

# Activity Recognition with Intended Actions, Answer Set Programming Approach

Alfredo Gabaldon

NICTA & UNSW

Sydney, Australia

alfredo@cse.unsw.edu.au

## Abstract

We consider an activity recognition problem as follows. We are given a description of the action capabilities of an agent being observed. This description includes a description of the preconditions and effects of atomic actions the agent may execute and a description of activities (sequences of actions). Given this description and a set of propositions about action occurrences and intended actions, called *history*, the problem is to determine what has already happened, what the intentions of the agent are, and what may happen as a result of the agent acting on those intentions. Our formalization is based on a recent approach to reasoning about intended actions in (*A*-like) action languages with answer set programming implementations.

## Introduction

We consider the following problem: given a partial record of what an agent being observed is doing, including a) intended actions, b) action executions, c) fluent values, all at various time points, determine a complete picture of what the agent has done, intends and may do in the future. This problem is what we will refer to as *activity recognition*, and our approach to it is based on logical reasoning about the observations with respect to a background knowledge base that includes a formal action theory representing how the world evolves as the agent executes actions, knowledge about non-elementary actions, called *activities*, that the agent might be executing, and a theory of intended actions.

Our approach to activity recognition is novel in that it is based on a formal theory of actions and reasoning with an explicit notion of intended actions. We discuss related work before Conclusions.

We will use a modified and extended version of the cooking example from (Kautz and Allen 1996) throughout the paper.

**Example 1** The observed agent in this domain is capable of executing various meal preparation activities: it can *cook chicken marinara* which consists in *making marinara sauce* and putting it together with chicken by *mixing chicken marinara*. It can *cook fettuccini alfredo* by *making fettuccini*,

*making alfredo sauce* and putting it together by *mixing fettuccini alfredo*. It can also *cook fettuccini marinara*, *cook spaghetti carbonara* and *cook chicken primavera* using similar steps. Then, if the agent declares that he intends to *make fettuccini* at time 1, and we observe that *mix chicken marinara* occurred at time 3, then the possible conclusions are that the agent intends to cook two dishes: one of the fettuccini dishes and chicken marinara. In the case that he is making fettuccini marinara, it is possible that he makes marinara sauce once, for both dishes, i.e. the two cooking activities share an action.

## Reasoning about Intended Actions

The action description language *ACT*, introduced in (Baral and Gelfond 2005), extends similar action languages, in particular *AL* (Baral and Gelfond 2000), with the capability of reasoning about intended actions. In this section we give an overview of this language.

Like other *A*-like languages, a domain description in *ACT* includes a set of *dynamic causal laws*, *static causal laws* and executability propositions. A set of such statements, called *action description*, describes a transition system which models how the world moves from one state to another as actions are executed. A separate set of constructs in the language is used to capture a *history*: a set of statements about observed values of fluents and occurrences of actions at specified time points. Given a domain description and a possibly incomplete history, the reasoning task is then to determine a complete trajectory that the world may have followed and that is compatible with the history. Additionally, the language *ACT* allows for reasoning about intended actions, thus it includes a construct for specifying in a history that at a given time the agent intends to execute a given action. The underlying principle incorporated in *ACT* states that *normally, unfulfilled intentions persist*, meaning that if the agent is not able to execute an intended action at a specified time (e.g. because the action was not executable at that time) then the intention persists until the agent successfully executes the action. The formal syntax and semantics of this language is as follows.

The signature of *ACT* consists of two disjoint, finite sets: a set of elementary action names *A*, and a set of symbols *F*, called *fluents*, which represent properties of the domain that change when actions are executed. A *fluent literal* is a

fluent  $f$  or its negation, denoted by  $\neg f$ . A set of literals  $Y$  is called *complete* if for every  $f \in F$ ,  $f \in Y$  or  $\neg f \in Y$ , and  $Y$  is called *consistent* if there is no  $f$  s.t.  $f, \neg f \in Y$ . A *state* is a complete and consistent set of fluent literals and represents one possible state of the domain. An *action* is a set  $\{a_1, \dots, a_n\}$  of elementary actions representing their simultaneous execution. Sequences of actions are lists of actions separated by commas and enclosed by  $\langle \cdot \rangle$ .

