Formalizing Informal Logic and Natural Language Deductivism *

Gopal Gupta, Sarat Varanasi, Kinjal Basu, Zhuo Chen
Elmer Salazar, Farhad Shakerin, Serdar Erbatur, Fang Li, Huaduo Wang
The University of Texas at Dallas, Richardson, USA
Joaquín Arias
Universidad Rey Juan Carlos, Madrid, Spain
Brendan Hall, Kevin Driscoll
Honeywell Corp, Minneapolis, MN

September 15, 2021

Abstract

Formalizing the human thought process has been considered fiendishly difficult. The field of *informal logic* has been developed in recognition of this difficulty. Work in informal logic interprets an argument as an attempt to present evidence for a conclusion. Holloway and Wasson have developed a primer to establish the terms, concepts, principles, and uses of arguments. We argue that recent advances in formal logic, especially incorporation of negation as failure, facilitate the formalization of the human thought process. These advances help formalize concepts that were hitherto thought of as impossible to formalize. We show how the paradigm of answer set programming can be used to formalize *all* the concepts presented in Holloway and Wasson's primer.

I don't see that human intelligence is something that humans can never understand.

— John McCarthy, March 1989

1 Introduction

Formalizing the human thought process has been considered very hard. The study of human thought process has been conducted over several millenia [17, 10]. In modern times this effort culminated in boolean logic [11], first order logic [14], and various other advanced logics. These logics, however, are limited and could not match the sophistication of human reasoning in the sense that it is hard to use these logics to faithfully model the human thought process in an elegant manner. First, we make number of points [19]:

- The early problems of naive set theory found by Russell (Russell's paradox [6]) led mathematicians and logicians to only focus on well-founded or inductive reasoning which stipulates that to reason soundly one has to start from the simplest object (e.g., an empty set) and then build larger objects by embellishing this simplest object (i.e., obtain one element sets by adding an element to the empty set, two element sets by adding an element to one element sets and so on). Thus, assumption-based reasoning that humans frequently employ, and which requires *circular* or *coinductive reasoning* [1], was banished from mathematical discourse. Only recently, work on coninductive reasoning has been taken up [1, 6, 33, 20].

*Work partially supported by NSF awards IIS 1718945, IIS 1910131, IIP 1916206, and by DoD and Amazon.

- Systems of logic could not reason about themselves, i.e., be reflective. Part of the reason is the utter focus on only allowing inductive structures. Meta-reasoning was disallowed due to fears of unsoundness and circularity. As a result, for example, classical logics cannot have the ability to predicate a conclusion based on failure of a proof in that logic itself. In fact, Tarski stipulated that given a logic L_1, we need another logic L_2 to reason about L_1, and yet another logic L_3 to reason about L_2, and so on, *ad infinitum* [34]. Thus, Tarski deemed it impossible for a language to have its own *truth predicate*. Only in 1975 did Kripke show that a language can consistently contain its own truth predicate [27].

- The concept of *negation as failure* [2] was added into logic along with notion of stable model semantics that admitted multiple worlds [16]. Negation-as-failure allows us to *take an action if a proof fails*: a notion frequently employed by humans (if something does not work out, do something else). Classical logic (based on inductive semantics) cannot reason about proof failure. If, for example, we program reachability (of one node from another in a directed graph) in logic, then the axioms for *reachability* cannot be used to prove *unreachability*. Axioms for unreachability have to be given separately to reason about unreachability. With negation-as-failure, we can easily realize unreachability by stating that if the proof of reachability from node N1 to N2 fails, then node N2 is unreachable from node N1.

- While coinductive reasoning and negation as failure have been around for 30-40 years, they did not lead to formalization of the human thought process. The advent of answer set programming based on the idea of negation-as-failure [15, 28] made this possible, where complex human thought processes such as default reasoning, counterfactual reasoning, nonmonotonic reasoning, abductive reasoning and possible-worlds reasoning could now be realized in a formal framework in an elegant manner. Progress was still limited by the type of implementations available for answer set programming that precluded goal-directed thinking. The design of goal-directed or query-driven ASP execution engines such as s(ASP) [29] and s(CASP) [4] solved this problem.

