My Research in Knowledge Representation

Dr. Richard Watson

The Knowledge Representation Lab

November 12, 2013

Outline of Talk

1. My research motivation
2. Background
   - Answer Set Programming
   - Epistemic Specifications
   - Action Languages
3. My research projects

My Research Motivation

How to represent and reason about the knowledge of intelligent agents?

Answer Set Programming

Answer Set Prolog is a language created by Gelfond and Lifschitz in 1988 and expanded in 1991.

An atom is a formula of the form:
$p(\vec{X})$
An atom represents a statement which an agent may believe is true, believe is false, or hold no belief about.

For example:
bird(tweety)
may represent the statement that tweety is a bird.

Answer Set Programming - Continued

A literal is an atom or its strong negation (e.g. bird(tweety), ¬bird(tweety)).

Rules have form:
l_1 or ... or l_k :- $l_{k+1}, \ldots, l_m$, not $l_{m+1}, \ldots,$ not l_n.
where each l_i is a literal.

The "not" is called weak or default negation. A formula "not l" can be read as "the agent has no reason to believe l."

A program is a collection of rules.

Answer Set Programming - Continued

Given a program, the answer sets of the program are collections of literals that a rational agent could possibly believe are true.

When we say "rational agent" we mean an agent that follows the rationality principle that one should only believe what one is forced to believe.

Example - Birds

fly(X) :- bird(X), not ab(X), not ¬fly(X).
¬fly(X) :- penguin(X).
ab(X) :- hurt(X).
bird(tweety).
bird(sam).
bird(sally).
penguin(sam).
hurt(sally).

This program has one answer set: {bird(tweety), bird(sam), bird(sally), penguin(sam), hurt(sally), ¬fly(sam), ab(sally), fly(tweety)}

Example - Sudoku Solver

```
num(1..9).
1{pos(A,X,Y):num(A)}1 :- num(X), num(Y).
inregion(X,Y,((X-1)/3)*3+((Y+2)/3))
        :- num(X), num(Y).
:- pos(N,X,Y1), pos(N,X,Y2), Y1 != Y2, num(X),
        num(Y1), num(Y2), num(N).
:- pos(N,X1,Y), pos(N,X2,Y), X1 != X2, num(X1),
        num(X2), num(Y), num(N).
:- pos(N,X1,Y1), pos(N,X2,Y2), X1 != X2, Y1 != Y2,
        inregion(X1,Y1,R), inregion(X2,Y2,R), num(X1),
        num(X2), num(Y1), num(Y2), num(N), num(R).
```

Example - Sudoku Input File

pos(5,1,7).
pos(9,1,9).
pos(7,2,3).
pos(2,2,4).
pos(8,2,7).
pos(6,3,2).
.
.
.

Epistemic Specifications

Introduced by Gelfond in 1991. Syntactically very similar to answer set programming, except there are two modal operators added: K and M.

Given a literal *l*:
K *l* intuitively means "*l* is known to be true."
M *l* intuitively means "*l* may be believed to be true."

Instead of answer sets, epistemic logic programs have world views, which are collections of belief sets (where a belief set is analogous to an answer set).

Example - Scholarship Eligibility

Suppose we have the following rules concerning scholarship eligibility:

- All students with high GPAs are eligible for scholarships.
- Minority students with fair GPAs are eligible.
- A student is not eligible if their GPA is not fair or high.
- If a student's eligibility can not be determined, the student should be interviewed.

Suppose Mike has either a high or fair GPA, but do not know if Mike is a minority or not.

Example - Scholarship Eligibilty

fairGPA(mike) or highGPA(mike).
eligible(X) :- highGPA(X).
eligible(X) :- minority(X), fairGPA(X).
¬eligible(X) :- ¬fairGPA(X), ¬highGPA(X).
interview(X) :- not eligible(X), not ¬eligible(X).

Answer sets:
{fairGPA(mike), interview(mike)}
{highGPA(mike), eligible(mike)}

Example - Scholarship Eligibilty

fairGPA(mike) or highGPA(mike).
eligible(X) :- highGPA(X).
eligible(X) :- minority(X), fairGPA(X).
¬eligible(X) :- ¬fairGPA(X), ¬highGPA(X).
interview(X) :- ¬K eligible(X), ¬K ¬eligible(X).

World view:
{{fairGPA(mike), interview(mike)},
{highGPA(mike), eligible(mike), interview(mike)}}

Action Languages

Action language follow in the line of research that started with Situation Calculus by McCarthy and Hayes in 1969.

In 1993, Gelfond and Lifschitz introduced Action Languages. Action Languages can be thought of as formal models of the part of natural language that is used for describing the effects of actions.

Semantically they can be viewed as concise descriptions of transition diagrams of dynamic systems.

Action Language - $\mathcal{AL}$

One of the most basic standard Action Languages is $\mathcal{AL}$.

$\mathcal{AL}$ has only three types of rules:
A causes f if $p_1, \ldots p_n$
f if $p_1, \ldots p_n$
A impossible if $p_1, \ldots p_n$
where A is an action and f and each p_i *are fluents.*

$\mathcal{AL}$ Example - the Yale Shooting Problem

Suppose we have the following dynamic domain:

- there is a turkey which may be alive or not,
- the turkey can be running or not,
- if the turkey is dead it cannot be running,
- there is a gun which may be loaded or not,
- the agent has two actions: loading the gun and pulling the trigger,
- pulling the trigger causes the gun to be unloaded,
- if the agent pulls the trigger and the gun is loaded, then the turkey will be dead.

