Some Thoughts on the Development of Action Theories

Michael Gelfond

Texas Tech University

July 10, 2011

Introduction

In this talk I'll discuss some history and some recent developments in the Theory of Action and Change –the area which has been one of my major research interests for more than 25 years.

I'd like to acknowledge my many collaborators without whom my work could not have been possible but who are of course not responsible for what I have to say.

This is not an attempt to give an overview. There are books written on the subject and they only cover a small part of work done in this area.

Introduction

Rather the talk gives a simplified personal story of my own research.

I describe some questions I attempted to address in this research, the choices I was confronted with, and methodological assumptions used in the process.

In my experience this type of information may be useful for newcomers to the field (and even for those who worked in it for some time) but not easily extractable from the published work.

The Goals

• **Goal: understand how to build software components of intelligent agents capable of reasoning and acting in changing environment.**

• **Main hypotheses: to exhibit intelligent behavior the agent should have a mathematical model of its environment and its own capabilities and goals.**

• **Main problem: what should this mathematical model look like?**

The Initial Landscape

The first answer I learned from John McCarhty: the agent's model of the world should be a logical theory describing the agent's domain.

Such a theory should contain description of relevant actions and their direct and indirect effects.

In the 80's I became familiar with two attempts to build such theories:

- **Situation Calculus (John McCarthy)**
- **Event Calculus (Robert Kowalski and Marek Sergot)**

Situation Calculus: the Original Ontology

1. **Objects of three types:**
   - **fluents – functions of time;**
   - **actions – atomic, deterministic and sequential;**
   - **situations – complete states of the universe at an instant of time.**
2. **Time is branching and discrete;**
3. **Changes in the values of fluents can only be caused by execution of actions.**

Situation Calculus: The Language

In the basic language of situation calculus we can

- **Name situations, e.g.**

$$s_0, \qquad res(load, s_0), \qquad res(shoot, res(load, s_0))$$

- **State that a fluent is true in them, e.g.**

$$holds(dead, res(shoot, s_0))$$

- **Use classical first-order logic to describe effects of actions and other properties of dynamic domains.**

Problems

1. **Ontology is *unclear*, e.g. are statements**

$$s_0 = res(a, s_0)$$

$$holds(f, res(a, s_0)) \wedge holds(g, res(b, s_0))$$

   consistent with the ontology?
2. **Ontology is *not sufficiently rich* – no concurrent actions, no continuity, etc.**
3. ***Adequacy of classical logic* is in question: frame problem, etc.**

Questions to Answer

1. **Can the intuition behind basic notions of Situation Calculus be clarified?**
2. **Can its ontology be enriched without basic change of the original notions?**
3. **What logical language and what entailment relation should be used to axiomatize Situation Calculus?**

Finding Logic: Circumscription

The original idea due to McCarthy was to:

use the language of first-order logic with entailment defined by truth in the set of models minimal with respect to some ordering relation.

Later the language was expended to include some second order constructs.

Using this formalism Baker, Lifschitz, Reiter, Fangzhen Lin and others found partial solutions to the frame problem which used non-trivial orderings on models.

Problems with Minimization Based Approaches

I admired this work but:

• The ordering relations were too complex and seemed tailored toward a particular problem.

I wanted to solve frame problem as suggested by McCarthy – by representing the inertia axiom as a default.

• Minimization changes the standard meaning of classical logical connectives in sometimes unpredictable ways.

I wanted the language with reasonably clear intuitive meaning of the connectives.

Problems with Minimization Based Approaches

• The distance from theories in Situation Calculus to implementable specifications seemed rather long.

(There were, however, interesting attempts to reduce circumscriptive theories to classical first-order theories and even to Prolog programs.)

• In my mathematical philosophy I am a constructivist. Among other things that means that I prefer to avoid the use of complex set theory which forms the basis of Tarskian semantics.

Finding Logic: Logic Programming Languages

Because its inability to represent incomplete information and difficulties with semantics for negation as failure logic programming had, for a long time, been viewed unsuitable to serve as language of SitCalc.

Two papers with V. Lifschitz have been instrumental in refuting this view:

- **"Classical Negation in Logic Programs and Disjunctive Databases"** ***introduced Answer Set Prolog.***
- **"Representing Actions and Change by Logic Programs", 1993,** ***formulated SitCalc in the new language and gave partial solution of the frame problem via the inertia axiom.***

Enriching Ontology

"*For a long time many authors took it for granted that the Situation Calculus was too ontologically impoverished to be able to represent concurrent actions. This fallacy was finally laid to rest by*

Gelfond, M., Lifschitz, V., and Rabinov, A. What are the Limitations of the Situation Calculus? 1991."

