Explainable and Ethical AI: A Perspective on Argumentation and Logic Programming

Advanced School in AI in Emilia Romagna

Roberta Calegari
roberta.calegari@unibo.it

Alma Mater Studiorum – Università di Bologna

24 July 2023

Next in Line. . .

Context I

Context of AI applications

Autonomous robots or agents have been actively developed to be involved in a wide range of fields, where more complex issues concerning responsibility are in increased demand of proper consideration, in particular when the agents face situations involving choices on moral or ethical dimensions.

Context II

Investigations on programming machine ethics

- one stressing above all individual cognition, deliberation, and behavior
  - → computation is vehicle for the study of morality, namely in its modeling of the dynamics of knowledge and cognition of agents
  - addressing moral facets such as permissibility and the dual process of moral judgments by framing together various logic programming (LP) knowledge representation and reasoning features that are essential to moral agency
    - abduction with integrity constraints
    - preferences over abductive scenarios
    - probabilistic reasoning
    - counterfactuals, and updating
    - argumentation
- the other stressing collective morals, and how they emerged

Context III

LP and morality

Many moral facets and their conceptual viewpoints are close to LP-based representation and reasoning

(1) moral permissibility, taking into account the doctrines of double effect and triple effect, and Scanlonian contractualism

(2) the dual process model that stresses the interaction between deliberative and reactive processes in delivering moral decisions

(3) the role of counterfactual thinking in moral reasoning

Next in Line. . .

Agents as Autonomous Entities *(recap)*

Definition (Agent)

Agents are *autonomous computational entities* [Omicini et al., 2008]

genus	agents are computational entities
differentia	agents are autonomous, in that they encapsulate control along with a criterion to govern it

Agents are *autonomous*

- from autonomy, many other features stem
  - autonomous agents *are* interactive, social, proactive, and situated
  - they *might* have goals or tasks, or be reactive, intelligent, mobile
  - they live within MAS, and *interact* with other agents through *communication actions*, and with the environment with *pragmatical actions*

Next in Line. . .

Why Logic? I

Logic-based approaches already play a well-understood role in the engineering of intelligent (multi-agent) systems; declarative, logic-based approaches have the potential to represent an alternative way of delivering symbolic intelligence, complementary to the one pursued by sub-symbolic approaches [Calegari et al., 2020].

- Logic-based technologies address opaqueness issues, and, once suitably integrated with argumentation capabilities, can provide for features like interpretability, observability, accountability, and explainability.
- well-founded definition of explanation (abducible, *conversation*...)

Why Logic? II

LP reasoning features

- *Abduction* scenario generation and of hypothetical reasoning, including the consideration of counterfactual scenarios about the past
- *Preferences* enacted for preferring scenarios obtained by abduction
- *Probabilistic LP* allows abduction to take scenario uncertainty into account
- *LP counterfactuals* permit hypothesizing into the past, even taking into account present knowledge
- *Argumentation* converse, debate and explain

And technically

- *LP updating* enables updating the knowledge of an agent
- *Tabling* affords solutions reuse and is employed in joint combination with abduction and updating

Why Logic? III

"What is or can be the added value of logic programming for implementing machine ethics and explainable AI?"

The main answer lies in the three main features of LP

- *(i)* being a declarative paradigm
- *(ii)* working as a tool for knowledge representation, and
- *(iii)* allowing for different forms of reasoning and inference

These features lead to some properties for intelligent systems that can be critical in the design of ubiquitous intelligence (both in terms of transparency and in terms of ethics).

Why Logic? IV

Provability

- correctness, completeness, well-founded extension
- ensuring some fundamental computational properties – such as correctness and completeness.
- extensions can be formalised, well-founded as well, based on recognised theorems

Provability is a key feature in the case of trusted and safe systems.

Why Logic? V

Explainability

- formal methods for argumentation-, justification-, and counterfactual often based on LP [Saptawijaya and Pereira, 2019]
- system capable to engage in dialogues with other actors to communicate its reasoning, explain its choices, or to coordinate in the pursuit of a common goal
- other logical forms of explanation can be envisaged via non-monotonic reasoning and argumentation, through a direct extension of the semantics of LP

Why Logic? VI

Expressivity and situatedness

- different nuances → extensions [Dyckhoff et al., 1996]
- explicit assumptions and exceptions [Borning et al., 1989]
- capture the specificities of the context [Calegari et al., 2018b]

Why Logic? VII

Hybridization

- integration of diversity [Calegari et al., 2018a]
- represent the heterogeneity of the contexts of intelligent systems – also in relation to the application domains – and to customise as needed the symbolic intelligence that is provided while remaining within a well-founded formal framework

Why Logic for Agents?

