Reasoning about intended actions

Chitta Baral
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85233, USA.
chitta@asu.edu

Michael Gelfond
Department of Computer Science
Texas Tech University
Lubbock, TX 79409, USA.
mgelfond@cs.ttu.edu

Abstract

In most research on reasoning about actions and reasoning about narratives one either reasons about hypothetical execution of actions, or about actions that actually occurred. In this paper we develop a high level language that allows the expression of intended or planned action sequences. Unlike observed action occurrences, planned or intended action occurrences may not actually take place. But often when they do not take place, they persist, and happen at an opportune future time. We give the syntax and semantics for expressing such intentions. We then give a logic programming axiomatization and show the correspondence between the semantics of a description in the high level language, and the answer sets of the corresponding logic programming axiomatization. We illustrate the application of our formalism with respect to reasoning about trips.

Introduction and Motivation

In reasoning about actions (for example, (Reiter 2001; Sandewall 1998) and reasoning about narratives we often reason about action sequences that are executed in a particular situation, or actions that happened at particular time points. Alternatively, there have been some work on reasoning about natural actions (Reiter 1996) and actions that are triggered. In this paper we consider *intended execution of actions* and formalize how to reason about such intentions.

To motivate this further, consider a narrative where an agent intended to execute action a at time point i. A commonsense reasoner looking back at this intention would conclude that the agent must have executed action a at time point i. To ground this example, consider that the wife of our reasoner says that she intends to leave work at 5 PM. At 6 PM the commonsense reasoner would conclude that his wife must have left at 5 PM. Now suppose the reasoner checks his email and finds an email from his wife saying that she has been held up in a meeting and later gets information that the meeting ended at 5:30. The reasoner would then conclude that his wife must have left at 5:30 PM. I.e., her intended action, since it became impossible at the initially intended time point, must have persisted and executed at the next time point when it became executable.

Now let us generalize this to a sequence of actions where an agent intends to execute a sequence $a_1, \ldots, a_n$ at time point i. Now what if it happens (the world evolved in such a way) that the executability condition of a_k is not true at the time point where a_{k-1} ended. Does this mean the agent abandoned his intention to execute $a_k, \ldots, a_n$? It seems to us that most agents, if they failed to execute their intended action a_k after the execution of a_{k-1}, would execute a_k in the next possible time point when it became executable. As before, let us consider a more grounded example. John is supposed to have taken a flight from A to B and then taken a connection from B to C. Suppose Peter finds out that John's flight from A to B was late. Once Peter knows when exactly John reached B, his reasoning would be that John would have taken the next flight from B to C. In other words, failure to go from B to C at a particular time point, does not mean that John would have abandoned his intention to go from B to C; rather most likely he would have just done it at the next possible time point. This actually happened to one of the authors. He correctly guessed that his wife would take the next flight (after missing a connection) and was able to meet her at the airport when the next flight arrived.

In most earlier work on reasoning about actions and narratives (for example, (Levesque *et al.* 1997)), if one or many of the actions in a given sequence $a_1, \ldots, a_n$ are not executable or otherwise prevented from execution then the reasoning process rigidly assumes that either the actions were not executed or considers the domain to be inconsistent. The formulation there is appropriate with respect to the assumptions in those languages. Here we consider the new notion of "intended (or planned) execution of actions", which needs a different formalization. In this we can take pointers from prior studies on intentions (Bratman 1990; Wooldridge 2000). In particular, intentions have been studied from the point of view of the design of rational agents (Wooldridge 2000), and they are one of the three main components of BDI (Belief-Desire-Intention) agents. In (Wooldridge 2000), various properties of 'intentions' of a rational agent is discussed. In particular the author says:

> *Summarizing, we can see that intentions play the following important roles in practical reasoning*
>
> • Intentions drive means-ends reasoning.
> *If I have formed an intention, then I will attempt to achieve the intention, ...*

• Intentions persist.
I will not usually give up on my intentions without good reason – they will persist, ...

...