M. Shanahan, Solving the Frame Problem.

The same paper addressed some other important limitations and influenced further developments.

One more limitation

There is one more limitation of SitCalc which remained largely unnoticed for a long time — its *inability to express actual occurrences of actions and observations of fluents*.

$holds(f, res(a, s_0))$ is not a statement of fact — it is a *hypothesis* which says "if a were executed in s_0 then f would be true in $res(s_0)$".

The original SitCalc was a language for hypothetical reasoning.

(Note that the original Event Calculus could represent actual events but could not deal with hypothetical reasoning.)

One more limitation (Continued)

The problem was addressed in the dissertation:

- **J. Pinto, "Temporal Reasoning in Situation Calculus", 1994.**

and, independently, in

- **C. Baral, M. Gelfond and A. Provetti, "Reasoning About actions: Laws, Observations and Hypotheses", 1995 (Journal version 1997)**

Both papers extended SitCalc to deal with *actual time line* and *history of actions and observations*.

Another Question: Correctness of Action Theories

How to ensure that axiomatization of an action domain is reasonably accurate? There were two methods:

- ***Reasonable behavior on a good collection of test cases.***
- ***Equivalence of action theories for the same domain written in different formalisms by different people.***

But different theories were built on different, and sometimes incompatible, underlying assumptions which normally were nor explicitly stated.

Proving entailment for different formalisms was difficult. Even more difficult was to prove equivalence of theories.

The Next Level of Abstraction

Vladimir and I decided to try another criteria:

- **Find a simpler, more abstract (but possibly less general) way to model dynamic systems.**
- **Prove correctness of action theory of a given domain with respect to its more abstract model.**
- **Prove equivalence of theories formulated in different formalisms by showing their equivalence to the abstract model.**

Transition Diagrams and Recorded Histories

After a number of tries and errors Vladimir and I came up with a "new" model of a dynamic system:

- **A transition diagram whose nodes represent physically possible states of the domain and whose arcs are labeled by actions.**
- **Recorded history – a collection of assertions about the behavior of the dynamic system up to some point n.**

The first describes all physically possible trajectories of the system, the second – trajectories compatible with the corresponding assertions.

Action languages are tools for describing such models.

Action Language A

"*Syntax and semantics of* A *precisely describe the class of action domains under consideration and the intended ontology of action.*

The representation of a particular domain in A *can be viewed as a high level specification for the task of formalizing this domain in logic programming or another logic based formalism.*

The soundness and completeness of each formalization become precisely stated mathematical questions. The possibilities and limitations of different representation methods can be compared in a precise fashion."

Conclusion of 1993 paper.

First Proofs of Correctness

The idea seemed to work. In 1993 paper with Vladimir and in 1995 paper with Chitta and Alessandro Provetti we were able to prove soundness of logic programming axiomatizations of a broad class of action descriptions of $\mathcal{A}$ and even soundness (and sometimes completeness) of Prolog based algorithms for temporal projections.

The proofs used non-trivial mathematics of logic programming: properties of signed programs, results on termination, non-floundering, and occur check, relation between ASP and Clark's semantics, etc.

Weakness of A

- By design, A was a weak language. No state constraints, no concurrent actions, etc.

- But A was not simple enough. There was no clear separation between description of a transition diagram and and description of trajectories of interest to the agent.

The first problem was solved by McCain and Turner. The structuring of action theories evolved gradually.

In Action Languages, 1998, Electronic Transactions to AI, we divided action theory into two parts. The second part was called *query language*. It is also referred to now as *recorded history*.

Action Description Language $\mathcal{B}$

Action description languages define transition diagrams of dynamic systems. There are two families of such languages which evolved from languages $\mathcal{B}$ and $\mathcal{C}$.

Action descriptions of $\mathcal{B}$ are collections of statements of the form:

- **dynamic causal law:**

$$\mathfrak{a} \textbf{ causes } \mathfrak{f} \textbf{ if } \mathfrak{p}$$

- **state constraint:**

$$\mathfrak{f} \textbf{ if } \mathfrak{p}$$

- **impossibility condition:**

$$\textbf{impossible } \mathfrak{a} \textbf{ if } \mathfrak{p}$$

Action Description Language $\mathcal{B}$

$\mathcal{B}$ is closely related to logic programming. In fact its laws can be translated into rules

- **dynamic causal law:**

$$a \textbf{ causes } f \textbf{ if } p$$

$$holds(f, T+1) \textbf{ :- } holds(p, T), occurs(a, T).$$

- **state constraint:**

$$f \textbf{ if } p$$

$$holds(f, T) \textbf{ :- } holds(p, T).$$

- **impossibility condition:**

$$\textbf{impossible } a \textbf{ if } p$$

$$\neg occurs(a, T) \textbf{ :- } holds(p, T).$$

Action Description Language $\mathcal{B}$

To define σ' in a transition (σ, a, σ') of the diagram defined by action description A of $\mathcal{B}$ we construct a logic program consisting of

- **encoding of laws of A as above,**
- **inertia axiom:**

$$holds(F, T+1) \text{ :- } holds(F, T),\ not\ \neg holds(F, T+1)$$

with T ranging over $\{0, 1\}$,

- **encoding of σ, a:**

$$\{holds(f, 0) : f \in \sigma\} \cup \{occurs(a, 0)\}.$$

σ' is defined by an answer set of this program.