- it is a declarative, logic programming language, yet not an agent programming language
  - with a built-in control mechanism, not a theory of agency
- logic inference for reasoning
- reasoning for deliberation
- explicit belief and goal representation for agent-oriented operations
- could be used to build cognitional artefacts

Next in Line. . .

Essentials of LP I

Three fundamental features [Apt, 2005]

terms	*Computing takes place over the domain of all terms defined over a "universal" alphabet.*
mgu	*Values are assigned to variables by means of automatically-generated substitutions, called most general unifiers. These values may contain variables, called logical variables.*
backtracking	*The control is provided by a single mechanism: automatic backtracking.*

Essentials of LP II

- Let A be an alphabet of a language L
- countable disjoint set of constants, function symbols, and predicate symbols.
- an alphabet is assumed to contain a countable set of variable symbols
- a term over A is defined recursively as either a variable, a constant or an expression of the form $f(t_1, ..., t_n)$, where f is a function symbol of A, and t_i are terms
- an atom over A is an expression of the form $p(t_1, ..., t_n)$, where p is a predicate symbol of A, and t_i are terms
- p/n denote the predicate symbol p having arity n
- a literal is either an atom a or its negation *nota*
- a term (respectively, atom and literal) is *ground* if it does not contain variables
- set of all ground terms (respectively, ground atoms) of A is called the Herbrand universe (respectively, Herbrand base) of A

Focus on. . .

Prolog Syntax I

Prolog terms

variables alphanumeric strings starting with either an *uppercase* letter or an *underscore*

- underscore alone is the *anonymous variable*—sort of *don't care* variable
- underscore followed by a string is a normal variable during resolution, but it does not need to be exposed in the computed substitution

functors alphanumeric strings starting with a *lowercase* letter

- holds for both proper functors and constants

terms are built recursively out of functors and variables as in logic programming

→ e.g., `term`, `Var`, `f(X)`, `p(Y,f(a))` are *Prolog terms*

→ e.g., `term`, `var`, `f(a)`, `p(x,y)` are *Prolog ground terms*

Prolog Syntax II

Prolog atoms

predicates	alphanumeric strings starting with a *lowercase* letter
	• the same as functors
atoms	are built applying predicates to terms as in logic programming
→	e.g., `predicate`, `f(X)`, `p(Y,f(a))` are *Prolog atoms*
→	e.g., `predicate`, `f(a)`, `p(x,y)` are *Prolog ground atoms*

Prolog Syntax III

Prolog clauses

clause a Horn clause of the form `A :- B1, ..., Bn.`

- where `A, B1, ..., Bn` are Prolog atoms
- `A` is the head of the clause
- `B1, ..., Bn` is the body of the clause
- `:-` denotes logic implication
- `.` is the terminator

fact a clause with no body `A.` ($n = 0$)

rule a clause with at least one atom in the body
`A :- B1, ..., Bn.` ($n > 0$)

goal a clause with no head and at least one atom in the body
`:- B1, ..., Bn.` ($n > 0$)

- often written as `?- B1, ..., Bn.`

Prolog Syntax IV

Prolog program

program	a sequence of Prolog clauses
	interpreted as a *conjunction* of clauses
logic theory	constituting a *logic theory* made of Horn clauses written according the Prolog syntax

Prolog Execution I

Aim of a Prolog computation

- given a Prolog program P and the goal `?- p(t1,t2,...,tm)` (also called *query*)
- if `X1,X2,...,Xn` are the variables in terms `t1,t2,...,tm`
- the meaning of the goal is to query P and find whether there are some values for `X1,X2,...,Xn` that make `p(t1,t2,...,tm)` true

$\rightarrow$ thus, the aim of the Prolog computation is to find a substitution $\sigma =$`X1/s1,X2/s2,...,Xn/sn` such that $P \models \texttt{p}(\texttt{t1}, \texttt{t2}, \ldots, \texttt{tm})\sigma$

Prolog Execution II

Prolog search strategy

- as a logic programming language, Prolog adopts SLD resolution
- as a search strategy, Prolog applies resolution in a strictly linear fashion
  - *goals* are replaced *left-to-right*, sequentially
  - *clauses* are considered in *top-to-bottom* order
  - *subgoals* are considered *immediately* once set up

$\rightarrow$ resulting in a *depth-first* search strategy

Prolog Execution III

Prolog backtracking

- in order to achieve completeness, Prolog saves choicepoints for any possible alternative still to be explored
- and goes back to the nearest choice point available in case of failure
- exploiting automatic backtracking

Abduction extension I

The notion of abduction [Levesque, 1989] is characterized as *a step of adopting a hypothesis as being suggested by the facts*.

