Logic Programming and Reasoning about Actions and Time

Michael Gelfond
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968
mgelfond@cs.utep.edu

Formal Theories of Actions are needed to:

1. **describe** dynamic behavior of programs, databases, robots, and other reasoning and acting agents;
2. **build** agents capable of performing actions and reasoning in the dynamic world;
3. **reason** about such agents.

Various disciplines put different emphasis on such theories.

Programming theory: simple domain, complex actions.

Artificial intelligence: complex domain, simpler actions.

I will concentrate on AI side of the story.

In AI our goal is to model and build agents capable of reasoning, planning and acting in a changing environment.

An agent is normally given some description of its domain and the set of goals.

Its task is to use his knowledge and reasoning and acting ability to achieve these goals.

Questions:

- What is a suitable ontology for dynamic domains?
- What languages can we use to describe such domains?
- What entailment relation(s) model commonsense reasoning about actions?
- What are the assumptions underlying this reasoning?

Possible answers:

Actions can be **atomic, concurrent, sequential, instantaneous** or **continuous, deterministic** or **non-deterministic**, . . .

Time can be **linear** or **branching, discrete** or **continuous**, represented by **points** or by **intervals**, . . .

Languages can be based on classical or non-classical **logics**, have **truth-theoretic** or **probabilistic** semantics, be **specialized** or **general purpose**, . . .

The answer should provide **modularity** (small changes in the agent's knowledge about the domain shall not cause big changes in his knowledge base).

We start with simple ontology and simple language and gradually move to more complex theories

Situation Calculus

Conceived by J. McCarthy in 1967. Is not born yet.

1. Objects of three types:

- **fluents** (functions of time)
- **actions** (atomic, deterministic and inertial)
- **situations** (sequences of actions representing possible histories of the world)

2. Time (represented by situations) is branching and discrete;

3. Changes in the values of fluents can only be caused by execution of actions;

4. First-order language is used to describe dynamic domains.

In the basic language of situation calculus we can:

- **name situations** - s_0, $res(shoot, res(load, s_0))$ (performing (an instance) of an action A in a situation S moves the world into a situation $res(A, S)$)
- **state that a fluent is true in them** - $holds(dead, res(shoot, res(load, s_0)))$
- **describe effects of actions** - $h(loaded, S) \supset \neg h(alive, res(shoot, S))$

  Inertia axiom (specify what fluents are NOT changed by actions)

  $$(F \neq alive \vee \neg h(loaded, S)) \supset$$
  $$(h(F, S) \equiv h(F, res(shoot, S))$$

Advantages:

- First-order logic has well understood semantics and proof theory.
- Branching structure of time allows hypothetical reasoning and hence is well-suited for planning.

Disadvantages:

- Unclear ontology

  Is statement $s_0 = res(a, s_0)$ consistent with the ontology?
- Can the ontology be enriched without basic change in the language?
- How can we write the Inertia Axiom in more complex domains?

Example: Suppose we want to expand previous theory by new effect axiom: $\neg h(loaded, res(shoot, S))$.

Old Inertia Axiom should be replaced by the new one:

$$((F \neq alive \land F \neq loaded) \lor \neg h(loaded, S)) \supset$$
$$(h(F, S) \equiv h(F, res(shoot, S))$$

The necessity to remove information from the knowledge base can be considered a substantial drawback of the method. The situation becomes much more complicated if the theory includes relationship between fluents, e.g.

$$h(alive, S) \equiv \neg h(dead, S) \qquad h(fly, S) \supset h(alive, S)$$

Attempts to find general formulation of Inertia Axiom greatly contributed to the development of NON-MONOTONIC LOGICS.

Formalizing defaults

Formalization of the Inertia Axiom can be viewed as a specific instance of a more general problem of representing **defaults**, i.e. statements of the form "**objects of the type A normally have property P**".

Reasoning with defaults seems to be inherently non-monotonic, i.e. new information can force a reasoner to withdraw his previous conclusions.

Defaults (sometimes called normative statements) may be understood as communication agreements and expressed in logic programming and other non-monotonic logics.