Given a signature  $\Sigma = (A, F)$ , a *transition diagram* over  $\Sigma$  is defined as a directed graph  $T$  where:

- the states of  $T$  are the states of  $\Sigma$ , which are denoted by  $\sigma_i$ 's;
- the arcs of  $T$  are labeled by actions of  $\Sigma$ .

A path  $\langle \sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n \rangle$  of a transition diagram is called a *trajectory* of the domain.

As mentioned above, an action description in  $\mathcal{AL}\mathcal{I}$  consists of a set of statements understood as describing a transition diagram. These statements are of the following three forms ( $a_e$  denotes an elementary action and the  $l_i$  denote fluent literals):

1. *dynamic causal laws*:  $causes(a_e, l_0, [l_1, \dots, l_n])$ , stating that executing  $a_e$  in a state where  $l_1, \dots, l_n$  hold causes  $l_0$  to be true in the resulting state;
2. *static causal laws*:  $caused(l_0, [l_1, \dots, l_n])$ , stating that  $l_0$  is caused to hold in every state where  $l_1, \dots, l_n$  hold;
3. *executability propositions*<sup>1</sup>:  $impossible\_if(a_e, [l_1, \dots, l_n])$ , stating that  $a_e$  cannot be executed in a state where  $l_1, \dots, l_n$  hold.

The definition of the transition diagram specified by an action description requires the following notions and notation: An action  $a$  is executable in a state  $\sigma$  if there is no proposition  $impossible\_if(a_e, [l_1, \dots, l_n])$  s.t.  $a_e \in a$  and  $\{l_1, \dots, l_n\} \subseteq \sigma$ . 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 s.t.  $\{l_1, \dots, l_n\} \subseteq S$ . 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  $Z$ . The notation  $E(a_e, \sigma)$  is used to denote the set of all literals  $l_0$  s.t. there is a dynamic causal law  $causes(a_e, l_0, [l_1, \dots, l_n])$  and  $\{l_1, \dots, l_n\} \subseteq \sigma$ . Moreover,  $E(a, \sigma) = \bigcup_{a_e \in a} E(a_e, \sigma)$ .

An action description specifies a transition diagram that satisfies certain properties. One is that all the states of the transition diagram must satisfy the static causal laws. Second, if there is a transition from  $\sigma$  to  $\sigma'$  labeled by  $a$ , then  $a$  must be executable in  $\sigma$ . Furthermore,  $\sigma'$  must include the direct effects  $E(a, \sigma)$  of  $a$ , the indirect effects that follow from the static causal laws and it must contain literals that are otherwise not affected by  $a$  but are preserved by the common sense law of inertia.

Formally, the transition system specified by an action description  $AD$  is defined as follows.

<sup>1</sup>This name is normally used for propositions declaring when an action is executable, not impossible. But since they are used with a similar purpose, we will call these the same.

**Definition 1** An action description  $AD$  with signature  $\Sigma$  describes the transition system  $T = (S, R)$  where:

1.  $S$  is the set of all the states of  $\Sigma$  that are closed under the static laws of  $AD$ ;
2.  $R$  is the set of all triples  $\langle \sigma, a, \sigma' \rangle$  s.t.  $a$  is executable in  $\sigma$  and  $\sigma'$  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 the static causal laws in  $AD$ .

For activity recognition, we are interested in determining what an agent is doing given an action description and a history. The propositions used in  $\mathcal{AL}\mathcal{I}$  to describe the history have the following form ( $\alpha$  denotes an action sequence,  $i$  a time point, and  $l$  a fluent literal):

1.  $intended(\alpha, i)$ : meaning that the agent intends to execute  $\alpha$  at time point  $i$ ;
2.  $happened(\alpha, i)$ : meaning that  $\alpha$  happened starting at time point  $i$ ;
3.  $observed(l, i)$ : meaning that  $l$  was observed to hold at time point  $i$ .