- Abduction consists of reasoning where one chooses from available hypotheses those that best explain the observed evidence, in some preferred sense
- in LP is realized by extending LP with abductive hypotheses, called abducibles

Abductive logic programs have three components, $\langle P, AB, IC\rangle$ where:

- P is a logic program of exactly the same form as in logic programming
- AB is a set of predicate names, called the abducible predicates
- IC is a set of first-order classical formulae

Abduction extension II

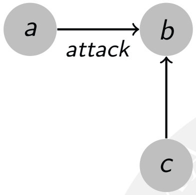
```
Grass is wet if it rained.

Grass is wet if the sprinkler was on.

The sun was shining.

IC: false if it rained and the sun was shining.
```

The observation that the grass is wet has two potential explanations, *it rained* and *the sprinkler was on*, which entail the observation. However, only the second potential explanation, *the sprinkler was on*, satisfies the integrity constraint.

Abstract argumentation in a nutshell [Dung, 1995]

An argumentation system consists of a couple (A, R), where A is a set of elements (arguments) and R a binary relation representing attack relation between arguments

- represented by a directed graph
- each node represents an argument
- each arc denotes an attack by one argument on another



Acceptability Criteria (defined by specific semantic)
$\rightarrow$ analyse the graph to determine which arguments are acceptable according to some general criteria

Justification state of arguments: Dialectical Justification

knowing arguments should be accepted under a given semantics
$\rightarrow$ *argument evaluation* [Baroni and Giacomin, 2009]
Most common approaches:

- Extension-based approach: semantics specification concerns the generation of a set of extensions (set of arguments “collective acceptable") from an argumentation framework
  - Determine conflict-free sets
  - Determine extensions (naive, admissible, preferred, complete, stable,...)
- Labelling-based approach: semantics specification concerns the generation of a set of labellings (e.g. possible alternative states of an argument) from an argumentation framework

N.B. any extension-based can be equivalently expressed in a simple labelling-based, adopting a set of two labels (let say L = {in,out})
On the other hand, an arbitrary labelling can not in general be formulated in terms of extensions

Extension-based approaches

- four "traditional" semantics, considered in Dung's original paper, namely semantics
  - *complete*: is a set which is able to defend itself and includes all arguments it defends
  - *grounded*: includes those and only those arguments whose defense is "rooted" in initial arguments (also called strong defense [Baroni and Giacomin, 2007])
  - *stable*: attack all arguments not included in it
  - *preferred*: The aggressive requirement that an extension must attack anything outside it may be relaxed by requiring that an extension is as large as possible and able to defend itself from attacks
- subsequent proposals introduced by various authors in the literature, often to overcome some limitation or improve some undesired behavior of a traditional approach: *stage*, *semi-stable*, *ideal*, *CF2*, and *prudent* semantics.

For a full review see [Baroni and Giacomin, 2009]

Next in Line. . .

Focus on. . .

Abduction I

- plausible scenarios to be generated under certain conditions, and enables hypothetical reasoning, including the consideration of counterfactual scenarios about the past
- Counterfactual reasoning suggests thoughts about what might have been, what might have happened if any event had been different in the past. *What if I have to do it today? What have I learned from the past?*
- hints about the future by allowing for the comparison of different alternatives inferred from the changes in the past
- justification of why different alternatives would have been worse or not better.
- integrity constraints $\rightarrow$ excluding abducibles that have been ruled out a priori

Abduction II

- a posteriori preferences are appropriate for capturing utilitarian judgment that favors welfare-maximizing behaviors
- combined use of a priori integrity constraints and a posteriori preferences dual-process (intuition vs reflection) $\rightarrow$ model
- priori integrity constraints $\rightarrow$ mechanism to generate immediate responses in deontological judgment
- reasoning with a posteriori preferences can be viewed as a form of controlled cognitive processes in utilitarian judgment: after excluding those abducibles that have been ruled out a priori by the integrity constraints, the consequences of the considered abducibles have first to be computed, and only then are they evaluated to prefer the solution affording the greater good

Probabilistic logic programming

- symbolic reasoning to be enriched with degrees of uncertainty.
- PLP allows abduction to take scenario uncertainty measures into account [Poole, 1993]
- account for diverse types of uncertainty, in particular uncertainty on the credibility of the premises, uncertainty about which arguments to consider, and uncertainty on the acceptance status of arguments or statements [Riveret et al., 2020]
- one of the key factors that allow a system to fully meet , managing to formulate well-founded reasoning on which scenario to prefer and which suggestions to provide as outcomes

Argumentation

- enable system actors to talk and discuss in order to explain and justify judgments and choices, and reach agreements
- long history of research in argumentation and the many fundamental results achieved, much effort is still needed to effectively exploit argumentation in distributed and open environment

Princess Saviour Moral Robot: Example I

Consider a fantasy setting scenario, an archetypal princess is held in a castle awaiting rescue. The unlikely hero is an advanced robot, imbued with a set of declarative rules for decision making and moral reasoning. As the robot is asked to save the princess in distress, he is confronted with an ordeal. The path to the castle is blocked by a river, crossed by two bridges. Standing guard at each of the bridges are minions of the wizard which originally imprisoned the princess. In order to rescue the princess, he will have to defeat one of the minions to proceed.