NONMONOTONIC LAW OF INERTIA: **Things normally stay as they are**.

Shooting scenario in LP

Inertia Axiom

```
true_after(F,[A|S]) :- true_after(F,S),
                       not ab(F,A,S).
```

Effect Axiom

```
ab(alive,shoot,S) :- true_after(loaded,S).
```

Initial State

```
true_after(alive,[]).   true_after(loaded,[]).
```

$?true\_after(alive, [shoot])$ NO

$?true\_after(loaded, [shoot]$) YES

To specify that the gun is one-shooter ADD:

```
ab(loaded,shoot,S) :- true_after(loaded,S).
```

EVENT CALCULUS (Kowalski and Sergot)

The simplified version has linear discrete time, fluents and action-tokens and action-types.

Domain Dependent Axioms:

```
initiates(E,loaded) :- act(E,load).
terminates(E,loaded) :- act(E,shoot).
terminates(E,alive) :- act(E,shoot),
                       happens(E,T),
                       holds_at(loaded,T).


happens(e0,0).      initiates(e0,alive).


happens(e1,1).      act(e1,load).
happens(e2,2).      act(e2,shoot).
```

Inertia Axiom

```
holds_at(P,T) :- happens(E1,T1),
                 initiates(E1,P),
                 T1 < T,
                 not clipped(T1,P,T).

clipped(T1,P,T) :- happens(E2,T2),
                   T1 < T2,
                   T2 < T,
                   terminates(E2,P).

? holds_at(alive,1)  -- Yes
? holds_at(alive,3)  -- No
```

Even though the above representations of the shooting scenario are simple and computationally efficient they were not taken seriously by the AI community. The main (technical) reason was the weakness of the logic programming language. It is difficult for instance to represent incomplete information without negation and disjunction, etc. How do we say "Initially, the gun is not loaded but there is no information on turkey's health"?

Recent extensions of LP languages by explicit negation, disjunction, etc. change the situation somewhat. Initial state above can be represented by

$\neg true\_after(loaded, [])$.

Recall, that a program Π answer "Yes" to a query Q if Π entails Q, "No" if Π entails $\neg Q$ and "Unknown" otherwise.

In the next two examples we expand our language by allowing

- Explicit Negation.
- Non-frame fluents, i.e. fluents defined in terms of basic (frame) fluents. Non-frame fluents can be used only as a shorthand and can (in principle) be eliminated from a domain description.
- Non-executable actions.
- Static causal laws.

We need the notion of fluent literal - fluent or expression $neg(f)$ where f is a fluent. We will assume that the programs below are extended logic programs containing a rule:

$$\neg true\_after(F, S) \text{ :- } fluent(F), true\_after(neg(F), S)$$

Blocks World in LP

DOMAIN DESCRIPTION

Types:

```
block(a).  block(b).  block(c).  block(t).


frame_fluent(on(_,_)).


action(move(X,Y)) :-
     block(X),
     block(Y),
     diff(X,Y),
     diff(X,t).
```

Executability and Causal Laws:

```
possible_if(move(X,Y),[clear(X),clear(Y)]).


causes(move(X,Y),on(X,Y),[]).

causes(move(X,Y),neg(on(X,Z)),[]) :-

     diff(Y,Z).
```

Initial Situation:

```
initially(on(a,b)).

initially(on(b,c)).

initially(on(c,t)).


initially(neg(on(X,Y))) :-

     not initially(on(X,Y)).
```

Definitions of Non-frame fluents

clear and **occupied** are non-frame fluents. They are defined in terms of frame fluents and are not subject to the inertia axiom. (strictly speaking $occupied(X)$ should be viewed as a shorthand for $neg(clear(X))$).

```
true_after(clear(t),S).


true_after(clear(X),S) :-
     not true_after(occupied(X),S).


true_after(occupied(X),S) :-
     block(Y),
     true_after(on(Y,X),S).
```

DOMAIN INDEPENDENT AXIOMS

```
%true_after(F,A)

%F - fluent or set of fluents,

%A - situation

%mode +  +
```