- Formalization of the human though process has been further compounded by the fact that humans may use the same linguistic pattern to represent different logical representations. This is made evident in Wason's Selection Task [25] where *if A then B* may be used by humans to represent both $A \Rightarrow B$ and $A \iff B$.

Given the lack of success of classical logic in formalizing the human though processes, work was started on study of "informal logic" [18]. Similar to classical logic, work in informal logic is focused on inference rules (called argument in the informal logic literature) and reasoning (proof). An argument is an attempt to present evidence for a conclusion (or a claim) and it relies on premises that support the conclusion. Halloway and Wasson have developed an excellent primer on this subject [24]. We show how all the terms in the primer can be mapped to answer set programming constructs and how arguments, evidence, etc., can be represented and executed on our s(CASP) ASP engine. The ASP code for various examples of the primer is shown later and the output of the s(CASP) system for the respective query of each example is shown in the Appendix. The output includes the computed answer, the model, and the proof trace. A more comprehensive example from the argumentation literature has also been worked out.

As an aside, it should be noted that ASP/s(CASP) technology can automate the overarching properties framework [21]. The OAP framework envisions that to have confidence in a system, establishing three *overarching* properties of the system suffices: *intent, correctness, and innocuity* [23].

2 Mapping Informal Logic to ASP

We first cast all the terms used in the primer in terms of those used in logic, logic programming (LP) and answer set programming (ASP). Terms from the primer are italicized while those from LP/ASP are in boldface font.

- A *conclusion* is a **theorem**. In LP/ASP, it translates into a **query**. The word *claim* is also used sometimes in assurance literature. A *claim* is also a **theorem** or a **query** in LP/ASP terminology.

- An *argument* is a **clause** (i.e., a rule).

- A *premise* ("what an *audience believes*") is a belief based on knowledge or assumption. A *premise* can be of two types: *evidence* or *assumption*. An *evidence* is a **fact**. An assumption is an **abducible**.
- *Reasoning* corresponds to a **proof**. To perform *reasoning*, in LP/ASP we develop a **proof** tree.
- A *binding* is a **substitution**, i.e., value imparted to a variable that is existentially or universally quantified. It is no different from what is understood to be a binding in LP/ASP.
- A defeater is a negated goal (negation here is negation-as-failure, as it "provides support for *not believing*"). Note that ASP has two types of negation. The second type, **classical negation**, represented as −p, is a *premise*.

Before we go on to mapping other concepts of the primer to ASP, we give a brief introduction to abductive reasoning. The term abduction refers to a form of reasoning that is concerned with the generation and evaluation of explanatory hypotheses. We could also think of abduction as assumptions based reasoning. Abductive reasoning leads back from facts to a proposed explanation of those facts or assumptions that will explain that fact. According to Harman [22], abductive reasoning takes the following form:

```
The fact B is observed.
But if A were (assumed) true, B would be a matter of course.
Hence, there is reason to suspect that A is true.
```

In this form, `B` can be either a particular event or an empirical generalization. `A` serves an explanatory hypothesis and `B` follows from `A` combined with relevant background knowledge. Note that `A` is not necessarily true, but plausible and worthy of further validation. We can also think of `A` as an assumption that we must make to explain the observation `B`. A simple example of abductive reasoning is that one might attribute the symptoms of a common cold to a viral infection. Or, that if we assume viral infection, then no wonder the person has symptoms of a cold. More formally, abduction is a form of reasoning where, given the premise $P \Rightarrow Q$, and the observation Q, one surmises (abduces) that P holds. More generally, given a theory T, an observation O, and a set of abducibles A, then E is an explanation of O (where $E \subset A$) if:

1. $T \cup E \models O$
2. $T \cup E$ is consistent

We can think of abducibles A as a set of assumptions. Generally, A consists of a set of propositions such that if $p \in A$, then there is no rule in theory T with p as its head (that is, there is no way to argue for p).

We assume the theory T to be an answer set program. Under a goal-directed execution regime, an ASP system can be extended with abduction by simply adding the following rule for an abducible p:

p :- not _not_p.
_not_p :- not p.

this is automatically achieved for a predicate p that we want to declare as an abducible in the s(CASP) system through the declaration:

#abducible p.

We now proceed to give the rest of the mapping from overarching properties (OAP) to ASP.