In this paper we first present an action language that allows the expression of intentions. We then use AnsProlog (logic programming with answer set semantics) to implement reasoning with intentions. *The ability of AnsProlog to express defaults and normative reasoning, becomes a key tool in expressing the normative reasoning associated with characterizing intentions, in particular the statements: (i) normally intended actions take place, and (ii) normally intentions, that are not executed as intended, persist.*

Syntax and Semantics of the language

The signature of our language $\mathcal{ALI}$ contains two disjoint finite sets: A, a set of names for elementary actions (agent's and exogenous); and F, whose elements are referred to as *fluents* and used to denote dynamic properties of the domain[1]. By *fluent literals* we mean fluents and their negations (denoted by $\neg f$). The set of literals formed from a set $X \subseteq F$ of fluents will be denoted by $lit(X)$. A set $Y \subseteq lit(F)$ is called *complete* if for any $f \in F$, $f \in Y$ or $\neg f \in Y$; Y is called *consistent* if there is no f such that $f, \neg f \in Y$.

Actions are sets $\{a_1, \ldots, a_n\}$ of elementary actions. Intuitively, execution of an action $\{a_1, \ldots, a_k\}$ corresponds to the simultaneous execution of its components. Action sequences are constructed using $\langle . \rangle$ a la Prolog, i.e. we allow sequences $\langle \{a_1, a_2\}, \{a_3, a_4\} \rangle$, etc. We will frequently identify an action a with the sequence $\langle a \rangle$.

By a *transition diagram* over signature Σ we mean a directed graph T such that:

(a) the states of T are labeled by complete and consistent sets of fluent literals (corresponding to possible physical states of the domain) denoted by σ_is.

(b) the arcs of T are labeled by actions.

Paths of a transition diagram, which are of the form $\langle \sigma_1, a_1, \sigma_2, \ldots, a_{n-1}, \sigma_n \rangle$, are called *trajectories* of the domain.

Background: Representation of the transition diagram

In this section we briefly review the syntax of an action description language $\mathcal{AL}$ (Baral & Gelfond 2000) and its semantics that defines the transition diagram corresponding to a given action description in $\mathcal{AL}$.

An action description of $\mathcal{AL}$ is a collection of propositions of the form (1) $causes(a_e, l_0, [l_1, \ldots, l_n])$, (2) $caused(l_0, [l_1, \ldots, l_n])$, and (3) $impossible\_if(a_e, [l_1, \ldots, l_n])$;

where a_e is an elementary action and $l_0, \ldots, l_n$ are fluent literals. The first proposition says that, if the elementary action a_e were to be executed in a situation in which $l_1, \ldots, l_n$ hold, the fluent literal l_0 will be caused to hold in the resulting situation. Such propositions are called *dynamic causal laws*. The second proposition, called a *static causal law*, says that, in an arbitrary situation, the truth of fluent literals, $l_1, \ldots, l_n$ is sufficient to cause the truth of l_0. The last proposition says that action a_e cannot be performed in any situation in which $l_1, \ldots, l_n$ hold. (The one presented here is actually a simplification of $\mathcal{AL}$. Originally $impossible\_if$ took as argument an action rather than an elementary one. The restriction on a_e being elementary is not essential and can be lifted. We require it to simplify the presentation).

To define the transition diagram, T, given by an action description AD of $\mathcal{AL}$ we use the following terminology and notation. A set S of fluent literals is closed under a set Z of static causal laws if S includes the head, l_0, of every static causal law such that $\{l_1, \ldots, l_n\} \subseteq S$. The set $Cn_Z(S)$ of *consequences* of S under Z is the smallest set of fluent literals that contains S and is closed under Z. $E(a_e, \sigma)$ stands for the set of all fluent literals l_0 for which there is a dynamic causal law $causes(a_e, l_0, [l_1, \ldots, l_n])$ in AD such that $[l_1, \ldots, l_n] \subseteq \sigma$. $E(a, \sigma) = \bigcup_{a_e \in a} E(a_e, \sigma)$. The transition system $T = \langle \mathcal{S}, \mathcal{R} \rangle$ *described* by an action description AD is defined as follows:

1. $\mathcal{S}$ is the collection of all complete and consistent sets of fluent literals of Σ closed under the static laws of AD,
2. $\mathcal{R}$ is the set of all triples $\langle \sigma, a, \sigma' \rangle$ such that a is executable in σ (i.e., AD does not contain a proposition of the form $impossible\_if(a_e, [l_1, \ldots, l_n])$ such that $a_e \in a$, $\{l_1, \ldots, l_n\} \subseteq \sigma$) and σ' is the fixpoint of the equation

$$\sigma' = Cn_Z(E(a, \sigma) \cup (\sigma \cap \sigma')) \quad (1)$$

where Z is the set of all static causal laws of AD. The argument of Cn_Z in (1) is the union of the set $E(a, \sigma)$ of the "direct effects" of a with the set $\sigma \cap \sigma'$ of facts that are "preserved by inertia". The application of Cn_Z adds the "indirect effects" to this union.