Effect Axioms

```
true_after(F,[]) :-

     frame_literal(F),

     initially(F).


true_after(F,[A|R]) :-

     frame_literal(F),

     possible_if(A,P1),

     true_after(P1,R),

     causes(A,F,P2),

     true_after(P2,R).
```

Inertia Axiom

```
true_after(F,[A|R]) :-
     frame_literal(F),
     possible_if(A,P),
     true_after(P,R),
     true_after(F,R),
     not ab(F,A,R).

ab(F,A,R) :-
     contrary(F,G),
     causes(A,G,P),
     true_after(P,R).
```

Evaluating sets of fluents

```
true_after([],A).
true_after([H|T],A) :-
     true_after(H,A),
     true_after(T,A).
```

Library

```
frame_literal(F) :-
    frame_fluent(F).
frame_literal(neg(F))  :-
    frame_fluent(F).
eq(X,X).
diff(X,Y) :- not eq(X,Y)
contrary(neg(F),F).
contrary(F,neg(F)).
```

Application - generate and test planner

```
generate([]).
generate([A|S]) :-
     generate(S),
     action(A),
     possible_at(A,S).

possible_at(A,S) :-
     possible_if(A,P),
     true_after(P,S).

plan :-
     write('Enter the goal'),nl,
     read(Goal),
     generate(Plan),
     true_after(Goal,Plan),
     nl,nl,write('Plan is '),write(Plan),nl.
```

Suitcase Domain

Consider a "suitcase" domain (Lin 95) in which there is a suitcase with two locks. We model the domain by naming the locks 1 and 2, introducing actions $unlock(X)$ and $lock(X)$ where X ranges over the locks, and fluents, $locked(X)$ and $open$ (where suitcase being open). The following causal relations are self-evident:

$causes(lock(X), locked(X), [])$

$causes(unlock(X), neg(locked(X)), [])$

Let us also assume that the suitcase is spring-loaded and therefore becomes open when both its locks are unlocked. How do we represent this information?

Static Causal Laws: "fluents cause fluents"

$causes([unlocked(1), unlocked(2)], open)$

Domain dependent axioms

```
frame_fluent(locked(1)).
frame_fluent(locked(2)).
frame_fluent(open).
```

Dynamic causal laws

```
causes(lock(X),locked(X),[]).
causes(unlock(X),neg(locked(X)),[]).
```

Static causal laws

```
causes([neg(locked(1)),neg(locked(2))],open).
```

Initial Situation

```
initially(locked(1)).
initially(neg(locked(2))).
initially(neg(open)).
```

Domain independent axioms

Effect Axioms

```
true_after(F,[]) :-  frame_literal(F),
                     initially(F).

true_after(F,[A|S]) :- frame_literal(F),
                       causes(A,F,P),
                       true_after(P,S).

true_after(F,S) :- frame_literal(F),
                   causes(G,F),
                   true_after(G,S).
```

Inertia Axiom

```
true_after(F,[A|R]) :- frame_literal(F),
                       true_after(F,R),
                       not ab(F,A,R).

ab(F,A,R) :- contrary(F,G),
             causes(A,G,P),
             true_after(P,R).

ab(F,A,R) :- contrary(F,G),
             causes(P,G),
             true_after(P,[A|R]).
```

Correctness Issues

Since the program attempts to capture and refine our intuition of dynamic systems we can not demonstrate its correctness mathematically. We can use mathematics though to check that the program has some properties necessary for its correctness. For instance we can show that the above program is consistent, i.e. has a consistent answer set. Determinism of the system corresponds to uniqueness of this set, etc. This is of course not enough. The above programs are consistent and have unique answer sets but they are incomplete and incorrect.

Incompleteness

$causes(a, f, [p]).$

$causes(a, f, [\neg p]).$

The answer to query

$?true\_after(f, [a])$

is UNKNOWN instead of intuitive YES.

Incorrectness

$initially(alive).$

$causes(shoot, \neg alive, [loaded])$

The answer to query

$?true\_after(alive, [shoot])$

is YES instead of intuitive UNKNOWN.

The error can be corrected by changing the cancelation axioms defining predicate ab. First, let us introduce

$\neg true\_after([F|T], S)$:- $\neg true\_after(F, S)$.