Atomic and Compound Arguments: An *atomic argument* is a **clause** (rule) that only uses **facts** and **abducibles** in its **body**. A *compound argument* is a **clause** (rule) that has facts, abducibles and a reference to other rules in its body.

Cogent Arguments: An *argument* is *cogent* if it rationally justifies believing its conclusion to the required standard of confidence [24]. How do we simulate cogency in ASP? ASP can model various shades of confidence in a conclusion by a combination of negation as failure and classical negation. Given *not* p, where *not* is interpreted as negation as failure, it will be interpreted as "no evidence of p". Similarly, *-p* denotes that p is unconditionally false, i.e., there is irrefutable evidence that p is false. Negation as failure and classical negation can be combined to create nuanced reasoning. Given a proposition `p` (e.g., `p` ≡ "it is raining now"):

1. `p`: denotes that `p` is unconditionally true.
2. `not -p`: denotes that `p` *maybe* true.
3. `not p` ∧ `not -p`: denotes that `p` is unknown, i.e., there is no evidence of either `p` or `-p`.

4. `not p`: denotes that `p` *may* be false (no evidence that `p` is true).

5. `-p`: denotes that `p` is unconditionally false.

Cogency can be represented in ASP using these five shades of truth. Thus, if we consider a criminal case vs a civil case, e.g., O.J. Simpson murder trial, then for O.J. Simpson to be acquitted, a proof of `not murdered(oj, nicole)` will be needed for the civil case, while a proof of `-murdered(oj, nicole)` will be needed for the criminal one.

3 Precepts

The Primer presents a number of guiding principles called precepts. We discuss them here in light of our ASP rendering.

Locality: The precept of locality states that the cogency of a compound argument never exceeds the cogency of its weakest atomic argument. This obviously holds in our ASP rendering, assuming rules are written in a reasonable manner following the precept of locality, and normal rules of logical inference are followed. For example, if we assert that all crows are birds (`bird(X) if crow(X)`), then if we establish that Jimmy may not be a crow (`not crow(jimmy)`), we can only conclude that Jimmy may not be a bird either and no more.

Depth: The precept of depth states that "argument decomposition must descend far enough to serve stakeholder objectives, and not so far as to to unnecessarily consume resources, create distraction, ...". In ASP rendering this translates to the level of granularity to which the modeling is done. When we give arguments to have our audience believe a certain conclusion, we furnish a proof. The proof tree serves as the justification for the conclusion. The proof tree can be displayed to the depth desired to keep the explanation high level, even though our modeling and reasoning may be very low level. The interactive proof viewing facility of s(CASP), for example, allows one to explore the proof tree to any level of detail [3].

Change: Change relates to the fact that the context in which an argument is made may change. Arguments may be made to assure that a system is safe before deployment. However, post deployment, a new set of situations may be encountered and the assurance argument may fall apart. Change, thus, pertains to non-monotonicity of our knowledge. That is, a conclusion drawn now may have to be withdrawn later as new knowledge becomes available. We know that Tweety is a bird, so we conclude it can fly. However, later we discover that Tweety is a penguin, so this conclusion has to be withdrawn. ASP is based on a non-monotonic logic as it incorporates negation as failure. Thus, change is easily accommodated in our ASP rendering. As our knowledge grows post deployment, we can refine our arguments and our proofs. In fact, ASP allows for any accommodations that may have to be modeled in advance (i.e., known unknowns can be modeled). However, if it turns out that known unknowns do not arise then, that is not a problem either. In fact, ASP's major strength is being able to model a situation even when information is lacking.

Induction: The induction precept states that not all reasoning may be deductive. This can also be modeled using ASP, since we can model analogical, explanatory, defeasible, counterfactual, and various other types of reasoning, thanks to the presence of negation as failure and possible world semantics. In fact, an ASP rule captures (enumerative[1]) inductive reasoning [35] quite precisely by also stating the exceptions to an induced default rule. For example, if we see a number of swans and all of them are white, then we may induce that all swans are white. However, we can never be sure that all swans are white, so we could leave room for exceptions and code it in ASP as:

```
color(X, white) :- swan(X), not abnormal_swan(X).
abnormal_swan(X) :- black_swan(X).
```

This property of default rules has been exploited for making machine learning explainable [31].