We call an action description *deterministic* if for any state σ_0 and action a there is at most one such successor state σ_1.

Syntax of the rest of the language: Observations and intentions

As we mentioned earlier our focus is on the recorded history, including past intentions, and their characterization on how the world evolved. The recorded history is a collection of statements of the following forms:
(i) $intended(\alpha_1, i)$,
(ii) $happened(\alpha_2, i)$, and (iii) $observed(l, i)$.
where α's are action sequences, l is a fluent literal, and i is a time-step. We assume that the elementary actions of α_1 are not exogenous.

Intuitively, the statement $intended(\alpha_1, i)$ means that the agent intended to execute the action sequence α_1 at time point i. In the context of an agent architecture one can view this as that the agent made a plan (at time point i) to achieve its goal and the plan was to execute α_1. As mentioned earlier, it may so happen that the first action of α_1 may not be immediately executable at time point i, as things might have

[1] Our definitions could be easily generalized to domains with non-boolean fluents. However, the restriction to boolean fluents will simplify the presentation.

changed while the agent was making its plan. In that case the intuition is that the agent would execute it at the next possible time point. (The agent would most likely not go for making a new plan immediately as there is no guarantee that things would remain unchanged while he is making the new plan. But if α_1 does not become executable for a long time, then the agent may indeed look for alternatives.)

The intuitive meaning of the statements $happened(\alpha_2, i)$ and $observed(l, i)$ are that the sequence of actions α_2 was observed to have happened starting from time point i, and l was observed to be true at time point i respectively.

Semantics

For the formal characterization, since we adopt the usual meaning of $happened$ and $observed$, our main focus is the characterization of $intended$. In particular, our characterization formulates the following assumptions:

1. a reasoner executes an intended action the moment such execution becomes possible;
2. intending an execution of a sequence of actions $a_1, \ldots, a_n$ at time step i consists of intending the execution of a_1 at i followed by intending the execution of a_2 at the time step at which execution of a_1 is completed, and so on. (The intuition remains the same, if a_is are action sequences themselves.)
3. Intentions persist even if execution of an action at intended time-step proves to be impossible.

The following example illustrates the above assumptions.

Example 1 In accordance with these assumptions a history consisting of $intended(\langle a_1, a_2\rangle, 1)$ defines a collection of trajectories of the form:
$\langle \sigma_1, a_1, \sigma_2, a_2, \sigma_3\rangle$,
while a history consisting of $intended(\langle a_1, a_2\rangle, 1)$ and $happened(a_3, 2)$, where a_2 and a_3 can not be executed in parallel, defines a collection of trajectories of the form
$\langle \sigma_1, a_1, \sigma_2, a_3, \sigma_3, a_2, \sigma_4\rangle$. □

We now define when a trajectory is a model of a history. In this we assume that all actions that have occurred are either recorded by $happened$ facts, or are due to intentions.

Definition 1 Let $P = \langle \sigma_1, a_1, \sigma_2, \ldots \sigma_m, a_m, \sigma_{m+1}\rangle$ be a trajectory.

1. P is said to satisfy a statement $intended(a, i)$, where a is an action, if there is $j \geq i$ such that $a \subseteq a_j$ and for every $i \leq k < j$, a is not executable at σ_k (i.e., for some $a_e \in a$, we have $impossible\_if(a_e, [l_1, \ldots, l_n])$ in our action description such that $\{l_1, \ldots, l_n\} \subseteq \sigma_k$). We then say that j+1 is the point of a's completion, and we say that each element of a is supported at j.
2. P is said to satisfy a statement $intended(\alpha, i)$ where $\alpha = \langle a'_1, \ldots, a'_n\rangle$, and $n > 1$, if P satisfies $intended(a'_1, i)$ and $intended(\langle a'_2, \ldots, a'_n\rangle, j)$ where j is the point of a'_1's completion.
3. P is said to satisfy a statement $observed(f, i)$ if f is true in σ_i.
4. P is said to satisfy a statement $happened(\alpha, i)$, where $\alpha = \langle a'_1, \ldots, a'_n\rangle$, if for $1 \leq j \leq n$, $a'_j \subseteq a_{i+j-1}$. We then say that each element of a'_j is supported at i+j-1.
5. P is a *model* of H if it satisfies all the statements of H, and for $1 \leq i \leq m$, all elements of a_i are supported at i. □

Thus models of a history may have some empty actions in between. If a_i is empty then it means that no actions occurred at time point i.