$\neg true\_after([F|T], S)$:- $\neg true\_after(T, S)$.

and change the old cancelation axioms by

```
ab(F,A,R) :- contrary(F,G),
             causes(A,G,P),
             not false_after(P,R).


ab(F,A,R) :- contrary(F,G),
             causes(P,G),
             not false_after(P,[A|R]).
```

(Here $false\_after(P, S)$ stands for $\neg true\_after(P, S)$).

Problem: How can we convincingly demonstrate correctness and completeness of our theories?

Possible solution: Specify dynamic systems we want to model and prove correctness and completeness of our representation w.r.t. these specifications.

- First-order Logic and Circumscription
- Sandewall (Ego-World Semantics)
- Action Description Languages

Next we discuss an action description language $\mathcal{L}$ (Baral, Gelfond, Provetti, to appear in special issue of JLP)

Ontology of L

1. Objects of three types:
   - fluents
   - actions
   - (actual) situation

2. Actions are indivisible, sequential, deterministic and inertial

3. Possible histories form a tree with actual situations located on a finite path corresponding to the actual history

4. Changes in the values of fluents can only be caused by execution of actions

Language

1. Language L_0 allows to express

- **effects of actions**
- **statements about fluents in particular situations**
- **statements about occurrences of actions in particular situations**

2. The alphabet of L_0 consists of three disjoint nonempty sets of symbols F, A and S, for fluents, actions, and situations.

If f is a fluent then f and ¬ f are fluent literals

$\mathcal{S}$ contains two special situations s_0 and s_c called initial and current situations.

$[a_1, \ldots, a_n]$ denotes a sequence of actions.

Effect Laws

Express general knowledge about effects of actions.

$$a \textbf{ causes } f \textbf{ if } p_1, \ldots, p_n$$

a **is an action and** f **and** p**'s are fluent literals.**

The intuitive reading: "f **is true after** a **is executed in any state which satisfies** $p_1, \ldots, p_n$**".**

p**'s are called preconditions of the effect axiom. If the axiom has no preconditions it can be written as**

$$a \textbf{ causes } f$$

Facts

Express particular observations about occurrences of actions and values of fluents.

f **at** s

"f is observed to be true in (actual) situation s**".**

a **occurs_at** s

"action a **was observed to have occurred in situation** s**".**

s_1 **precedes** s_2

"situation s_2 **occurred after situation** s_1**.**

Domain Description - a set of effect laws and facts.

Example

Given: a series of observations about Fred:

(a) when the water pistol was squirted *Fred* **was seen to be** *alive* **and** *dry***,**

(b) in a later moment a shot was fired at *Fred***.**

Suppose also that it is generally known that

(c) *squirting* **makes** *Fred wet***,**

(d) *shooting* **makes** *Fred dead*

Fred's story in $\mathbf{L}_0$

The information from the story can be represented by a domain description D_1

$$\left.\begin{array}{l}
Facts: \\
(p1)\ alive\ \textbf{at}\ \ s_0 \\
(p2)\ dry\ \textbf{at}\ \ s_0 \\
(p3)\ squirt\ \textbf{occurs\_at}\ \ s_0 \\
(p4)\ s_0\ \textbf{precedes}\ \ s_1 \\
(p5)\ shoot\ \textbf{occurs\_at}\ \ s_1 \\
\\
Laws: \\
(p6)\ squirt\ \textbf{causes}\ \ \neg dry \\
(p7)\ shoot\ \textbf{causes}\ \ \neg alive
\end{array}\right\} D_1$$

Assumptions

Domain descriptions are used in conjunction with the following informal assumptions:

(a) Changes in the values of fluents can only be caused by execution of actions.

(b) Effects of executing an action are specified by the effect laws

(c) No actions occur except those specified by the domain description.

A semantics should specify the sets of acceptable conclusions which can be reached from domain descriptions and assumptions (a)-(c).