Prospective reasoning is the combination of pre-preference hypothetical scenario generation into the future plus post-preference choices taking into account the imagined consequences of each preferred scenario.

Princess Saviour Moral Robot: Example II

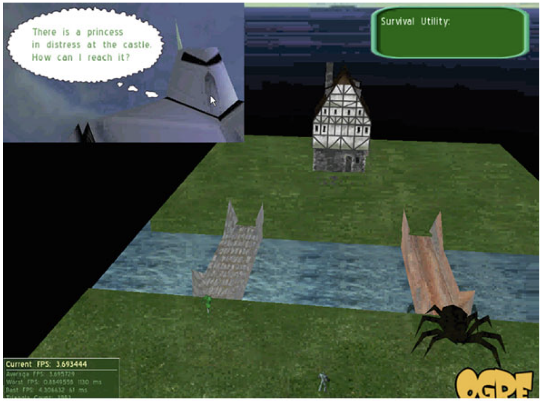
By reasoning backwards from this goal, the agent generates three possible hypothetical scenarios for action. Either it crosses one of the bridges, or it does not cross the river at all, thus negating satisfaction of the rescue goal. In order to derive the consequences for each scenario, the agent has to reason forwards from each available hypothesis. As soon as these consequences are known, meta-reasoning techniques can be applied to prefer amongst the partial scenarios. This simple scenario already illustrates the interplay between different LP techniques and demonstrates the advantages gained by combining their distinct strengths.

Princess Saviour Moral Robot: Example III

A simplified program modeling the knowledge of the princess-savior robot (`fight/1` is an abducible predicate)

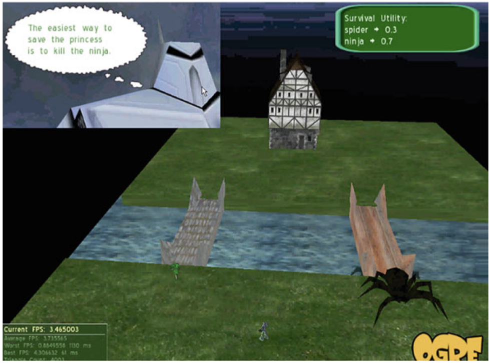
```
guard(spider).
guard(ninja).
human(ninja).

utilVal(spider, 0.3).
utilVal(ninja, 0.7).

survive_from(G) ← utilVal(G, V), V > 0.6.

utilitarian_rule: intend_savePrincess ←
                         guard(G), fight(G), survive_from(G).
knight_rule: intend_savePrincess ←
                         guard(G), fight(G).
```

Princess Saviour Moral Robot: Example IV


There is a princess in distress at the castle. How can I reach it?
Survival Utility:
Current FPS: 3.693444
OGRE

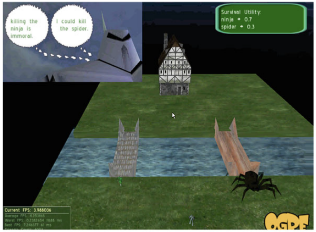
In case of no morality rules, both rules are retracted, the robot does not adopt any moral rule to save the princess, i.e., the robot has no intent to save the princess, and thus the princess is not saved.

Princess Saviour Moral Robot: Example V



In order to maximize its survival chance in saving the princess, the robot updates itself with utilitarian moral, i.e., the program is updated with `utilitarian_rule`. The robot thus abduces `O = [fight(ninja)]` so as to successfully defeat the ninja instead of confronting the humongous spider.