Axiomatization of the semantics in AnsProlog

In this section we give an AnsProlog encoding that captures the semantics of the previous section. Initially, we assume that there is a set of rules which capture the transition diagram. With that assumption, our initial goal is to write the additional AnsProlog rules which when given facts about the history H, consisting of $happened$, $observed$ and $intended$ atoms, will enumerate trajectories (through its answer sets) that are models of H. This encoding of H consists of the representation of the $happened$, $intended$, and $observed$ facts as given below (denoted by $\alpha(H)$), and the rules itemized in 1, 2, and 3 below. The rules are denoted as Π_1.

Since $happened$ and $intended$ facts are about sequences of actions, we represent them in $\alpha(H)$ as follows. To encode $happened(\alpha, i)$, where $\alpha = \langle a_1, \ldots, a_n\rangle$, and $a_i = \{a_{i1}, \ldots, a_{ij_i}\}$, we write the facts:
$happened(s, i)$.
$seq(s, 1, a_1).\ in(a_{11}, a_1). \ldots in(a_{1,j_1}, a_1).$
...
$seq(s, n, a_n).\ in(a_{n1}, a_n). \ldots in(a_{n,j_n}, a_n).$
$intended(\alpha, i)$ is encoded similarly. $observed$ facts are encoded directly.

The collection of rules Π_1 that reasons about a given history consists of the following.

1. To account for $happened$ atoms we have the following rule:

```
occurs(A,I+J-1) :- happened(S,I),
                   seq(S,J,A'), in(A,A').
```

2. To account for $observed$ atoms we have the following rules:

```
holds(L,0) :- observed(L, 0).
:- not holds(L,T), observed(L,T).
```

3. To account for $intended$ atoms we need to add several rules as explained below.

(a) Unfolding intention of executing a sequence to planning the execution of actions in that sequence.

```
planned(A,I)   :- intended(S,I),
                  seq(S,1,A).
planned(B,K+1) :- intended(S,I),
                  I <= K,
                  seq(S,J,A),
                  occurs_set(A,K),
                  seq(S,J+1,B).
```

The first rule above encodes that an individual action A is planned for execution at time point I, if, A is the first

action of a sequence which is intended to be executed in time point I. The second rule encodes that an individual action B is planned for execution at time point K+1, if B is the J+1th action of a sequence intended to be executed at an earlier time point and the Jth action of that sequence is A which is executed at time point K.

(b) Planned actions occur unless they are prevented

```
occurs_set(A,I) :- planned(A,I),
                   not -occurs_set(A,I).
```

(c) If a planned action does not occur as planned then the plan persists.

```
planned(A,I+1) :- planned(A,I),
                  not occurs_set(A,I).
```

(d) If an action A occurs then all elementary actions in A occur.

```
occurs(B,I) :- occurs_set(A,I),
               in(B,A).
```

(e) If an elementary action B does not occur then all actions containing B do not occur.

```
-occurs_set(A,I) :- -occurs(B,I),
                    in(B,A).
```

Example 2 We now illustrate Smodels (Niemelä & Simons 1997) encoding of the above with respect to the second part of Example 1. Since that example deals with actions that are singletons, we simplify the code a bit.

```
action(a1;a2;a3).  time(1..3).
intended(s,1).     seq(s,1,a1).
seq(s,2,a2).       happened(a3,2).

occurs(A,I) :- happened(A,I).

-occurs(B,I) :- action(A), action(B),
                time(I),occurs(A,I),A !=B.

planned(A,I) :- intended(S,I), seq(S,1,A).

planned(B,K+1) :- intended(S,I),seq(S,J,A),
                  occurs(A,K),time(K),
                  seq(S,J+1,B).

occurs(A,I) :- action(A), planned(A,I),
               time(I), not -occurs(A,I).

planned(A,I+1) :- action(A), time(I),
                  planned(A,I),
                  not occurs(A,I).
```

As expected, the above program has a single answer set which contains:

```
planned(a1,1) planned(a2,2) planned(a2,3)
 occurs(a1,1) occurs(a3,2) occurs(a2,3)
```

Translation of the action description

So far we assumed the existence of an AnsProlog encoding of the action description part. To precisely relate the semantics of $\mathcal{ALI}$ with an AnsProlog encoding we now present the encoding of the action description part as given in (Balduccini & Gelfond 2003).