Action Description Language $\mathcal{C}$

Language $\mathcal{C}$ is syntactically close to $\mathcal{B}$.

Action description of a (simplified and syntactically incorrect version) of $\mathcal{C}$ has impossibility conditions as in $\mathcal{B}$ and causal laws of the form:

- **dynamic causal law:**

$$a \textbf{ causes } has\_a\_cause(f) \textbf{ if } p$$

- **state constraint:**

$$has\_a\_cause(f) \textbf{ if } p$$

Action Languages $\mathcal{B}$ and $\mathcal{C}$

Despite many similarities $\mathcal{B}$ and $\mathcal{C}$ are very different languages based on different intuitions and different logical formalisms.

- **The semantics of $\mathcal{B}$ incorporates the *inertia axiom* – "*Things normally stay unchanged*" and is based on *ASP*.**

- **$\mathcal{C}$ incorporates *causality principle* – "*Everything true in the world must be caused*" and is based on *Causal Logic* of McCain and Turner.**

Action Languages $\mathcal{B}$ and $\mathcal{C}$

• **Addition of a recursive constraint "f if f" does not change a theory of $\mathcal{B}$.**

But addition of analogous constraint "has_a_cause(f) if f" does change a theory of $\mathcal{C}$.

• **Theories of $\mathcal{C}$ can be translated into ASP but the translation is somewhat less natural, e.g.**

$$\mathrm{has\_a\_cause(f)\ if\ p}$$

is translated as

$$\mathrm{holds(f,T)\ :-\ \ not\ \neg holds(p,T)}$$

Action Languages $\mathcal{B}$ and $\mathcal{C}$

Both, $\mathcal{B}$ and $\mathcal{C}$ gave rise to families of languages sharing their basic principles.

$\mathcal{B}$ evolved into $\mathcal{AL}$, $\mathcal{H}$ and $\mathcal{ALM}$; C into C^+ and $\mathcal{MAD}$

Unfortunately I am not aware about results establishing precise relationship between languages from these two families.

I found *Causal Logic* to be less intuitive and, in some imprecise sense, less general than *ASP*, and hence opted for $\mathcal{B}$.
Another choice is also reasonable but it shall not be based on the alphabetical order.

The Impact of Action Languages

Rather early it became clear that we were somewhat successful in our original goals: Action languages provided high level specification which

- **allowed to prove soundness and (sometimes) completeness of various axiomatizations;**
- **helped to establish precise relationship between axiomatizations (including those given in different logical languages);**
- **provided a tool for classification of action theories.**

The Impact of Action Languages (Change of model)

But there was another important impact:

- **The language of mathematical model of dynamic domain changed from logic to automata.**
- **Logic is used to describe this (intended) model. Another logic is used to reason about it.**
- **The model was provided with structure. In addition to action description and history Jorge Lobo and I later added description of policies specifying desirable trajectories of the agent. (Authorization and Obligation Policies in Dynamic Systems, ICLP08, 2008)**

The Impact of Action Languages (New Methodology)

There are now foundations of *methodology* for building intelligent agents which is based on *action languages* and *logic programming*.

1. Action languages define *mathematical model* of agent and its environment.
2. Problems like planning, diagnostics, etc. are reduced to questions about this model.
3. The process of answering these questions is reduced to reasoning about logic programs.
4. Correctness of the corresponding algorithms is proven with respect to original action language specification.

What Needs to be Done? Improve ASP Solvers

1.To decrease the gap between high level and executable specifications:

- ***Integrate different reasoning mechanisms*** **such as *ASP*, *CLP*, *SMT*, etc. to improve efficiency of logic programming engines.**

S. Baselice , P. A. Bonatti , M. Gelfond. Towards an integration of answer set and constraint solving, 2005 Mellarkod and M. Gelfond, Enhancing ASP systems for Planning with Temporal Constraints, LPNMR 2007

- **Develop and implement *incremental* reasoning algorithms which accommodate interaction of agent with the environment. (See Reactive Answer Set Programming by Potsdam group)**

What Needs to be Done? Modular Action Languages

Gelfond, M., Inclezan, D. (2009). Yet Another Modular Action Language. In Proceedings of SEA-09, pp. 64–78.