Semantics: Interpreting the laws

- **state** is a set of fluents.
- fluent f *holds* in a state σ if $f \in \sigma$; $\neg f$ *holds* in σ if $f \notin \sigma$.
- f is an **effect** of (executing) a in σ if there is an effect law "a **causes** f **if** $p_1, \ldots, p_n$" whose preconditions hold in σ. Let

$E_a^+(\sigma) = \{f : f \text{ is an effect of } a \text{ in } \sigma\}$.

$E_a^-(\sigma) = \{f : \neg f \text{ is an effect of } a \text{ in } \sigma\}$.

- **Transition Function**

$\Psi(a, \sigma) = \sigma \cup E_a^+(\sigma) \setminus E_a^-(\sigma)$ if $E_a^+ \cap E_a^- = \emptyset$ and undefined otherwise. Ψ can be naturally expanded to allow sequences of actions as its second parameter.

Semantics: Actual Path

1. Mapping Σ from $\mathcal{S}$ to sequences of actions from the language of D is called **a situation assignment** if it satisfies the following properties:

   - $\Sigma(s_0) = [\,]$
   - for every $s_i \in \mathbf{L}_0$, $\Sigma(s_i)$ is a prefix of $\Sigma(s_c)$.

2. **Interpretation** M of $\mathbf{L}_0$ is a triple (σ_0, Ψ, Σ), where σ_0 stands for the initial state, Ψ is the transition function of D, Σ is a situation assignment of $\mathcal{S}$ and $(\sigma_0, \Sigma(s_c))$ belongs to the domain of Ψ.

3. $\Sigma(s_c)$ is called the **actual path** of M.

Semantics: Interpreting facts

1. For any interpretation $M = (\sigma_0, \Psi, \Sigma)$.

• $(f$ **at** $s)$ is true in M (or satisfied by M) if f is true in $\Psi(\sigma_0, \Sigma(s))$.

• $(\alpha$ **occurs_at** $s)$ is true in M if the sequence $\Sigma(s) \circ \alpha$ is a prefix of the actual path of M.

• $(s_1$ **precedes** $s_2)$ is true in M if $\Sigma(s_1)$ is a proper prefix of $\Sigma(s_2)$

• Truth of non-atomic facts in M is defined as usual.

2. A set of facts is true in interpretation M if all its members are true in M.

To define models we need to formalize the assumption: " No actions occur except those specified by the domain description."

Semantics: Models and Entailment

- An interpretation $M = (\sigma_0, \Psi, \Sigma)$ is a **model** of D if

  1. Facts of D are *true* in M.

  2. There is no other interpretation $N = (\sigma_0, \Psi, \Sigma')$ such that N satisfies condition (1) and $\Sigma'(s_c)$ is a subsequence of $\Sigma(s_c)$.

- A domain description D is said to be **consistent** if it has a model.

- A domain description D entails a fact p (written as $D \models p$) iff p is *true* in all models of D.

Example: Reasoning about fluents

$$\left.\begin{array}{l}
Facts:\\
(p1)\ alive\ \textbf{at}\ \ s_0\\
(p2)\ dry\ \textbf{at}\ \ s_0\\
(p3)\ s_0\ \textbf{precedes}\ \ s_1\\
(p4)\ squirt\ \textbf{occurs\_at}\ \ s_0\\
(p5)\ shoot\ \textbf{occurs\_at}\ \ s_1\\
\\
Laws:\\
(p6)\ squirt\ \textbf{causes}\ \ \neg dry\\
(p7)\ shoot\ \textbf{causes}\ \ \neg alive
\end{array}\right\}D_2$$

1. $D_2 \models ((\neg dry \wedge alive)\ \textbf{at}\ s_1\mathbf{)}$

2. $D_2 \models ((\neg dry \wedge \neg alive)\ \textbf{at}\ s_c\mathbf{)}$

Example: Reasoning by cases

Shooting with two guns:

$$\left.\begin{array}{l}
Facts: \\
(q1)\ alive\ \mathbf{at}\ \ s_0 \\
(q2)\ (loaded_1\ \mathbf{at}\ \ s_0) \lor (loaded_2\ \mathbf{at}\ \ s_0) \\
(q3)\ [shoot_1, shoot_2]\ \mathbf{occurs\_at}\ \ s_0 \\
Laws: \\
(q4)\ shoot_1\ \mathbf{causes}\ \ \neg alive\ \mathbf{if}\ \ loaded_1 \\
(q5)\ shoot_2\ \mathbf{causes}\ \ \neg alive\ \mathbf{if}\ \ loaded_2
\end{array}\right\} D_3$$

