BUILDING A COMMONSENSE THEORY OF TRAVEL

This is a second attempt on the step-by-step explanation of the development of the commonsense theory of travel.

Michael Gelfond, Nov 20, 04

New Features:

1. Trips allow multiple visits to the same location.

2. New specification of a trip allows to simplify the original representation. Emergency stop is gone (now after an unexpected stop the trip goes to the next planed destination). The notion of emergency can be introduced if needed.

3. Instead of defeasible triggers I use description of planned actions together with general "intentions axioms" from our first paper. Now NAF only occurs in general axioms. I like this solution better. What do you think?

Modeling trips: names and positions.

The basic object of travel module T is that of a TRIP - a short journey over a set route.

We assume that names of trips and their possible stops are given by classes (unary relations)

$$trip(J) \qquad location(L)$$

At any given step the trip may be in transit (en_route) or at one of its possible stops. Its possible positions are given by a class

$$postion = location \cup \{en\_route\}$$

Modeling Trips: itineraries.

Description of a trip's planned route is given by a list of atoms:

$$stop(j, l_{i_0}, 0), \ldots, stop(j, l_{i_k}, k)$$

enumerating stops $l_{i_0}, \ldots, l_{i_k}$ of a trip j. Even though the list maybe incomplete the origin, l_{i_0}, and the destination, l_{i_k}, must be given.

Trip's Actions (depart and stop)

There are two main actions which can be "performed" by a trip j:

$$depart(j) \qquad\qquad stop(j, l)$$

"j departs its current location; j stops at l"

If j is a trip with the origin c_0, destination c_2, and the intermediate stop c_1 then departure, a_0, of j will normally trigger actions

$a_1 = stop(j, c_1)$, $a_2 = depart(j)$, $a_3 = stop(j, c_2)$.

The corresponding trajectory is

$\langle c_0, a_0, en\_route, a_1, c_1, a_2, en\_route, a_3, c_2 \rangle$

T_0 - the theory of trip's actions

Smodels notation is used to declare typed variables.

```
#domain location(L;L1;L2;L3).
#domain position(Pos;Pos1;Pos2;Pos3).
#domain trip(J;J1;J2;J3).

#domain action(A;A1;A2;A3).
#domain fluent(Fl;Fl1;Fl2;Fl3).
#domain step(S;S1;S2;S3).
#domain step(K;K1;K2;K3).
```

Fluents are inertial; $step$ **is the set of natural numbers from** 0 **to some** $n-1$**. We use** S **to denote steps of the domain histories;** K **will denote ordering of a trip's stops.**

The language of T_0

T_0 will be used in conjunction with descriptions of trips and their possible positions. T_0's actions and fluents are

```
action(depart(J)).
actor(depart(J),J).
action(stop(J,L)).
actor(stop(J,L),J).
fluent(at(J,Pos)).
fluent(last_planned_stop(J,K)).
```

The language of T_0

T_0 uses relations of LP based action theories:

$obs(Fl, S, 0/1),\ hpd(A, S),\ holds(Fl, S),\ occurs(A, S)$

The first two are used to record (always accurate) observations. The last two, abbreviated by h and o, denote tentative, defeasible conclusions of the agent.

Dynamic causal laws:

After the departure J is en_route:

```
h(at(J,en_route),S+1) :-
          o(depart(J),S).
```

Stops are successful:

```
h(at(J,L),S+1) :-
          o(stop(J,L),S).
```

Planned stops increase the counter:

```
h(last_planned_stop(J,K+1),S+1) :-
          h(last_planned_stop(J,K),S),
          stop(J,K+1,L),
          o(stop(J,L),S).
```

State Constraints

```
object(J).

#domain object(O;O1;O2;O3).


Position of an object is unique:

-h(at(O,Pos1),S) :-
          h(at(O,Pos2),S),
          neq(Pos1,Pos2).


Last stop is unique:

-h(last_planned_stop(J,K1),S) :-
            neq(K1,K2),
            h(last_planned_stop(J,K2),S).
```

Executability Conditions

```
Trip can't depart from en_route
and after the final arrival.
-o(depart(J,L),S) :-
          h(at(J,en_route),S).
-o(depart(J),S) :-
          num_of_planned_stops(J,K),
          h(last_planned_stop(J,K),S).

Trip stops when en_route:
-o(stop(J,L),S) :-
          -h(at(J,en_route),S).
Location of stop is unique:
-o(stop(J,L),S) :-
          o(stop(J,L1),S),
          neq(L,L1).
```

Trip's Intentions:

Trip intents to follow its route.

```
planned(stop(J,L),S) :-
          h(at(J,en_route),S),
          h(last_planned_stop(J,K),S),
          stop(J,L,K+1).
planned(depart(J),S+1) :-
          o(stop(J,L),S),
          num_of_planned_stops(J,K),
          -h(last_planned_stop(J,K),S+1).
```

Domain Independent Axioms

Axioms for Intentions:

Normally intentions are acted upon immediately:

```
o(A,S) :-
        planned(A,S),
        not -o(A,S).
```

Intended actions are postposned if necessary:

```
planned(A,S+1) :-
              planned(A,S),
              not o(A,S).
```

Domain Independent Axioms

Inertia:

Things tend to stay as they are.

```
h(Fl,S+1) :-
      h(Fl,S),
      not -h(Fl,S+1).

-h(Fl,S+1) :-
      -h(Fl,S),
      not h(Fl,S+1).
```

Domain Independent Axioms

Reality Check Axioms:

```
o(A,S) :-
        hpd(A,S).

:- obs(F,S,0),
   h(F,S).

:- obs(F,S,1),
   -h(F,S).
```

Axioms for Initial Situation

```
h(F,0) :-
       obs(F,0,1).
-h(F,S) :-
       obs(F,0,0).
num_of_planned_stops(J,K-1) :-
       K{stop(J,X,Y) : location(X) : step(Y)}K.
h(last_planned_stop(J,0),0).
```

Computing the trip's trajectories

T_0 can be viewed as a function which takes as an input $X = D \cup H$ where

(a) D is a collection of atoms defining trips, their positions and itineraries referred to as a trip domain;

(b) H is a history of the domain,

and returns the domain trajectory $T_0(X)$ extracted from the answer set W of $T_0 \cup X$.