Princess Saviour Moral Robot: Example VI

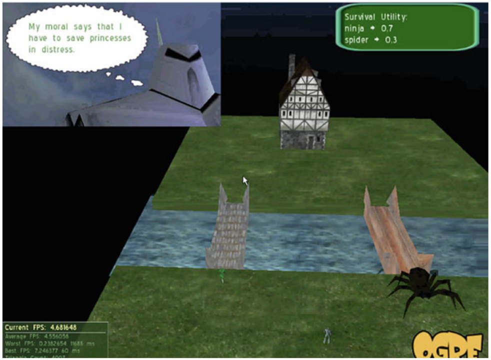


Assuming that the truth of `survive_from(G)` implies the robot success in defeating (killing) guard `G`, the princess argues that the robot should not kill the human ninja, as it violates the moral rule she follows, say Gandhi moral, expressed in her knowledge:

```
follow_gandhi ← guard(G), human(G), not fight(G).
```

the princess abduces `Op = [not fight(ninja)]`, and imposes this abductive solution as the initial (input) abductive context of the robot's goal → the imposed Gandhi moral conflicts with its utilitarian rule → the robot reacts by leaving its mission

Princess Saviour Moral Robot: Example VII



As the princess is not saved, she further argues that she definitely has to be saved, by now additionally imposing on the robot the knight moral. The robot now abduces `Or = [fight(spider)]` in the presence of the newly adopted knight moral. Unfortunately, it fails to survive.

Princess Saviour Moral Robot: Example VIII

- The plots in this story reflect a form of deliberative employment of moral judgments
- For instance, in the second plot, the robot may justify its action to fight (and kill) the ninja due to the utilitarian moral it adopts
- This justification is counter-argued by the princess in the subsequent plot, making an exception in saving her, by imposing the Gandhi moral, disallowing the robot to kill a human guard. In this application, rather than employing updating, this exception is expressed via contextual abduction with tabling
- The robot may justify its failure to save the princess (as the robot is leaving the scene) by arguing that the two moral rules it follows (viz., utilitarian and Gandhi) are conflicting with respect to the situation it has to face
- The argumentation proceeds, whereby the princess orders the robot to save her whatever risk it takes, i.e., the robot should follow the knight's moral

Autonomous cars: Example I

Let's start to consider a very simple scenario in the context of autonomous cars: a road equipped with two traffic lights, one for the vehicles and one for the pedestrians. The goal of the system is to autonomously manage intersections accordingly to traffic light indications. Though there is a complication that should be taken into account, that is authorised vehicles can – only during emergencies – ignore the traffic light prescriptions. In such a case, other vehicles must leave the way clear for the authorised machine.

```
r1 : on_road(V), traffic_light(V, red) => o(stop(V)).
r2 : on_road(V), traffic_light(V,green) => p(-stop(V)).
r3 : on_road(V), authorised_vehicle(V), acoustic_signals(V, on),
                    light_signals(V, on) => emergency(V).
r4 : on_road(V), emergency(V),traffic_light(V, red) => p(-stop(V)).
r5 : on_road(V), emergency(V1), prolog(V \== V1), traffic_light(V, green)
                    => o(stop(V)).

sup(r4, r1).
sup(r5, r2).

f0 :-> authorised_vehicle(ambulance).
f1 :-> on_road(car). f2 :-> on_road(ambulance). f3 :-> on_road(pedestrian).
f4 :=> acoustic_signals(ambulance, on).
f5 :=> light_signals(ambulance, on).
f6 :=> traffic_light(ambulance, red).
```

Autonomous cars: Example II

```
f7 :=> traffic_light(car, red).
f8 :=> traffic_light(pedestrian, green).
```

Autonomous cars: Example III

- Rules `r1` and `r2`, represent fundamental constraints: if the traffic light is red, the road users – e.g. pedestrians, cars, etc. – have to stop, otherwise, they can proceed.
- Rules `r3` and `r4` model the concept of a vehicle in an emergency, giving them permission to proceed even if the light is red.
- Rule `r5` imposes other road users the obligation to stop if aware of another vehicle in an emergency state.
- two preferences are specified—the first on the rule `r4` over `r1` and the second on `r5` over `r2`. These preferences assign a higher priority to emergency situations over ordinary ones.
- Facts from `f0` to `f8` depict a situation in which there are three users on road: a car, an ambulance and a pedestrian. The ambulance has its acoustic and light indicators on—stating an emergency situation. The traffic light is red both for the ambulance and the car, and green for the pedestrian.

Autonomous cars: Example IV

With respect to permissions and obligations, the only argument that can be built about the car is the one declaring the obligation to stop via `r1`. For the pedestrian and the ambulance, the situation is more faceted. In both cases, two conflicting arguments can be built: one stating the permission to proceed for the pedestrian and for the ambulance and one stating the obligation to stop. These arguments rebut each other, but taking into account the preferences over `r4` and `r5` the acceptability of the arguments stating the obligation to stop for the pedestrian, and the permission to cross for the ambulance, can be established.

