Imperial College London

Department of Computing

UROP

Intention Recognition - Computational Morality

Author:
Ioana Daniela Radu

Supervisor:
Dr. Fariba Sadri

July 31, 2014

Contents

1 Introduction

We implemented a Java computational model to assign responsibility to agents participating in a certain given scenario that was transformed into an LPS. We used previous research done by Mao and Gratch[1] and concepts from Attribution Theory[2] to design an algorithm that would take the events of a real life situation and compute the blame for a given outcome to each individual participating in said situation.

2 Situation Modelling

2.1 LPS

An LPS, or Logic-based Production System, developed by Prof.F.Sadri and Prof.R.Kowalski, is a system that uses both logic programming and reactive programming to model a changing environment that behaves according to some rules. I used an LPS language developed by Inhyeok Kim for his final year project at Imperial College London[3]. For details of how an LPS is structured refer to his report on how he organised his language.

We are interested in the history that Kim's project provides after running the LPS through his program. This tool is very helpful because it gives us each state in our respective scenario. This history is in the form of a JSON file, which Java can, with the help of a special library called GSON, parse back to Java Objects. We use the State Class that Kim wrote for his program to get the states of our situation.

2.2 State manipulation

After we get the states, we need only some specific information from each one of them such as : the facts, the observed events and the executed actions. Thus, we parse the states to only acquire that information and create a strip downed new states.

2.3 Action Effect Conditions Database

We know that specific actions have certain effects under some conditions. Therefore, we require a database with all the actions and conditions and effects, in a general, unbound version. We give this in a form of a text file, that will be parsed by our program to create a list of what we call AECs. The reason for this list is so that we can backtrack each outcome from this database to the states and discover the chain of events that led to a certain outcome. This list contains both the LPS' Reactive Rules and its Domain Theory.

[1]Modelling Social Causality and Responsibility `http://ijcai.org/papers13/Papers/IJCAI13-479.pdf`
[2]Evaluating a Computational Model of Social Causality and Responsibility`http://tinyurl.com/kcrn9xf`
[3]GitRepository : `https://github.com/elijah6/LPSWorkflow`

2.4 Backtrack

To discover the chain of events that produced the result, we must use the AEC database in order to see which action has as effect the result. After we discover this action, we must bind the variables of the AEC to the ones contained in the searched outcome. After binding the variables of the outcome, we go through all of the executed actions and observed events of our states and try to find a match. When we find one, we check if the bindings are correct. An example would be if we were looking for do(throwSwitch) and had something like this :

Bound AEC :

$$orders(X, throwSwitch)\ \&\ head(X) \rightarrow do(throwSwitch)$$

And we found a state which contained

In the observed events: *orders(tom, throwSwitch)*

In the facts: *head(tom)*

We want to rebind the variables and check if after the rebinding both the rebound action and conditions are found in the state. That is how we know that we have found the next link in our chain of events.

3 Agents' knowledge

After we get our list of completely bound AECs, that is our story for what happened to get to a certain outcome we have to get the agents that participate in the scenario: their name, rank and knowledge. We built a myAgent class that constructs our agent. We then give an input of the agents through the console. For each agent we provide their name, rank and knowledge. After this we can parse the information given and make a list of agents. The first agent we give as input is our observed agent and the rest are auxiliary agents who interact with the observed agent or with the environment.

The knowledge of each agent is paramount, because this will play a significant role in determining the distribution of responsibility. It is important factor in deciding an agent intention to do a certain action or if they were coerced or not into taking the action. Modifying an agent's knowledge could therefore impact all of the agents' responsibility.

4 Assigning Responsibility

The factors we took into consideration when we decide how to assign blame for an outcome are physical causality, intention and coercion. Of course, the program could be extended to support other factors, such as : forseeability or justification. To learn more about Attribution Theory and how humans process blame, we also used the Topics Project that was on the same topic[4].

[4] `http://www.doc.ic.ac.uk/project/2013/163/g1316329/web/morality.html`

4.1 Physical Causality

Physical causality explains the simplest relationship between the action and the effect. In other words, it assigns all the responsibility to the agent that performed the closest action to the outcome. Hence, our program takes the first action in our list of chained events and links it to an agent, who will be assigned a responsibility number[5].