We start with the encoding of the static and dynamic causal laws and the impossibility conditions. This encoding is done via a mapping α, from action descriptions of $\mathcal{AL}$ into programs of AnsProlog, defined as follows:

1. $\alpha(causes(a, l_0, [l_1 \dots l_m]))$ is the collection of atoms

   $d\_law(d), head(d, l_0), action(d, a),$
   $prec(d, 1, l_1), \dots, prec(d, m, l_m), prec(d, m+1, nil).$

   Here and below d will refer to the name of the corresponding law. Statement $prec(d, i, l_i)$, with $1 \leq i \leq m$, says that l_i is the i'th precondition of the law d; $prec(d, m+1, nil)$ indicates that the law has exactly m preconditions. This encoding of preconditions has a purely technical advantage. It will allow us to concisely express the statements of the form '*All preconditions of a law d are satisfied at moment T*'. (See rules (3-5) in the program Π_2 below.)

2. $\alpha(caused(l_0, [l_1 \dots l_m]))$ is the collection of atoms

   $s\_law(d), head(d, l_0),$
   $prec(d, 1, l_1), \dots, prec(d, m, l_m), prec(d, m+1, nil).$

3. $\alpha(impossible\_if(a, [l_1 \dots l_m]))$ is the rule

   $$-occurs(a, T) \leftarrow holds(l_1, T), \dots, holds(l_n, T).$$

   where $occurs(a, t)$ stands for '*elementary action a occurred at time t*'.

By $\alpha(AD)$ we denote the result of applying α to the laws of the action description AD. Finally, for any history, H, of S

$$\alpha(AD, H) = \Pi_1 \cup \alpha(H) \cup \Pi_2 \cup \alpha(AD)$$

where Π_2 is defined as follows:

$$\Pi_2 \begin{cases} 1.\ holds(L, T') \leftarrow d\_law(D), head(D, L), action(D, A), occurs(A, T), prec\_h(D, T). \\ 2.\ holds(L, T) \leftarrow s\_law(D), head(D, L), prec\_h(D, T). \\ 3.\ all\_h(D, N, T) \leftarrow prec(D, N, nil). \\ 4.\ all\_h(D, N, T) \leftarrow prec(D, N, P), holds(P, T), all\_h(D, N', T). \\ 5.\ prec\_h(D, T) \leftarrow all\_h(D, 1, T). \\ 6.\ holds(L, T') \leftarrow holds(L, T), not\ holds(\overline{L}, T'). \\ 7. \leftarrow holds(L, T), holds(\overline{L}, T). \end{cases}$$

Here D, A, L are variables for the names of laws, actions, and fluent literals respectively, T, T' denote consecutive time points, and N, N' are variables for consecutive integers. (To run this program under Smodels we need to either define the above types or add the corresponding typing predicates in the bodies of some rules of Π_2.)

Relation $prec\_h(d, t)$, defined by the rule (5) of Π_2, says that all the preconditions of law d are satisfied at moment t. This relation is defined via an auxiliary relation $all\_h(d, i, t)$

(rules (3), (4)), which holds if the preconditions $l_i, \ldots, l_m$ of d are satisfied at moment t. (Here $l_1, \ldots, l_m$ stand for the ordering of preconditions of d used by the mapping α.) Rules (1),(2) of Π_2 describe the effects of causal laws and constraints of AD. Rule (6) is the inertia axiom (McCarthy & Hayes 1969), and rule (7) rules out inconsistent states.

The following terminology will be useful for describing the relationship between answer sets of $\alpha(AD, H)$ and models of H.

Definition 2 *Let AD be an action description, and A be a set of literals over $lit(\alpha(AD, H))$. We say that A* defines *the sequence*

$$\langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$$

if $\sigma_k = \{l \mid holds(l, k) \in A\}$ and $a_k = \{a \mid occurs(a, k) \in A\}$.