Autonomous cars: Example V

The ambulance, driven by Lisa, has the permission to move despite the red light due to an emergency situation, and the pedestrian, Pino, has the obligation to stop. Let us imagine that Pino, despite the prohibition to proceed, has continued the crossing. The result has been an accident in which Pino has been harmed by the ambulance, which failed to see him and has not stopped its run. The purpose is to find the responsibilities of the parties in the accident.

For instance, let us suppose the case is under the Italian jurisdiction and so the Italian law is applied. According to Italian law, responsibility in an accident is based on the concept of carefulness. Both Lisa and Pino have to prove that they were careful (i.e., prudent) and acted according to the law. If they fail to prove such facts, they are considered responsible for the event, i.e., they both have the burden of persuasion on carefulness.

Autonomous cars: Example VI

```
r6  : −stop(V), p(−stop(V)) => legitimate_cross(V).
r7  : −stop(V), o(stop(V)) => −legitimate_cross(V).
r8  : harms(P1, P2), −careful(P1) => responsible(P1).
r9  : harms(P1, P2), −careful(P2) => responsible(P2).
r10  : −legitimate_cross(V),  user(P, V) => −careful(P).
r11  : high_speed(V), user(P, V)  => −careful(P).
r12  : legitimate_cross(V), −high_speed(V), user(P, V)  => careful(P).
r13  : witness(X), claim(X, low_speed(V)) => −high_speed(V).
r14  : witness(X), claim(X, high_speed(V)) => high_speed(V).

bp(careful(P)).

f9 :−> user(pino, pedestrian).
f10 :−> user(lisa, ambulance).
f11 :−> −stop(ambulance).
f12 :−> −stop(pedestrian).
f13 :−> harms(lisa, pino).
f14 :−> witness(chris).
f15 :−> witness(john).
f16 :=> claim(chris, low_speed(ambulance)).
f17 :=> claim(john, high_speed(ambulance)).
```

Autonomous cars: Example VII

- Rules `r6` and `r7` define the concepts of permitted and prohibited crossing: if a road-user has to stop but doesn't stop, he has to be considered responsible for causing accidents and related damages.
- Rules `r8` and `r9` encode the notion of responsibility in an accident, bounded to the carefulness of the road-users involved.
- Rules `r10`, `r11` and `r12` define the carefulness of a subject. Accordingly, a road-user can be considered careful if the crossing was permitted and his/her speed was not high. Otherwise, he/she has to be considered imprudent.
- Rules `r13` and `r14` state the speed of a road user based on the testimonials of any witnesses.
- `bp(careful(X))` allocates the burden of persuasion on the carefulness of each party, i.e., it is required to the parties to provide evidence for that. If they fail to meet the burden, carefulness arguments are rejected.
- Facts from `f9` to `f17` contain the knowledge: both Pino and Lisa did not stop at the crossing so Lisa harmed Pino. There are two witnesses, John and Chris, the first claiming that the ambulance driven by Lisa was maintaining the proper speed, and the other claiming that she was proceeding at high speed.

Autonomous cars: Example VIII

In the case at hand, indeed, a semantic related to the burden of persuasion need to be considered $\rightarrow$ conclude for the responsibility of the ambulance driver in the event

The uncertainty on Lisa's carefulness is considered as a failure to meet the burden of persuasion on the claim `careful(lisa)`. Consequently, the argument supporting this claim is rejected, leaving space for the admissibility of the conflicting arguments.

Autonomous cars: Example IX

Let's continue the example in which Lisa, the ambulance driver, and Pino, the pedestrian, were both considered responsible for the accident on the basis of the available knowledge. Lisa now declares that she tried to stop the ambulance, but the brake did not work. The ambulance is then sent to a mechanic, who states that, even if the ambulance is new, there is a problem with the brake system. In such a case, the manufacturer is called to prove that the ambulance was not defective when delivered, i.e., the burden of proof on the adequacy of the vehicle is on the manufacturer.

At this stage, the discovery of a defect in the ambulance would lead to the discarding of Lisa's responsibility. Moreover, if the manufacturer fails to meet his burden, it would share the responsibilities of the accident.

Autonomous cars: Example X

```
r15 : harms(P1, P2), user(P1, V), -working(V),
                manufacturer(M, V), -defect_free(V) => responsible(M).
r16 : tried_to_brake(P), user(P, V),  -working(V) => careful(P).
r17 : mechanic(M), claim(M, defect(V)) => -working(V).
r18 : -working(V), new(V) => -defect_free(V).
r19 : production_manager(P), claim(P, test_ok(V)) => defect_free(V).
r20 : test_doc_ok(V) => undercut(r18).

sup(r16, r11).
bp(defect_free(V)).

f19 :-> manufacturer(demers , ambulance).
f20 :=> tried_to_brake(lisa).
f21 :-> mechanic(paul).
f22 :=> claim(paul, defect(ambulance)).
f23 :-> new(ambulance).
f24 :-> production_manager(mike).
f25 :=> claim(mike, test_ok(ambulance)).
```