Plausibility: The precept of plausibility states that people have biases, beliefs, etc., that will color their perception of the world. This is also easily modeled in the ASP framework through the five shades of truths mentioned above. Consider a physician who is about to prescribe a medicine to a patient. A physician with

[1]Enumerative induction is an inductive method in which a conclusion is constructed based upon the number of instances that support it [35].

aggressive thinking may immediately prescribe the medicine and if any side-effects show up later, he/she will worry about them then. In ASP, this will be modeled as:

```
prescribe(M, D, P) :- cures(M, D), not contraindicated(M, P).
contraindicated(M, P) :- positive_for_side-effects(M, P).
```

which states that medicine M can be prescribed to patient P for disease D if, normally, medicine M cures disease D, and there is no evidence of contraindications for medicine M in person P. The rule here states that the medicine should be given without testing for side-effects, however, if for some reason we know, (or we learn later, that the patient tests positive for medicine M's side-effects), then the medicine will be stopped from being prescribed. That's how reasoning in ASP will work. Note that this reasoning make sense, say, for example when we know that only 1% of the patients are allergic to the medicine so the chances of having a side-effect are very low.

In contrast, a conservatively thinking doctor may decide to first ensure in advance that medicine M does not test positive for any side effects for person P. This doctor does not want to take even a 1% chance. This conservative thinking will be modeled as:

```
prescribe(M, D, P) :- cures(M, D), not contraindicated(M, P).
contraindicated(M, P) :- not -positive_for_side-effects(M, P).
```

which states that the medicine can be prescribed *only if* the patient does not test positive for M's side-effect.

The aggressive reasoning rule is read as follows: prescribe medicine M to patient P for disease D if M cures D and contraindication of M for P *maybe* false. Since contraindication is qualified with a *maybe*, the `prescribe` goal can succeed without performing the test. The conservative reasoning rule, in contrast, is read as follows: prescribe medicine M for patient P for disease D if M cures D and patient definitely does not test positive to M's side-effect.

4 Discussion

It should be noted that ASP is especially good at default reasoning. Defaults are used by humans all the time to jump to conclusions. Defaults are statements that begin with the word *normally* (*normally*, birds fly). Humans learn defaults and then gradually learn exceptions to them (e.g., exceptions to the default rule about flying such as: penguins don't fly, wounded birds don't fly, ostriches don't fly, newly born baby birds don't fly, etc.). There may be multiple defaults in some cases, and humans learn to prefer one over the other. In fact, our biases, expertise, etc., is captured as default rules and exceptions (plus preferences over defaults) that reside in our minds. Expert knowledge is nothing but a set of defaults, exceptions and preferences about some very specialized knowledge that the expert has acquired through studying and practice over a number of years. Given that ASP is very good at representing defaults, exceptions and preferences, modeling real world situations in ASP is quite feasible. In fact, our group has built tools [12] that model a cardiologist's expertise for treating congestive heart failure using ASP. Our tool outperform cardiologists [13]. This system is based on ASP and represents complex expert knowledge found in guidelines for treating heart failure [36] as ASP rules. Similarly, ASP technology allows us to answer natural language questions against a textual passage or a graphical image by invoking common sense knowledge [8, 7].

Significant amount of research has been invested in formalizing argumentation using logic programming and answer set programming [9]. However, the logic-based modeling in all these approaches is based on propositions. In contrast, we can model claims, arguments, evidence and assumptions at the predicate level using our s(CASP) query-driven ASP engine. We next illustrate our method with a detailed example.

5 An Illustrative Example

We take an example from the work of Modgil and Prakken [30] that narrates a scenario where a person, John, is seen in Holland Park by Mary, an observer. Modgil and Prakken use this scenario to illustrate the complexity of argumentation research. We use it to demonstrate how elegantly this complex scenario can be modeled in ASP and executed in our s(CASP) system to automatically verify claims. Note that the original text presents the scenario as observed by "us". We have changed the observer to Mary.

> Suppose Mary believes that John was in Holland Park some morning and that Holland Park is in London. Then Mary can deductively reason from these beliefs, to conclude that John was in London that morning. So the reasoning cannot be attacked. However, perfection remains

unattainable since the argument is still fallible: its grounds may turn out to be wrong. For instance, Jan may tell us that he met John in Amsterdam that morning around the same time. We now have a reason against Mary's belief that John was in Holland Park that morning, since witnesses usually speak the truth. Can we retain our belief or must we give it up? The answer to this question determines whether we can accept that John was in London that morning.