$D_3 \models \neg alive\ \mathbf{at}\ \ s_c$

Example: Explaining observations

$$\left.\begin{array}{l}
Facts: \\
(p1)\ alive\ \ \mathbf{at}\ \ \ s_0 \\
(p2)\ dry\ \ \mathbf{at}\ \ \ s_0 \\
(p3)\ s_0\ \ \mathbf{precedes}\ \ \ s_1 \\
(p4)\ squirt\ \ \mathbf{occurs\_at}\ \ \ s_0 \\
(p5')\ \neg alive\ \ \mathbf{at}\ \ \ s_1 \\
\\
Laws: \\
(p6)\ squirt\ \ \mathbf{causes}\ \ \ \neg dry \\
(p7)\ shoot\ \ \mathbf{causes}\ \ \ \neg alive
\end{array}\right\} D_4$$

$$D_4 \models [squirt, shoot]\ \mathbf{occurs\_at}\ \ s_0$$

Hypothetical Reasoning

Hypothesis are expressions of the form

f **after** α **at** s (*)

read as "Assuming that the sequence α of actions occur starting at the situation s fluent f would be true in the resulting situation".

Let $M = (\sigma_0, \Psi, \Sigma)$ be a model of a domain description D.

A **hypothesis (*) is true** in M if f is true in the state $\Psi(\sigma_0, \Sigma(s) \circ \alpha)$.

Let H_1 and H_2 be two sets of hypothesis. H_1 **entails** H_2 in D if H_2 is true in every model of D in which H_1 is true.

Applications

Consider a reasoner, whose knowledge is specified by a domain description of $\mathbf{L}_0$ and whose behavior is defined by the following loop:

- **observe the world and add new information to** D
- **select a goal** G
- **find plan** $[a_1, \ldots, a_n]$ **to achieve** G
- **execute** a_1

A sequence α **of actions is a PLAN for achieving a goal** G **if for every fluent** $f \in G$

$\models_D (f$ **after** α **at** $s_c)$

Example

Jack needs to bring his packed suitcase to the airport. He has a car, knows how to drive it to the airport, etc. Consequently, his plan of packing a suitcase and driving to the airport is adequate. He packs the suitcase and observes that his car being hit. The car is no more, the previous plan is invalidated. New plan - rent a car and drive to the airport.

Problem: Construct a program simulating Jack's behavior.

Jack's trip to the airport

$$\left.\begin{array}{l}
initial\ facts:\\
(f1) home\ \textbf{at}\ \ s_0\\
(f2) \neg at\_airport\ \textbf{at}\ \ s_0\\
(f3) has\_car\ \textbf{at}\ \ s_0\\
Laws:\\
(l1) rent\ \textbf{causes}\ \ has\_car\\
(l2) hit\ \textbf{causes}\ \ \neg has\_car\\
(l3) drive\ \textbf{causes}\ \ at\_airport\ \textbf{if}\ \ has\_car\\
(l4) drive\ \textbf{causes}\ \ \neg home\ \textbf{if}\ \ has\_car\\
(l5) pack\ \textbf{causes}\ \ packed\ \textbf{if}\ \ home
\end{array}\right\} D_6$$

Goal: bring packed suitcase to the airport

Jack's trip continued

- planning:

  $? \models_{D_6} (packed \wedge at\_airport)$ **after** α **at** s_c

  $\alpha_0 = [pack, drive]$

- execution: $D_7 = D_6 \cup (pack$ **occurs_at** $s_0)$
- observation: $D_8 = D_7 \cup (hit$ **occurs_at** $s_1)$
- planning:

Does the old plan work?

$? \models_{D_8} (packed \wedge at\_airport)$ **after** α_0 **at** s_c

No. Find new plan.