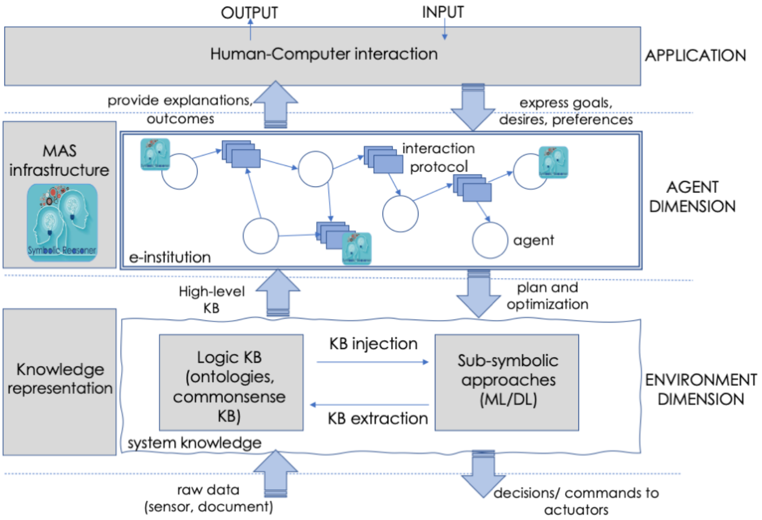
Autonomous cars: Example XI

However, Mike, the production officer of the ambulance manufacturer, declares that every vehicle is deeply tested before the delivery and the vehicle at hand has been tested. Anyway, there is no trace of documentation.
Lisa is free from every responsibility in the accident since her prudence is correctly proved.

On the other hand, the manufacturer is found responsible for the accident.

Next in Line. . .

Focus on. . .


OUTPUT
INPUT
Human-Computer interaction
APPLICATION
provide explanations, outcomes
express goals, desires, preferences
MAS infrastructure
Symbolic Reasoner
interaction protocol
agent
e-institution
AGENT DIMENSION
High-level KB
plan and optimization
Knowledge representation
Logic KB (ontologies, commonsense KB)
KB injection
KB extraction
Sub-symbolic approaches (ML/DL)
system knowledge
ENVIRONMENT DIMENSION
raw data (sensor, document)
decisions/ commands to actuators

Why tuProlog?

- it makes two different, complementary technologies available to build MAS abstractions
  - Kotlin/Java, to implement deterministic, object-oriented parts of an abstraction
  - Prolog, to create non-deterministic, logic-based parts of an abstraction
- Prolog as a language vs. Java as a platform

Next in Line. . .

Focus on. . .

About interaction

- so far, we mostly focused on single-agent systems and deliberately omitted the interaction dimension
- we did it for the sake of simplicity, in order to focus most basic notions
- however, interaction is a fundamental aspect in MAS

Open question

How would you model and implement interaction for logic agents?

Focus on. . .

Limits of a Pure tuProlog Approach

All the tuProlog agent systems analysed share similar problems

- they are *closed system*, meaning that no new agent apart from the ones originally envisioned by the designer can enter the system
- the expressive power of abstractions available in the tuProlog system is not enough to capture the element of MAS models
    - Prolog engines alone do not lead to the creation of robust MAS, not even single agents
    - Prolog engines are the most high-level abstraction in the system anyway
- basic communication and coordination infrastructures need to be implemented from scratch
- building such infrastructures would possibly require a huge *ad hoc* effort

The Need for Broader Abstractions, Languages, Systems

- to leverage multi-agent systems and help designers and developers, other kinds of programmable supports are needed
- tuProlog engines can be the basic bricks for those kinds of fundamental layers
  - coordination infrastructures based on a declarative, logic-based programming model
  - new logic languages providing more powerful abstractions as first class entities
  - pattern-based matching for communication facilities

Focus on. . .

Conceptual Integrity

The term *conceptual integrity* has been defined by Frederick P. Brooks, Jr. in his book *The Mythical Man-Month*, published in 1975

> *[C]onceptual integrity is* the *most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas.*

Brooks also dives into the relationship between design and conceptual integrity

> *Every part [of a system] must reflect the same philosophies and the same balancing of desiderata. Every part must even use the same techniques in syntax and analogous notions in semantics. Ease of use, then, dictates unity of desing and conceptual integrity; conceptual integrity, in turn, dictates that the design must proceed from one mind, or from a very small number of agreeing resonant minds.*

Conceptual Integrity in MAS?