Maybe Mary originally believed that John was in Holland Park for a reason. Maybe Mary went jogging in Holland Park and she saw John. We then have a reason supporting Mary's belief that John was in Holland Park that morning, since we know that a person's senses are usually accurate. But we cannot be sure, since Jan told us that he met John in Amsterdam that morning around the same time. Perhaps Mary's senses betrayed her that morning? But then we hear that Jan has a reason to lie, since John is a suspect in a robbery in Holland Park that morning and Jan and John are friends. We then conclude that the basis for questioning Mary's belief that John was in Holland Park that morning (namely, that witnesses usually speak the truth and Jan witnessed John in Amsterdam) does not apply to witnesses who have a reason to lie. So our reason in support of Mary's belief is undefeated and we accept it.

The narrative above is an excellent example of claims being made (e.g., John is in London) and then arguments and evidence being used to establish that claim. The arguments made may possibly encounter exceptions (defeaters) along the way. We will model the various scenarios using answer set programming. We will use the event calculus [32, 5] to model the situation as it evolves. We also have to make assumptions about Mary's eyesight being good, Jan and John being friends and John being a robbery suspect. We will treat these as abducibles, i.e., we will attempt to prove the claim with the assumption being true or the assumption being false. Some of the knowledge used in this example is, in fact, generated automatically from the English text above using techniques we have developed that make use of English text parsers, VerbNet and our s(CASP) system [8]. Some commonsense knowledge about various concepts that is needed is also added.

We start with Mary's claim that John was in London in the morning. This amounts to proving the query:

```
holds(in_london,morning)
```

where `in_london` represents the *fluent* that John is in London. We next represent the knowledge encapsulated in the story. Mary, John, and Jan are people. Holland Park is in London. This is represented as facts in ASP:

```
person(jan).
person(john).
person(mary).
is_in(holland_park,london).
```

Next we translate rest of the information. Consider the sentence: Mary saw John in Holland Park. This sentence is automatically translated into the following facts using our SQuARE system and VerbNet primitives (for verbs 'discover' and 'occur'):

```
discover(morning, during(see_1), agent(mary), theme(john),
            source(unknown)).
occur(morning, event(see_1), theme(john), location(holland_park)).
```

We automatically extract commonsense knowledge about an observer (Mary and John are both observers of events). Note that in ASP '_' denotes an anonymous variable (i.e., a variable whose value we don't care for).

```
observer(E,X) :- discover(_, during(E),agent(X), _, _).
```

Next we define the commonsense knowledge of an event happening. An event `E` happens at time `T` in some location with some theme.

```
happens(E,T) :- occur(T, event(E), theme(_), location(_)).
```

John's presence in London is modeled using the *fluent* `in_london`. A fluent is a variable whose value changes with time as events happen. We model the knowledge following the event calculus [32]. The fluent `in_london` is initiated to become true if `Y` is seen by `A` in London.

```
initiates(E, in_london, T) :-
     discover(T, during(E), agent(A), theme(Y), source(_)),
     occur(T,event(E), theme(Y), location(Loc)),
     is_in(Loc,london), not ab_initiates(E,in_london,T, A).
```

Note that the initiation process can be defeated due to an abnormal situation (e.g., A has poor eyesight). This is reflected in the ab_initiates predicate in the rule above.

```
ab_initiates(E,in_london,T, A) :- person(A), observer(E,A),
                                  not accurate(A,sense).
```

Accuracy of a person's senses is modeled as a rule. A person X's senses are accurate, unless X is old.

```
accurate(X,sense) :- person(X), not ab_accurate(X,sense).
ab_accurate(X,sense) :- person(X), age(X,A), A = old.
```

Next, we automatically generate knowledge about Jan seeing John in Amsterdam.

```
perceive(morning, during(meet_1),experiencer(jan),stimulus(john)).
occur(morning, event(meet_1), theme(john), location(amsterdam)).
```