4.2 Intention

It is clear that if an agent has intention then they are aiming for that specific act or outcome. In logic this could be translated as :

$$knows(outcome(action)) \ \& \ does(action) \rightarrow intends(action)$$

The way we decide if an agent knows an outcome of an action is by looking at their knowledge. If an agent knows all the conditions(facts/fluents) necessary for an action to have an outcome then they intends said outcome. Furthermore, as we go up along the chain of events, an agent needs to know all the conditions for all the actions after them, for them to intend the final outcome. That is why we keep a list, that we update with the conditions at each step.

If we found that an agent intended the outcome then we update their responsibility and we add this to their particular intention list that each myAgent object has. This will help with determining if an agent was coerced or not.

4.3 Coercion

If a higher authority issues an order to a subordinate, then that subordinate may be coerced to do that action. In that case, more blame should be assigned to the authority and some blame should be subtracted from the subordinate.

We defined coercion as :

$$orders(X, Y) \ \& \ \neg intends(Y) \rightarrow coerced(ObservedAgent)$$

Therefore, to assign responsibilty for coercion we search for all the 'orders()' actions and find the agent who ordered and if the ObservedAgent intended the action. If the ObservedAgent didn't intend then we add responsibility to the ordering agent, but only if they are of a higher rank than the ObservedAgent.

[5] All the numbers assigned for responsibility are dummy numbers. They can be manipulated to reflect better how much responsibility humans assign for each factor. In our case the higher the number, the higher the blame assigned to the respective agent.

5 Train example

To test my program I used the following scenario:

A cargo train is headed for the station. The driver DAUG has been drinking too much and has lost control of the vehicle. If the train isn't stopped it will crash and any cargo will be lost.

There is a switch that can make the train derail onto a secondary track with a heavy object on it that can stop the moving train. A bystander named BOB is also walking on the secondary tracks with his back turned, thus not being able to see the train coming. Throwing the switch would allow the train to stop, saving the cargo but killing BOB.

When the loss of communication with the train is detected, an order is issued over the phone by Head of Transports TOM to employee JOE the station:

TOM : 'If the train isn't stopped we will lose the precious cargo, do whatever it takes to stop it¡

As the train approaches the station, JOE decides to throw the switch. As a result of his action, BOB has lost his life and the cargo is saved.

We try and figure out how to assign blame for Bob's death. Depending on how we change Joe's(our ObservedAgent) and Tom's knowledge of the situation this assignment turns to be very different.

For example if we give Joe no knowledge, but we give Tom the knowledge : *on_rails(bob)* and *head(tom)*. We get Tom with a much higher responsability. On the other hand, if we only give Jow the knowledge of *on_rails(bob)* then Joe is the one with higher blame.

6 Further improvements

6.1 Low-level improvements

There are various ways this could be improved. One of the main improvements is regarding String manipulation and dealing with the difference between Actions and Conditions. Also, adding more flexibility when it comes to the agents who perform the actions. At the moment, we are bound by the ObservedAgent's actions.

Another improvement would be finding bindings not just for actions, but also for conditions. For example if we have a variable Y only in the conditions, we have to find a state which contains those conditions with that variable Y bound to the same components.

6.2 Algorithm and assignment improvements

One upgrade that could be done is adding more factors to the coercion aspect. Such as dialog between the agents, to see if the coerced agent had other options or if he protested or not. That could influence how we percieve the blame assignment.

Another upgrade, would be adding the time element. Maybe at the time the agent accepted to do an action he didn't know certain information, but he found out later. The development of actions and effects in a time constraint environment would make a much more realistical model and could improve how we process the information.

A final improvement would be on the actual backtracking. This could be done by tracing back not only the actions that produced the effect, but also tracing back what produced the conditions for the effect to happen.

6.3 Final words

This project had a great impact on how I view the topic of Intention Recognition and more importantly that of Computational Morality. The human thinking process is very complex when deciding what happened in a situation and how to assign blame to the agents involved. It relies both on facts and on people's psychology.

The challenge is to model the brain's processes and to make a computer think like a human being. This is an incredible task and much work would have to be done for an AI complete judge to be created. I know I only scratched the surface of what can be done in terms of computational assignment of responsibility, but I hope to continue working on this project and further develop it.