$? \models_{D_6} (packed \wedge at\_airport)$ **after** α **at** s_c

$\alpha_1 = [rent, drive]$

Implementation

- Translate domain descriptions of L_0 into declarative logic programs.
- Prove soundness (completeness) w.r.t. entailment of L_0.
- Translate the declarative program into procedural program with a general purpose query-answering mechanism.
- Prove soundness (completeness) w.r.t. declarative program.
- Optimize.

Simple Domain Descriptions

We will say that a domain description D is **simple** if

1. D is consistent,
2. D has an explicit actual path,
3. all facts of D are atomic, and
4. D does not contain contradictory causal laws.

We'll use the following notation:

A	for actions
R	for lists of actions
F, G	for fluent literals
P	for lists of fluent literals

Let D be a simple domain description with explicit actual path $a_0, \ldots, a_{k-1}$.

The LP approximation Π_D of D consist of the rules:

Domain Dependent Axioms:

(AP) Description of Actual Path

$imm\_follows(s_1, s_0).$

$\ldots$

$imm\_follows(s_k, s_{k-1}).$

$occurs\_at(a_0, s_0).$

$\ldots$

$occurs\_at(a_{k-1}, s_{k-1}).$

(BC) Boundary Conditions

$true\_at(f, s_i) \in \Pi_D$ for each $(f$ **at** $s_i) \in D$

(CL) Causal Laws

$causes(a, f, p) \in \Pi_D$ for each $(a$ **causes** f **if** $p) \in D$

Domain Independent Axioms:

(EA) Effects of actions

$$
\begin{array}{lll}
true\_after(F, [\,], S) & :- & true\_at(F, S). \\
true\_after(F, [A|R], S) & :- & causes(A, F, P), \\
 & & all\_true\_after(P, R, S). \\
false\_after(F, R, S) & :- & contrary(F, G), \\
 & & true\_after(G, R, S). \\
all\_true\_after([\,], R, S). & & \\
all\_true\_after([F|P], R, S) & :- & true\_after(F, R, S), \\
 & & all\_true\_after(P, R, S). \\
one\_false\_after(P, R, S) & :- & member(F, P), \\
 & & false\_after(F, R, S).
\end{array}
$$

(FI) First Inertia Axiom

$$
\begin{aligned}
true\_after(F,[A|R],S) \ :- \ & fluent\_literal(F),\\
& true\_after(F,R,S),\\
& not\ ab(F,A,R,S).
\end{aligned}
$$

$$
\begin{aligned}
ab(F,A,R,S) \ :- \ & contrary(F,G),\\
& causes(A,G,P),\\
& not\ one\_false\_after(P,R,S).
\end{aligned}
$$

(SI) Second Inertia Axiom

$$
\begin{aligned}
true\_at(F,S2) \ :- \ & fluent\_literal(F),\\
& imm\_follows(S2,S1),\\
& occurs\_at(A1,S1),\\
& true\_after(F,[A1],S1).
\end{aligned}
$$

Soundness and Completeness of Π_D

Since $D \models f$ **at** s iff $D \models f$ **after** $[\,]$ **at** s, from now on we will limit our query language to formulas of the form f **after** α **at** s. For any query q of this form, by $\pi(q)$ we will denote $true\_after(f, \alpha, s)$.

We will say that a (declarative) program Π_D is *sound* w.r.t. D if Π_D is consistent and for any query q, if $\Pi_D \models \pi(q)$ then $D \models q$.

Theorem. For any simple domain description D, Π_D is sound w.r.t. D.

Example of incompleteness

$$\left.\begin{array}{l}
Facts: \\
(p1)\ alive\ \textbf{at}\ \ s_0 \\
(p3)\ s_0\ \textbf{precedes}\ \ s_1 \\
(p4)\ shoot\ \textbf{occurs\_at}\ \ s_0 \\
(p5')\ \neg alive\ \textbf{at}\ \ s_1 \\
\\
Laws: \\
(p7)\ shoot\ \textbf{causes}\ \ \neg alive\ \textbf{if}\ \ loaded
\end{array}\right\} D_5$$

It is easy to see that $D_5 \models (loaded\ \textbf{at}\ \ s_0)$ while Π_{D_5} does not.