We next complete the definition of the fluent in_london: if Jan sees John in Amsterdam, John cannot be in London. The definition is completed by defining the terminate primitive of the event calculus for the fluent in_london. We state that event E *terminates* the fluent in_london at time T, if Y is seen by X in a place other than London.

```
terminates(E, in_london, T) :-
   perceive(T, during(E),experiencer(X),stimulus(Y)),
   occur(T,event(E), theme(Y), location(Loc)),
   not is_in(Loc,london),
   not ab_terminates(E,in_london,T).
```

However, the claim that John is not in London may be defeated due to an abnormal situation (e.g., supposed observer is a liar).

```
ab_terminates(E,in_london,T) :- person(X), person(Y), observer(E,X),
                        theme(E,Y), not speaks(X, truth, Y).
```

Next we define the observer w.r.t. the VerbNet verb perceive, as part of our commonsense knowledge. We also define what a theme is. Note that some of these technicalities are introduced due to our attempt to automate translation of English text into ASP using VerbNet [8, 26]).

```
observer(E,X) :- person(X), perceive(_, during(E), experiencer(X), _).
theme(E,X) :- person(X), perceive(_, during(E), _, stimulus(X)).
```

The concept of speaking truth is also modeled as a rule ("witnesses usually speak the truth"). The rule below states that X will normally speak the truth about observing Y, unless X is a liar (defeater).

```
speaks(X,truth,Y) :- person(X), person(Y), observer(E,X),
                     theme(E,Y), not ab_speaks(X,truth,Y).
ab_speaks(X,truth,Y) :- may_lie(X,Y).
```

We have to model the situations in which a person may lie. We assume that a person lies if we fail to prove that he/she is a truth-teller (not -lie). We also assume that a person may lie if there is evidence of conflict of interest. Note that arguments with defeaters can be thought of as default rules with exceptions. As is obvious, we make extensive use of default rules in this example.

```
may_lie(X,Y) :- person(X), person(Y), not -lie(X,Y).
-lie(X,Y) :- person(X), person(Y), not conflict_interest(X,Y).
conflict_interest(X,Y) :- person(X), person(Y), friends(X,Y),
              crime_suspect(Y), not ab_conflict_interest(X).
crime_suspect(X) :- person(X), robbery_suspect(X),
              not ab_crime_suspect(X).
```

Finally, we represent our assumptions as *abducibles*. These abducibles are simply declared using the #abducible declaration in our s(CASP) system. The assumptions we may make are the following: (i) Jan and John are friends, (ii) John is a robbery suspect, and (iii) Mary is old and infirm.

```
#abducible friends(jan, john).
#abducible robbery_suspect(john).
#abducible age(mary,old).
```

With the above arguments, evidence, assumptions, and defeaters expressed in ASP, we are ready to verify our claims. We can make a number of claims: John is in London, John is not in London, and John's location is unknown. For simplicity, we put these claims down as rules.

Claim #1: John's location is unknown.

```
claim(john_location_unknown) :- not holds(in_london, morning),
                                not -holds(in_london, morning).
```

Claim #2: John is not in London.

```
claim(john_not_in_london) :- -holds(in_london, morning).
```

Claim #3: John is in London.

```
claim(john_in_london) :- holds(in_london, morning).
```

Now we can execute the query:

```
?- claim(X).
```

to find the answers under various assumptions. These answers—five of them—computed by our s(CASP) system along with the assumptions under which the claims hold are shown next.

There is one scenario (one world, in ASP parlance) in which John's location is unknown:

```
X = john_location_unknown
Assumptions:  age(mary,old)),friends(jan,john),robbery_suspect(john))
```

The answer above states that if Mary is infirm, Jan and John are friends, and John is a robbery suspect, then we cannot really say with certainty if John is in London or not in London.

There are two scenarios in which we can support the claim that John is not in London.

```
X = john_not_in_london;
Assumptions:  age(mary,old)), not friends(jan, john)

X = john_not_in_london;
Assumptions:  age(mary,old), friends(jan,john),
              not robbery_suspect(john)
```

The claim that John is not in London can only be true if we cannot trust Mary, so the assumption about Mary being old has to hold. In the first case, if Jan and John are not friends then regardless of whether John is a robbery suspect or not, we can trust Jan to tell the truth, so John must be in Amsterdam. In the second case, Jan and John are friends, but under the assumption that John is not a robbery suspect we can trust Jan to tell the truth as well. So John must be in Amsterdam in this case as well.