A *history* is then a set of propositions of the above forms. The semantics of  $happened$  and  $observed$  is defined in the usual way for  $\mathcal{A}$ -like languages (see Definition 2 below). The semantics of  $intended$  as defined in (Baral and Gelfond 2005) is based on the following assumptions:

1. once an agent establishes the intention to execute an action, it does so as soon as the action is executable;
2. for an intended sequence  $\langle a_1, \dots, a_n \rangle$ ,  $a_1$  is intended first and each  $a_{i+1}$  is subsequently intended only after  $a_i$  executes;
3. if an intended action is not executable, the intention to execute it persists until it becomes possible to execute it.

A history is interpreted by trajectories of the background transition system. The following definition describes when a trajectory is a model of a history.

**Definition 2** Let  $AD$  be an action description,  $P = \langle \sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n \rangle$  be a trajectory,  $H$  be a history and  $1 \leq i \leq n$ .

1.  $P$  satisfies  $observed(l, i)$  if  $l$  is true in  $\sigma_i$  ( $l \in \sigma_i$ ). Similarly for initial state statements,  $P$  satisfies  $observed(l, 0)$  if  $l$  is true in  $\sigma_0$ .
2.  $P$  satisfies  $happened(\langle a'_1, \dots, a'_n \rangle, i)$  if for all  $1 \leq j \leq n$ ,  $a'_j \subseteq a_{i+j-1}$ . In this case, each element of  $a'_j$  is said to be *supported* at  $i + j - 1$ .
3.  $P$  satisfies  $intended(a, i)$  if
  - (a)  $a \subseteq a_i$ ; or
  - (b)  $a$  is not executable in  $\sigma_{i-1}$  (i.e. there is a proposition  $impossible\_if(a_e, \vec{l}) \in AD$  s.t.  $a_e \in a$  and  $\vec{l} \subseteq \sigma_{i-1}$ ,  $i < n$  and  $P$  satisfies  $intended(a, i + 1)$ ).

If  $a \subseteq a_i$ , we say that  $a$  *ends* at  $i+1$  and that each element of  $a$  is *supported* at  $i$ .

4.  $P$  satisfies  $intended(\alpha, i)$  where  $\alpha = \langle a'_1, \dots, a'_m \rangle$  and  $m > 1$ , if

- (a)  $P$  satisfies  $intended(a'_1, i)$  and
- (b)  $P$  satisfies  $intended(\langle a'_2, \dots, a'_m \rangle, j)$  where  $a'_1$  ends at  $j$  in  $P$ .

We say that  $\alpha$  ends at  $k$  in  $P$ , if  $a'_m$  ends at  $k$  in  $P$ .

5.  $P$  is a model of a history  $H$  if it satisfies all the statements of  $H$  and, for all  $1 \leq i \leq n$ , all the elements of  $a_i$  are supported at  $i$  in  $P$ .

Notice that the definitions do not rule out empty actions (i.e. nothing occurs) from appearing in a history.

## Activities

Technically, the problem of activity recognition as described above can be cast as the problem of finding models of the recorded history, after taking care of a few issues. Activity recognition differs from reasoning about histories in  $\mathcal{AL}\mathcal{I}$  in several respects. First, activity recognition is done from the perspective of an external observer of the agent that is executing the actions. Second, in addition to the background theory of actions and the statements in the history, the reasoner has additional knowledge in the form of a set of activities that the agent being observed may do and information about which actions are “purposeful.”

### Named Activities

Our approach to activity recognition, like other approaches, is based on the availability of a set background activities that the observed agent may do. These then serve as a hypothesis space to the recognition system. In our case these activities will be represented as pairs  $(s, \alpha)$  where  $s$  is the name of the activity and  $\alpha$  a sequence of actions (including other activities). Thus we extend the signature of the language with an additional set  $C$  of activity names. An action description will contain a set of pairs  $(s, \alpha)$  with at most one pair for each  $s \in C$ . We will often use  $s$  to refer to an activity  $(s, \alpha)$ . Sequences  $\alpha$  in named activities are assumed not to repeat actions.

In the cooking domain example, we have activities

$(ccm, \langle mk\_marinara, mix\_chicken\_marinara \rangle)$

$(cfm, \langle mk\_fettuccini, mk\_marinara, mix\_fettuccini\_marinara \rangle)$

$(cfa, \langle mk\_fettuccini, mk\_alfredo, mix\_fettuccini\_alfredo \rangle)$

and so on.