“ modular action language $\mathcal{ALM}$, which extends action language $\mathcal{AL}$ by providing it with a modular structure and the ability to separate the definition of *classes of objects* of the domain from the definition of *instances* of these classes. This, together with the means for defining actions and fluents of the domain as special cases of previously defined ones facilitates the stepwise development, testing, and readability of a knowledge base, as well as the development of knowledge representation libraries.”

What Needs to be Done? Mathematics

Build serious mathematical theory of action languages, e.g.

Give sufficient conditions to guarantee that action description in $\mathcal{B}$ and $\mathcal{C}$

- **is deterministic;**
- **for every state σ and action a, a is executable in σ iff there is impossibility condition "impossible a if p" in D such that p is true in σ.**

What Needs to be Done? Mathematics

T' *approximates* T **if nodes of** T' **are partial states of** T **and paths of** T' **preserve some properties** T**, e.g.**

if a sequence α **of actions moves** s **to** s' **in** T' **then for every** $\sigma \in \mathsf{T}$ **which is compatible with** s**,** α **moves** σ **to a state** σ' **in** T **compatible with** s'**.**

What Needs to be Done? Action Architecture

Design new languages, or integrate existing languages like Golog with action languages.

Build agent environments combining discrete and continuous computation.

Use this to build interesting (but not necessarily practical) agents.

Build Libraries of Commonsense Knowledge.

What Needs to be Done? Social Issues

- **Create a distinct community with close ties to ASP, non-monotonic logics, etc.**
- **Better explain internal problems and achievements of the field.**
- **Better explain connections with other fields including some sub-fields of AI and software engineering.**
- **Find better ways to share and publish the results.**

Other Approaches: Situation Calculus

Now minimization based SitCalc is a well developed logical theory.

R. Reiter, "Knowledge in Action – logical foundations for specifying and implementing dynamic systems"

contains accurate account of very rich SitCalc with parallel actions knowledge producing actions, continuous time, etc.

SitCalc based programming language Golog allows to program robots with non-trivial reasoning powers.

Despite its well developed theory SitCalc solution of the frame problem was only applicable to domains without cyclic dependencies between fluents.

This seems to change this year.
In "Causal Theories of Actions Revisited", 2011 Fangzhen Lin and Mikhail Soutchanski presented a *general solution via an extra minimization*.

Last Slide

THANK YOU!

Modules of $\mathcal{ALM}$

Syntactically a *module* can be viewed as a collection of declarations of *sort*, *fluent* and *action* classes of the system.

module name
sort declarations
fluent declarations
action declarations

Example: move_between_areas

module move_between_areas

sort declarations

things : **sort**

movers : things

areas : **sort**

fluent declarations

loc_in(things, areas) : **inertial fluent**

axioms

$\neg$loc_in(T, A_2) **if** disjoint(A_1, A_2), loc_in(T, A_1).

end of loc_in

Example: move_between_areas

action declarations

move : **action**

attributes

actor : movers

origin, dest : areas

axioms

move **causes** loc_in(O, A) **if** actor = O, dest = A.

impossible move **if** actor = O, origin = A, ¬loc_in(O, A).

impossible move **if** origin = A_1, dest = A_2, ¬disjoint(A_1, A_2).

end of move

Comments

- Actions of a module are action classes.
- Sorts, fluents, actions, and axioms of the module are *uninterpreted*.
- Semantically, a collection of modules can be viewed as a *mapping* of possible interpretations of the symbols of the domain into the transition diagram describing a dynamic system.
- *System description* is a set of modules followed by an interpretation of its symbols.
- Modules can be combined into *libraries* and imported from there using *import* statements.

Interpreting the Symbols

structure of basic_travel

sorts

michael, john **in** movers

london, paris, rome **in** areas

actions

instance move(O, A_1, A_2) : move

actor := O

origin := A_1

dest := A_2

statics

disjoint(london, paris).

disjoint(paris, rome).

disjoint(rome, london).

Actions as Special Cases

module carrying_things

sort declarations

areas, things : **sort**

movers, carriables : things

fluent declarations

holding(things, things) : **inertial fluent**

loc_in(things, areas) : **inertial fluent**

axioms

loc_in(T, A) ≡ loc_in(O, A) **if** holding(O, T).

Actions as Special Cases

action declarations

carry : move

attributes

carried_thing : carriables

axioms

impossible carry **if** actor = O,
carried_thing = T,
¬holding(O, T).

The language introduced by Daniela Inclezan and myself has already changed my KR style. It is however still work in progress.