Finally, there are two scenarios in which the claim that John is in London will hold.

```
X = john_in_london:
Assumptions:  friends(jan,john), robbery_suspect(john)

X = john_in_london
Assumptions:  age(mary, B | B ≠ old)), friends(jan,john)),
              robbery_suspect(john)
```

In the first case, the claim is true if Jan and John are friends and John is a robbery suspect. Mary's being young or old does not matter as Jan has a strong motivation to lie. In the second case, the claim that John is in London is obviously true if Mary is not old (and so is not infirm and her senses can be trusted), Jan and John are friends, and John is a robbery suspect.

The example above illustrates the power of ASP and of our s(CASP) system in modeling commonsense reasoning and how they can be used to automatically verify claims in the OAP framework. The mapping that we have developed between OAP and ASP facilitates this task.

6 Conclusion

We showed that ASP can elegantly model human-style arguments as laid out in the Primer of Holloway and Wasson. It is believed that human discourse can be reasoned about only through informal reasoning. We advance the argument that human discourse can be reasoned about through formal reasoning as well. Answer set programming has a well-defined declarative and operational semantics [16, 29, 4] that can model the human thought process very effectively, as demonstrated in this paper. Additionally, query-driven answer set programming can be extended with constraints over reals, which allows for reasoning over time to be performed faithfully (i.e., without discretizing time [4, 5]) as well.

References

[1] Peter Aczel (1988): *Non-well-founded sets*. *CSLI lecture notes series* 14, CSLI.

[2] Krzysztof R. Apt & Roland N. Bol (1994): *Logic Programming and Negation: A Survey*. *J. Log. Program.* 19/20, pp. 9–71.

[3] Joaquín Arias, Manuel Carro, Zhuo Chen & Gopal Gupta (2020): *Justifications for Goal-Directed Constraint Answer Set Programming*. In: *Proceedings 36th ICLP (Technical Communications)*, *EPTCS* 325, pp. 59–72. ArXiv:2009.09158.

[4] Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple & Gopal Gupta (2018): *Constraint answer set programming without grounding*. *TPLP* 18(3-4), pp. 337–354.

[5] Joaquín Arias, Zhuo Chen, Manuel Carro & Gopal Gupta (2019): *Modeling and Reasoning in Event Calculus Using Goal-Directed Constraint Answer Set Programming*. In: *Proc. LOPSTR 2019, Porto, Portugal*, *LNCS* 12042, Springer, pp. 139–155.

[6] John Barwise & Lawrence A. Moss (1996): *Vicious Circles*. Cambridge University Press.

[7] Kinjal Basu, Farhad Shakerin & Gopal Gupta (2020): *AQuA: ASP-Based Visual Question Answering*. In: *Practical Aspects of Declarative Languages*, Springer International Publishing, Cham, pp. 57–72.

[8] Kinjal Basu, Sarat Chandra Varanasi, Farhad Shakerin & Gopal Gupta (2020): *SQuARE: Semantics-based Question Answering and Reasoning Engine*. In: *Proceedings 36th International Conference on Logic Programming (Technical Communications), Rende, Italy*, *EPTCS* 325, pp. 73–86.

[9] Philippe Besnard, Claudette Cayrol & Marie-Christine Lagasquie-Schiex: *Logical Theories and Abstract Argumentation: A Survey of Existing Works*. *Argumentation and Computation* vol. 11, no. 1-2, pp. 41-102, 2020.

[10] Susanne Bobzien (2020): *Ancient Logic*. In Edward N. Zalta, editor: *The Stanford Encyclopedia of Philosophy*, summer 2020 edition, Metaphysics Research Lab, Stanford University.

[11] George Boole (1854): *An Investigation of the Laws of Thought*. Walton & Maberly.

[12] Zhuo Chen, Kyle Marple, Elmer Salazar, Gopal Gupta & Lakshman Tamil (2016): *A Physician Advisory System for Chronic Heart Failure management based on knowledge patterns*. *Theory Pract. Log. Program.* 16(5-6), pp. 604–618.

[13] Zhuo Chen, Elmer Salzar, Kyle Marple, Sandeep R. Das, Alpesh Amin, Daniel Cheeran, Lakshman Tamil & Gopal Gupta (2018): *An AI-Based Heart Failure Treatment Adviser System*. *IEEE journal of translational engineering in health and medicine* 6, 2800810.