Completeness Results

Theorem For any simple domain description D with complete initial state and any query q, $D \models q$ iff $\Pi_D \models q$.

Theorem. Let Π^*_D be obtained from D by adding

$true\_at(f, s_0)$ or $\neg true\_at(f, s_0)$.

for every fluent f. Then for every simple domain description D and any query q, $D \models q$ iff $\Pi^*_D \models q$.

Π_D as a Prolog program

Recall that Π_D is viewed as a schema describing the set of all ground instances of program with variables which respect types. All previous results are proven under this assumption. Syntactically Π_D can also be viewed as a logic program with variables.

Theorem. For any simple domain description D and any query q, Prolog interpreter applied to Π_D and q answers "yes" iff $\Pi_D \models q$.

Π_D can be used as a "test" part for "generate and test" planner.

Comments on the planner

Generate part is simplistic but Test part is not!

- Planner is build from specification to implementations with necessary correctness proofs being a part of programming process.
- Works even if information is incomplete
- Allows complex domains with appearing and disappearing objects, exogenous actions, etc.
- Will grow and improve with development of action theories and inference methods for logic programs and deductive databases

Generate part can be made more sophisticated by some top-down analysis and by incorporation of heuristics.

What Next? (Very short list)

- Inference methods for programs with multiple answer sets or/and disjunction or/and abduction to make the test part of the planner complete. (Clarity at this stage can still be more important than efficiency, but efficiency should be sufficient to run prototypes).
- Allow domains with more sophisticated ontologies (e.g. real numbers and continuous processes) and more expressive logic (e,g. allow normative causal statements such as "Shooting normally ends in death").
- Compute entailment in domain description (preferrably by LP methods).
- Keep comparing with other approaches.

Some references

Situation and Event Calculi:

J. McCarthy and P. Hayes,” Some philosophical problems from the standpoint of artificial intelligence”, ”Machine Intelligence”, vol 4, 1969

Kowalski and Sergot, “A logic-based calculus of events”, New Generation Computing, Vol 4, No 1, 1986, pp. 67-95

M. Gelfond, V. Lifschitz and A. Rabinov, ”What are the limitations of the situation calculus?”, Automated Reasoning: Essays in Honor of Woody Bledsoe, 1991, edited by R. Boyer, ”Kluwer Academic”.

R. Reiter, ”Proving Properties of States in the Situation Calculus”, Artificial Intelligence, 64:337-351, 1993

Pinto and Reiter, “Temporal reasoning in logic Programming: a case for the Situation Calculus”, Proc. of ICLP93

Miller and Shanahan, “Narratives in the situation calculus”, Journal of Logic and Computation, 4(5):513-530, 1994

Van Belleghem, K. and Denecker, M. and De Schreye, D., ”Combining Situation Calculus and Event Calculus”, ”Proc. of 1995 International Conference on Logic Programming”, pp. 83-97,1995.

Reasoning about Actions in Abductive Logic Programming

M. Shanahan,"Prediction is deduction but explanation is abduction", Proc. of IJCAI-89, 1055–1060.

Denecker, M. and De Schreye, D., Representing Incomplete Knowledge in Abductive Logic Programming", Journal of Logic and Computation, 1995, Vol 5, Num 5, pp 533-579.

Action Description Languages

M. Gelfond and V. Lifschitz, "Representing Actions and Change by Logic Programs", the Journal of Logic Programming,1993, vol 17, num 2,3,4, pp. 301-323.

N. McCain and H. Turner, “A causal theory of ramifications and qualifications”, In Proc. of IJCAI 95, pp. 1978-1984

E. Giunchiglia and V. Lifschitz, "Dependent Fluents", In Proc. of IJCAI95, pp. 1964-1969

C. Baral, M. Gelfond, A. Provetti, "Representing Actions: Laws, Observations and Hypotheses", to appear in the special issue of the Journal of logic Programming, 1997

C. Baral, M. Gelfond "Reasoning about effects of concurrent actions", to appear in the special issue of the Journal of logic Programming, 1997

List of active researchers and other related info can be found in

http://cs.utep.edu/actions/researchers.html