We assume that only actions can be observed to occur and thus will not use activity names in *happened* statements. On the other hand, we do allow activity names in *intended* statements as part of the history, as we consider the possibility that the observed agent declares its intentions or that the activity recognizer is otherwise informed of those intentions.

## Purposeful Actions

By purposeful actions we mean actions whose execution in isolation, as opposed to as part of a more complex activity, is considered reasonable. For example, one may consider the action of taking a bus as not purposeful since normally a person does not take a bus for the sake of taking a bus, but does so as part of a more complex activity such as commuting to work. Here we treat purposefulness as a fixed (non fluent) property of activities. A more elaborate treatment of purposefulness intuitively seems to require this notion to be context dependent and to be captured as a default. We will consider this more general notion of purposefulness in future work.

In our example, we declare the action of *mk\_marinara* sauce as not purposeful. The activity of cooking *fettuccini marinara*, *cfm* above, on the other hand, is considered purposeful since it is a complete meal cooking activity. Purposefulness is subjective, one can easily elaborate the cooking scenario by adding an activity for “having a meal” in the context of which cooking a meal may no longer be considered purposeful.

Purposeful actions are simply declared to be so by means of statements of the form *purposeful(c)*, where  $c$  is an action or an activity. Actions not declared to be purposeful are assumed not to be purposeful. In the next section we formally describe how knowledge about the purposefulness of actions and activities influences reasoning about intended actions.

## Activity Recognition

In our formalization of activity recognition we do not allow a history to state that a named activity happened, only actions can be observed. We moreover assume that all the observed actions were intentional.

The definition of satisfaction of history statements *observed(l, i)*, *happened(a, i)* and *intended(a, i)* by a trajectory is as before. While activities cannot be observed to occur, we allow that the reasoner may be informed that the observed agent intends to execute a named activity. This means that we must extend histories by allowing statements *intended(s, i)* where  $s$  is an activity name, and extend the definition of satisfaction of such statements by a trajectory  $P$ . The definition is as follows.

**Definition 3** A trajectory  $P$  satisfies a statement *intended(s, i)*, where  $(s, \alpha)$  is a named activity, if  $P$  satisfies *intended( $\alpha, i$ )*.

We will assume that any intended sequence in the history is named so that *intended(s, i)* is used instead of *intended( $\alpha, i$ )* for an activity  $(s, \alpha)$ . From now on we will use *actions* to refer to both non-elementary actions (sets of elementary actions) and named activities

Before we introduce models of a history, we need to introduce some terminology. We say that an action (including named activities)  $c$  starts at  $i$  in a trajectory  $P$  if  $P$  satisfies *intended(c, i)* but does not satisfy *intended(c, i - 1)*, i.e. it is said to start when it becomes intended. A named activity  $(s, \langle a_1, \dots, a_n \rangle)$  ends at  $i$  in  $P$  if  $a_n$  ends at  $i$  in  $P$  (as in

Definition 2).<sup>2</sup> Furthermore, an action  $c$  (including named activities) is said to be *in progress* at  $k$  in  $P$  if  $c$  starts at  $i$  and ends at  $j$  in  $P$  and  $i \leq k < j$ . Henceforth we will omit a reference to  $P$  when clear from context and say  $c$  starts at  $i$ ,  $c$  ends at  $j$ , etc.

Next we define the key notion of *justified actions*. This notion captures the intuition that if an action that is not purposeful is stated to have occurred or to be intended at time point  $i$ , then it must be the case that a purposeful activity is in progress at the same time  $i$  and the action is part of it. The following definition formally describes when an action is justified.

**Definition 4** Let  $AD$  be an action description,  $P$  be a trajectory and  $c$  be an action.

1.  $c$  is *justified by  $c$*  (self-justified) at  $i$  if  $purposeful(c) \in AD$ ;
2.  $c$  is *justified by  $s$*  at  $i$  if the following conditions are satisfied:
  - (a)  $(s, \alpha)$  is a named activity in  $AD$ ,
  - (b)  $c$  appears in  $\alpha$ ,
  - (c)  $s$  is in progress at  $i$ ,
  - (d)  $s$  does not justify  $c$  at an earlier time point in its current execution, that is, if  $l$  is the latest start time of  $s$  such that  $l < i$ , then  $s$  does not justify  $c$  at  $k$  such that  $l \leq k < i$ .

