WLP*, 128–137.

Faber, W.; Leone, N.; and Pfeifer, G. 2004. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In Alferes, J. J., and Leite, J. A., eds., *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings*, volume 3229 of *Lecture Notes in Computer Science*, 200–212. Springer.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. clasp: A conflict-driven answer set solver. In Baral, C.; Brewka, G.; and Schlipf, J., eds., *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Artificial Intelligence*, 260–265. Springer-Verlag.

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. 1998. Action Languages. *Electronic Transactions on Artificial Intelligence* 3(6).

Nedelcu, A., and Balduccini, M. 2015. An Approach and Tool for Reasoning about Situated Cyber-Physical Systems. In *The First Workshop on Action Languages, Process Modeling, and Policy Reasoning (ALPP 2015)*.

Niemelä, I.; Simons, P.; and Soininen, T. 1999. Stable model semantics for weight constraint rules. In *Proceedings of the 5th International Conference on on Logic Programming and Nonmonotonic Reasoning*, 315–332.

Pelov, N.; Denecker, M.; and Bruynooghe, M. 2004. Partial stable models for logic programs with aggregates. In *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings*, volume 2923 of *Lecture Notes in Computer Science*, 207–219. Springer.

Sabater, J., and Sierra, C. 2005. Review on Computational Trust and Reputation Models. *Artificial Intelligence Review* 24:33–60.

Son, T., and Pontelli, E. 2007. A Constructive Semantic Characterization of Aggregates in Answer Set Programming. *Theory and Practice of Logic Programming* 7(03):355–375.