The following theorem establishes the relationship between action domains and histories in $\mathcal{ALI}$ and their encoding in AnsProlog.

Theorem 1 *If the initial situation of H is* complete *(i.e. for any fluent f of AD, H contains $obs(f, 0)$ or $obs(\neg f, 0)$), and the action sequences in the atoms of H do not have repeated actions then M is a model of H iff M is defined by an answer set of $\alpha(AD, H)$.*

We now elaborate Example 2 by adding information about the actions, their executability, and their impact on the states. We also move the starting time point to 0 to make it more interesting.

Example 3 Lets assume that we have a fluent f which is initially true. We have three actions a_1, a_2 and a_3. a_1 is executable in all situations and causes $\neg f$. a_3 is executable in all situations and causes f. a_2 is executable in situations where f is true and causes $\neg f$. Now suppose a_3 has been observed to occur at time point 2, and the agent intended to execute $\langle a_1, a_2 \rangle$ at time point 0.

In that case a_1 must have been executed in time point 0. But a_2 could not have been executed at time point 1 because at time point 1, $\neg f$ would have been true making a_2 inexecutable. The action a_2 could not have been executable at time point 2 because at time point 2 a_3 occurred and two actions can not happen at the same time. Now, at time point 3, the executability conditions of a_3 was satisfied, and no other action is observed to have occurred at that time, and hence a_2 must have occurred at time point 3.

We now illustrate how an AnsProlog encoding based on the previous sections does the same reasoning. Since this example does not have static causal laws, and only deals with singleton actions, we simplify the code a bit.

```
fluent(f).
literal(F)      :- fluent(F).
literal(neg(F)) :- fluent(F).
action(a1;a2;a3).   time(0..4).
intended(s,0).  seq(s,1,a1).  seq(s,2,a2).
happened(a3,2).

occurs(A,I)  :- happened(A,I).

-occurs(B,I) :- action(A), action(B),
              time(I), occurs(A,I), A !=B.
planned(A,I) :- intended(S,I), seq(S,1,A).
planned(B,K+1) :- intended(S,I),
                  seq(S,J,A), occurs(A,K),
                  time(K), seq(S,J+1,B).
occurs(A,I) :- action(A), time(I),
               planned(A,I),
               not -occurs(A,I).
planned(A,I+1) :- action(A), time(I),
                  planned(A,I),
                  not occurs(A,I).
holds(f,0).

-holds_set(C,T) :- in(F,C), literal(F),
                   set(C), time(T),
                   not holds(F,T).
holds_set(C,T)  :- set(C), time(T),
                   not -holds_set(C,T).

holds(F,T+1) :- causes(A,F,C), literal(F),
                set(C), time(T),
                occurs(A,T),holds_set(C,T).
holds(F,T+1) :- holds(F,T), fluent(F),
                time(T), not -holds(F,T+1).
-holds(F,T+1) :- -holds(F,T), fluent(F),
                 time(T), not holds(F,T+1).

-holds(F,T) :-  fluent(F), time(T),
                holds(neg(F),T).
holds(neg(F),T) :-  fluent(F), time(T),
                    -holds(F,T).

causes(a1, neg(f), empty). set(empty).
causes(a3, f,empty).
causes(a2, neg(f), empty).

-occurs(a2,T) :- time(T), holds(neg(f),T).
```

As expected, the above program has a single answer which contains the following:

```
occurs(a1,0) occurs(a3,2) occurs(a2,3)
planned(a1,0) planned(a2,1)
planned(a2,2) planned(a2,3)
holds(f,0) -holds(f,1) -holds(f,2)
holds(f,3) -holds(f,4) -holds(f,5)
```

Allowing repeated actions

In the previous encodings we assumed that sequences of intended actions do not have the same action repeated. To remove this assumption, the following changes in the encoding suffices.

- Replace 3(a) by

```
planned(S,1,I) :- intended(S,I).
planned(S,J+1,K+1) :- intended(S,I),
                      occurs(S,J,K).
```

- Replace 3(b) by

```
occurs(S,J,K) :- planned(S,J,K),
                 not -occurs(S,J,K).
```

- Replace 3(c) by

```
planned(S,J,I+1) :- planned(S,J,I),
                    not occurs(S,J,I).
```

- Replace 3(d) by

```
occurs(B,I) :- occurs(S,J,I),
               seq(S,J,A), in(B,A).
```

- Replace 3(e) by

```
-occurs(S,J,I) :- -occurs(B,I), in(B,A),
                  seq(S,J,A).
```

An application: reasoning about trips

We came across the issue of reasoning about intentions when we were trying to develop a representation and reasoning module to reason about trips. We now briefly mention some of the aspects of modelling trips and its relationship with reasoning about intentions.