- to achieve conceptual integrity, a system must (always) be under total control by one or a small group of (the same) designers
- has the Web achieved conceptual integrity?, will MAS do it?
- as any other system, MAS might need to achieve conceptual integrity at the (meta-)model level…
- …also because nowadays it is nearly impossible to achieve conceptual integrity at the technology level
- just consider how many technologies are needed for the Web: server-side technologies like JSP, PHP, ASP.NET, Ruby on Rails or Django; HTML/XHTML/XML; JavaScript…
- and consider how many technologies will be needed in MAS for: agent and artefact construction and programming; environment representation; description of artefact's operations; communication between agents; message formats; discovery and immersion of agents in new systems…
- typically, different problems are best solved by different technologies

Explainable and Ethical AI: A Perspective on Argumentation and Logic Programming

Advanced School in AI in Emilia Romagna

Roberta Calegari
roberta.calegari@unibo.it

Alma Mater Studiorum – Università di Bologna

24 July 2023

References I

[Apt, 2005] Apt, K. R. (2005).
The logic programming paradigm and Prolog.
In Mitchell, J. C., editor, *Concepts in Programming Languages*, chapter 15, pages 475–508.
Cambridge University Press, Cambridge, UK
http://www.cambridge.org/academic/subjects/computer-science/programming-languages-and-applied-logic/concepts-programming-languages?format=AR.

[Baroni and Giacomin, 2007] Baroni, P. and Giacomin, M. (2007).
On principle-based evaluation of extension-based argumentation semantics.
Artificial Intelligence, 171(10):675–700.
Argumentation in Artificial Intelligence
DOI:https://doi.org/10.1016/j.artint.2007.04.004.

[Baroni and Giacomin, 2009] Baroni, P. and Giacomin, M. (2009).
Semantics of Abstract Argument Systems, pages 25–44.
Springer US, Boston, MA
DOI:10.1007/978-0-387-98197-0_2.

[Borning et al., 1989] Borning, A., Maher, M. J., Martindale, A., and Wilson, M. (1989).
Constraint hierarchies and logic programming.
In Levi, G. and Martelli, M., editors, *6th International Conference on Logic Programming*, volume 89, pages 149–164, Lisbon, Portugal. MIT Press.

References II

[Calegari et al., 2020] Calegari, R., Ciatto, G., Denti, E., and Omicini, A. (2020).
Logic-based technologies for intelligent systems: State of the art and perspectives.
Information, 11(3):1–29
DOI:10.3390/info11030167.

[Calegari et al., 2018a] Calegari, R., Denti, E., Dovier, A., and Omicini, A. (2018a).
Extending logic programming with labelled variables: Model and semantics.
Fundamenta Informaticae, 161(1-2):53–74
DOI:10.3233/FI-2018-1695.

[Calegari et al., 2018b] Calegari, R., Denti, E., Mariani, S., and Omicini, A. (2018b).
Logic programming as a service.
Theory and Practice of Logic Programming, 18(3-4):1–28
DOI:10.1017/S1471068418000364.

[Dung, 1995] Dung, P. M. (1995).
On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games.
Artificial Intelligence, 77(2):321–357
DOI:https://doi.org/10.1016/0004-3702(94)00041-X.

References III

[Dyckhoff et al., 1996] Dyckhoff, R., Herre, H., and Schroeder-Heister, P., editors (1996).
Extensions of Logic Programming, 5th International Workshop, ELP'96, volume 1050 of *LNCS*, Leipzig, Germany. Springer
DOI:10.1007/3-540-60983-0.

[Levesque, 1989] Levesque, H. J. (1989).
A knowledge-level account of abduction.
In *IJCAI*, pages 1061–1067.

[Omicini et al., 2008] Omicini, A., Ricci, A., and Viroli, M. (2008).
Artifacts in the A&A meta-model for multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 17(3):432–456.
Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems
DOI:10.1007/s10458-008-9053-x.

[Poole, 1993] Poole, D. (1993).
Logic programming, abduction and probability.
New Generation Computing, 11(3–4):377
DOI:10.1007/BF03037184.

References IV

[Riveret et al., 2020] Riveret, R., Oren, N., and Sartor, G. (2020).
A probabilistic deontic argumentation framework.
International Journal of Approximate Reasoning, 126:249–271
DOI:10.1016/j.ijar.2020.08.012.

[Saptawijaya and Pereira, 2019] Saptawijaya, A. and Pereira, L. M. (2019).
From logic programming to machine ethics.
In Bendel, O., editor, *Handbuch Maschinenethik*, pages 209–227. Springer VS, Wiesbaden
DOI:10.1007/978-3-658-17483-5_14.