Displaying the trajectories.

In the language of Smodels this can be done by the following display rules and directives.

```
at(O,Pos,S) :-
        h(at(O,Pos),S).
do(A,S) :-
        o(A,S).
hide.
show do(A,B), at(A,B,C).
```

Example: domain D

Consider two trips defined as:

```
trip(j1).          trip(j2).
```

with possible positions defined by the program:

```
city(rome).
city(baghdad).          location(X) :- city(X).
city(boston).
...
position(L).
position(en_route).
```

Note that the last two axioms are part of every travel domain.

Example: domain D

Domain D is obtained by extending the above axioms by statements:

```
stop(j1,boston,0).    stop(j2,baghdad,1).
```

and the itinerary

```
stop(j2,boston,0).
stop(j2,rome,1).
stop(j2,berlin,2).
stop(j2,baghdad,3).
```

for $j2$.

Example: histories H_1 and H_2

Let H_1 be

```
hpd(depart(J),0).     obs(at(J,boston),0,1).
```

H_2 be

```
hpd(depart(J),0).     h(at(J,boston),0,1).
hpd(stop(J,paris),1).
```

and $X_1 = D \cup H_1$ and $X_2 = D \cup H2$

Example: trajectory of j_1 in X_1

The trajectory of j_1 in X_1 is computed by setting n in the definition of *step* to 6 and finding the answer set of $T_0 \cup X_1$. It is

```
at(j1,boston,0)          do(depart(j1),0),
at(j1,en_route,1)        do(stop(j1,baghdad),1)
at(j1,baghdad,2)
```

Since X_1 contains neither the itinerary for j_1 nor information about its other stops we conclude that the trip goes directly to Baghdad.

Example: trajectory of j_2 in X_1

Similarly for j_2

`at(j2,boston,0)`	`do(depart(j2),0)`
`at(j2,en_route,1)`	`do(stop(j2,rome),1)`
`at(j2,rome,2)`	`do(depart(j2),2)`
`at(j2,en_route,3)`	`do(stop(j2,berlin),3)`
`at(j2,berlin,4)`	`do(depart(j2),4)`
`at(j2,en_route,5)`	`do(stop(j2,baghdad),5)`
`at(j2,baghdad,6)`	

j_2 goes to Baghdad making all the required intermediate stops.

Example: trajectory of j_1 in X_2

```
at(j1,boston,0)          do(depart(j1),0)
at(j1,en_route,1)        do(stop(j1,paris),1)
at(j1,paris,2)           do(depart(j1),2)
at(j1,en_route,3)        do(stop(j1,baghdad),3)
at(j1,baghdad,4)
```

As stated in its history the trip visits Paris, and then continues to its final destination, Baghdad.

Example: trajectory of j_2 in X_2

`at(j2,boston,0)`	`do(depart(j2),0)`
`at(j2,en_route,1)`	`do(stop(j2,paris),1)`
`at(j2,paris,2)`	`do(depart(j2),2)`
`at(j2,en_route,3)`	`do(stop(j2,rome),3)`
`at(j2,rome,4)`	`do(depart(j2),4)`
`at(j2,en_route,5)`	`do(stop(j2,berlin),5)`
`at(j2,berlin,6)`	`do(depart(j2),6)`
`at(j2,en_route,7)`	`do(stop(j2,baghdad),7)`
`at(j2,baghdad,8)`	

The trip makes an unplanned emergency stop in Paris, and then continues on its route.

Modeling the trip's participants

To model travel by individuals we expand T_0 by adding a new variable declaration

```
#domain person(P;P1;P2;P3)
```

additional actions and fluents

```
action(embark(P,J))      actor(embark(P,J),P)
action(disembark(P,J))   actor(disembark(P,J),P)
action(go_on(P,J))       actor(go_on(P,J),P)
fluent(participant(P,J))
fluent(at(P,Pos))
```

and additional objects

```
object(P)
```

Direct Effects

P joins the trip J:

```
h(participant(P,J),S+1) :-
                    o(embark(P,J),S).
```

P ends the trip J:

```
-h(participant(P,J),S+1) :-
            o(disembark(P,J),S).
```

Triggers

go_on is a sequence of two simpler actions:

```
Embark on the trip:
o(embark(P,J),S) :-
            o(go_on(P,J),S).

Disembark at arrival:
o(disembark(P,J),S) :-
          h(participant(P,J),S),
          num_of_planned_stops(J,K),
          h(last_planned_stop(J,K),S).
```

State Constraints

Participants share the current location of their trip

```
h(at(P,Pos),S) :-
            h(participant(P,J),S),
            h(at(J,Pos),S).
```

Executability Conditions

Can't embark (disembark) if already done so:

```
-o(embark(P,J),S) :-
                h(participant(P,J),S).
-o(disembark(P,J),S),
                -h(participant(P,J),S).
```

Need to be at the right place:

```
-o(embark(P,J),S) :-
                h(at(P,L),S),
                -h(at(J,L),S).
-o(embark(P,J),S),
                h(at(J,en_route),S).
-o(disembark(P,J),S),
                h(at(J,en_route),S).
```