[14] Gottlob Frege (1884): *Grundlagen der Arithmetik*. Wilhelm Koebner.

[15] Michael Gelfond & Yulia Kahl (2014): *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press.

[16] Michael Gelfond & Vladimir Lifschitz (1988): *The stable model semantics for logic programming.* In: *ICLP/SLP*, 88, pp. 1070–1080.

[17] Brendan Gillon (2016): *Logic in Classical Indian Philosophy*. In Edward N. Zalta, editor: *The Stanford Encyclopedia of Philosophy*, fall 2016 edition, Metaphysics Research Lab, Stanford University.

[18] Leo Groarke (2020): *Informal Logic*. In Edward N. Zalta, editor: *The Stanford Encyclopedia of Philosophy*, spring 2020 edition, Metaphysics Research Lab, Stanford University.

[19] Gopal Gupta (July 7, 2020): *Automating Common Sense Reasoning*. Available at `http://utdallas.edu/~gupta/csg`. Tutorial talk.

[20] Gopal Gupta, Ajay Bansal, Richard Min, Luke Simon & Ajay Mallya (2007): *Coinductive Logic Programming and Its Applications*. In: *Proc. 23rd ICLP 2007*, *Lecture Notes in Computer Science* 4670, Springer, pp. 27–44.

[21] Brendan Hall, Sarat Chandra Varanasi et al. (2020): *Knowledge-Assisted Reasoning of CLEAR Requirements with Event Calculus and Goal-Directed Answer Set Programming*. Paper in preparation.

[22] G. H. Harman (1965): *The Inference to the Best Explanation*. *The Philosophical Review* 74(1), pp. 88–95.

[23] C. Michael Holloway (2019): *Understanding the Overarching Properties*. Available at `https://ntrs.nasa.gov/citations/20190029284`.

[24] C. Michael Holloway & Kimberly S. Wasson (2020): *A Primer on Argument*. Available at `https://shemesh.larc.nasa.gov/people/cmh/cmhpubs.html`.

[25] Philip Johnson-Laird (2006): *How We Reason*. Oxford University Press.

[26] Karin Kipper, Anna Korhonen, Neville Ryant & Martha Palmer (2008): *A large-scale classification of English verbs*. *Language Resources and Evaluation* 42(1), pp. 21–40.

[27] Saul Kripke: *An Outline of a Theory of Truth*. *The Journal of Philosophy* Col. LXXII, No. 19, Nov. 6, 1975, pp. 690-716.

[28] Vladimir Lifschitz (2019): *Answer Set Programming*. Springer.

[29] Kyle Marple, Elmer Salazar & Gopal Gupta (2017): *Computing stable models of normal logic programs without grounding*. *arXiv preprint arXiv:1709.00501*.

[30] Sanjay Modgil & Henry Prakken (2014): *The* ASPIC$^+$ *framework for structured argumentation: a tutorial*. *Argument Comput.* 5(1), pp. 31–62.

[31] Farhad Shakerin, Elmer Salazar & Gopal Gupta (2017): *A new algorithm to automate inductive learning of default theories*. *Theory Pract. Log. Program.* 17(5-6), pp. 1010–1026.

[32] Murray Shanahan (1999): *The Event Calculus Explained*. In Michael J. Wooldridge & Manuela M. Veloso, editors: *Artificial Intelligence Today: Recent Trends and Developments*, *Lecture Notes in Computer Science* 1600, Springer, pp. 409–430.

[33] Luke Simon, Ajay Mallya, Ajay Bansal & Gopal Gupta (2006): *Coinductive Logic Programming*. In: *Proc. ICLP'06*, *Lecture Notes in Computer Science* 4079, Springer, pp. 330–345.

[34] Alfred Tarski (1939): *On Undecidable Statements in Enlarged Systems of Logic and the Concept of Truth*. *J. Symb. Log.* 4(3), pp. 105–112.

[35] Wikipedia (retrieved February, 2021): *Inductive Reasoning*. `https://en.wikipedia.org/wiki/Inductive_reasoning`.

[36] Clyde W. Yancey, Mariell Jessup et al. (2013): *ACCF/AHA Guideline for the Management of Heart Failure*. *Circulation* 28(16), pp. e240–e327.