A trip is an activity with many participants who join the trip and may drop out at different points of the trip. The trip has a sequence of planned (or intended) stops, and the same location may appear multiple times in the sequence as some locations are hubs. The first stop of the trip is referred to as its origin, and the last stop is referred to as its destination. The trip may use many different vehicle types for its different legs. At any point the status of a trip may be in transit or in one of its possible stops (in our case - cities). The various actions associate with a trip include: a person embarking on the trip, a person dropping out (or disembarking) from the trip, depart (from a stop), and stop (in a stop). To embark on the trip a person may need to have some travel documents. Things packed in various containers can make the trip more pleasant, etc.

Our goal was to develop a reasoning and representation system which encodes the above general knowledge about trips and together with additional facts about the trips can answer various relevant questions. Following is a sample of questions which our system, built using the various encodings mentioned in this paper, answers correctly. (Our encodings of many of these and similar questions are available at http://www.public.asu.edu/∼cbaral/aquaint04-06/travel-module/.)

- j1 is a trip which starts in Boston in day 1, supposed to go to Baghdad on day 3, leave Baghdad on day 5 and come back to Boston on day 6. The plane broke down in Baghdad for 2 days on the day it was supposed to leave. When did the plane reach Boston? (Answer: day 8).
- John took the plane from Paris to Baghdad. He planned to meet his friend Mike there. Did John meet Mike? (Answer: yes.)
- John joined a plane trip which was scheduled to go from Paris to Rome to Baghdad. John was arrested in Rome. Where is John? (Answer: Rome.) Where is the plane? (Answer: Baghdad.)

Conclusion

Intentions have been discussed and properties of intentions have been formalized using modal logic in the past (Bratman 1990; Wooldridge 2000). But this is perhaps the first time intentions about actions has been *formalized and declaratively implemented* together with reasoning about actions and narratives. In this paper we not only give a formal characterization of intended actions but also give a provable correct implementation of it using AnsProlog. Our implementation is part of a bigger project involving representation and reasoning about trips.

Although in this paper we consider intentions of action sequences, this can be generalized to more expressive execution structures, such as Golog programs (Levesque *et al*. 1997) by combining the encodings in this paper together with the encodings in (Son, Baral, & McIlraith 2001), with minor changes.

Acknowledgements

We would like to than the anonymous reviewers for their insightful comments. This work was supported by NSF grants 0070463 and 0412000 and an ARDA contract.

References

Balduccini, M., and Gelfond, M. 2003. Diagnostic reasoning with a-prolog. *Journal of Theory and Practice of Logic Programming (TPLP)* 3(4-5):425–461.

Baral, C., and Gelfond, M. 2000. Reasoning agents in dynamic domains. In Minker, J., ed., *Logic Based AI*. Kluwer.

Bratman, M. 1990. What is intention? In Cohen, P.; Morgan, J.; and Pollack, M., eds., *Intentions in communication*. MIT Press. 15–32.

Levesque, H.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1-3):59–84.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, volume 4. Edinburgh University Press. 463–502.

Niemelä, I., and Simons, P. 1997. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. *Proc. 4th LPNMR*, 420–429.

Reiter, R. 1996. Natural actions, concurrency and continuous time in the situation calculus. In Aiello, L.; Doyle, J.; and Shapiro, S., eds., *KR 96*, 2–13.

Reiter, R. 2001. *Knowledge in action: logical foundation for describing and implementing dynamical systems*. MIT press.

Sandewall, E. 1998. Special issue. *Electronic Transactions on Artificial Intelligence* 2(3-4):159–330. http://www.ep.liu.se/ej/etai/.

Son, T.; Baral, C.; and McIlraith, S. 2001. Planning with different forms of domain dependent control knowledge – an answer set programming approach. In *Proc. of LPNMR'01*, 226–239.

Wooldridge, M. 2000. *Reasoning about Rational Agents*. MIT Press.