$\mathcal{AL}$ Example - the Yale Shooting Problem

The Yale shooting domain can be represented by the following rules:
load causes loaded
pull_trigger causes ¬loaded
pull_trigger causes ¬alive if loaded
¬running if ¬alive

Action Languages - Continued

Given a domain description, we can then perform a variety of reasoning tasks such as prediction, planning, and diagnosis. Normally, a domain description is translated into an answer set program that captures the semantics of the Action Language. Each of the tasks mentioned above then becomes a problem of finding answer sets.

My research projects

Application - Modeling Safety Cases

Work with: Dr. Rushton and Dr. Gelfond

In many projects in industry, safety cases must be made to ensure that project is safe (i.e chance of a critical failure is acceptably low). Currently such safety cases are done using fairly informal arguments. While the arguments tend to follow chains of reasoning, the connections are not always clear and, as a result, are not always convincing.

We are working on a formal, logic based framework (most likely using answer set programming) that could be used for such safety arguments. With such a formal system, properties of a safety case can be proven, thus verifying if the safety conditions are ensured by the arguments or not.

Language Development - Epistemic Specifications

Work with: Patrick Kahl, Dr. Gelfond, and Dr. Zhang

As mentioned earlier, Dr. Gelfond introduced Epistemic Specification in 1991. He later proposed some changes in 1994. Due to a variety of reason, not the least of which being computational issues, very little was done with the language since that time. Dr. Gelfond was aware of an issue though. In his semantics the program:
p :- K p
has two world views $\{\{\}\}$ and $\{\{p\}\}$. He strongly believed the second one should not be a world view.

In 2011 he came up with a new semantics which fixed this issue. However the new semantics dropped M as a separate modal operator.

We believe that M is useful in modeling some problems and are working on developing a semantics that properly handles M. Patrick is also working on a solver for Epistemic Specifications.

Language Development - Support Trees

Work with: Patrick Kahl and Dr. Gelfond

One direction of research on Epistemic Specifications was a semantics based on support. In simple cases, support is the chain of rules that shows that something must be true. For example, if we have two rules:
a.
b :- a.
then "a" is supported because it is given as a fact and "b" is supported by the second rule since "a" is supported.

Language Development - Support Trees

An alternative, but equivalent, semantics based on support has been given for large subclasses of answer set programs. We are attempting to expand these ideas for Epistemic Specifications. Due to the nature of Epistemic Specifications, a chain of rules will not work. It will require a tree structure. Beyond defining semantics for Epistemic Specifications, such support trees may also provide a semantics for all answer set programs, not just subclasses.

Language Development - Action Language $\mathcal{H}$

Work with: Dr. Chintabathina and Dr. Gelfond

Action Language $\mathcal{H}$ is an extension of Action Language $\mathcal{AL}$ that allows for real time (rather than simply time steps) and for time dependent processes. For example, imagine the action of dropping a ball. The result of such an action is that the height of the ball is a function of time based on when the ball was dropped and the height it was dropped from. Such effects of actions were impossible to model in $\mathcal{AL}$ but can easily be modeled in $\mathcal{H}$. Development of $\mathcal{H}$ was completed in 2010, but a suitable translation into answer set programming does not exist for all but simple domains and simple tasks. This is primarily due to a lack of a suitable answer set solver.

Answer Set Solvers - SigmaSolver

Work with: Amanda Videtich

SigmaSolver is an answer set solver based on the algorithm for a previous solver called Asperix. We are modifying and improving the Asperix algorithm. The resulting algorithm will work for a larger class of problems than Asperix did. In addition it allows for sorts and user defined functions. We are working on an implementation of the new algorithm (which will be written in SequenceL). We believe the new solver will be able to handle Action Language $\mathcal{H}$ translations much better that other solvers.

Multi-Agents - Planning

Work with: Greg Gelfond and Deb Hagar

For the most part, action languages have been used for modeling domains in which a single agent is operating. In 2007, as part of Greg Gelfond's Masters thesis, we developed a framework for multi-agent systems where an agent could ask another agent to perform tasks for them. This allowed for cooperative planning. The framework was a first step - quite a few simplifying assumptions were made. For example, requests are never refused and agents are always able to accomplish tasks they asked to do. We are working to lift some of the restrictions in order to make the framework more general.

Multi-Agents - Trust

Work with: Dr. De Vos

As was seen earlier, answer set programming can be used to model the beliefs and reasoning of a rational agent. In a multi-agent environment such agents may share information between one another. As an agent's knowledge consists of beliefs, not necessarily facts, information received from other agents may be contradictory and may disagree with beliefs held by the agent receiving the information. In this research we are working on ways to determine which information received should be trusted and which should be ignored.

Multi-Agents - Normative behavior

Work with: Dr. De Vos and Dr. Balke

Intelligent agents often work in a environment where they are subject to social norms. While an agent may be capable of performing a variety of actions, some actions violate norms and thus carry consequences beyond the normal effects of the action. In this research we wish to develop frameworks and methodologies for modeling such agents and their behaviors.

The End

Questions and/or Comments?