We say that  $c$  is *justified* at  $i$  if  $c$  is justified by  $b$  at  $i$  for some action  $b$ .

Note that an action can be justified by more than one action at the same time instant.

We are now ready to define models of a history. This definition must take into account whether actions are justified or not for the purpose of reasoning about activity recognition.

In addition to satisfying history statements as defined above, a trajectory must satisfy a number of additional conditions to be a model. Condition (2) below precludes vacuous actions from models. Condition (3) intuitively says that for every action in progress there must be at least one action that justifies it from start to end. Condition (4) says that an activity cannot end if an action that appears in its sequence is still intended, unless that action is justified by some other activity. Finally, Condition (5) says that at the end of the trajectory, no intended actions remain.

**Definition 5** A trajectory  $P = \langle \sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n \rangle$  is a model of a history  $H$  of an action description  $AD$  if the following conditions hold:

1.  $P$  satisfies all the statements of  $H$ ;
2. for each  $1 \leq i \leq n$ , all the elements of  $a_i$  are supported at  $i$ ;
3. for every action  $c$  such that  $P$  satisfies  $intended(c, i)$  and  $c$  starts at  $i$  and ends at  $j$ , there is an action  $c'$  such that  $c$  is justified by  $c'$  at  $k$  for every  $i \leq k < j$ ;

<sup>2</sup>An action “starts” when it becomes intended even if it has not yet started to *execute*. So “starts” and “ends” mark the period between an action first becoming intended and the termination of its execution.

4. for every activity  $(s, \langle c_1, \dots, c_m \rangle)$  such that  $s$  ends at  $i + 1$ , there is no action  $c_k$ ,  $1 \leq k < m$ , in the sequence of  $s$  such that
  - (a)  $P$  satisfies  $intended(c_k, i)$ ,
  - (b)  $c_k$  is justified by  $s$  at  $i$ ,
  - (c) there is no  $s' \neq s$  such that  $c_k$  is justified by  $s'$  at  $i$ ;
5. for every action  $c$ , if  $c$  is in progress at  $n$ , then  $c$  ends at  $n + 1$ .

**Example 2** Consider again the Cooking domain of Example 1. Suppose that we have a history containing the following statements:

$intended(mk\_fettuccini, 1)$ ,  
 $intended(mix\_chicken\_marinara, 3)$ .

If we do not allow concurrent actions, this history has no models of length less than 4. It has one model of length 4 with actions:

$mk\_fettuccini$ ,  
 $mk\_marinara$ ,  
 $mix\_chicken\_marinara$ ,  
 $mix\_fettuccini\_marinara$

occurring in that order. (With concurrency it can be shortened to a length of 3.) Intuitively, this means that two activities are occurring: cooking chicken marinara ( $ccm$ ), which is in progress from time 2 to 3, and cooking fettuccini marinara ( $cfm$ ), which is in progress from 1 to 4. The activities share the action  $mk\_marinara$  which is justified by both  $ccm$  and  $cfm$  at time 2.

The same history has 4 models of length 5. One of them contains the actions

$mk\_marinara$ ,  
 $mk\_fettuccini$ ,  
 $mix\_chicken\_marinara$ ,  
 $mk\_marinara$ ,  
 $mix\_fettuccini\_marinara$

In this model  $ccm$  and  $cfm$  are again the purposeful activities that occur.  $ccm$  is in progress from 1 to 3 and  $cfm$  from 1 to 5. This time  $mk\_marinara$  is not shared between the activities because it occurs for  $ccm$  before it becomes intended for  $cfm$ .

In the other two models,  $ccm$  and cooking fettuccini alfredo ( $cfa$ ) occur, in one  $mk\_marinara$  occurs at 1 and  $mk\_fettuccini$  at 2. In the other they are in the opposite order.

## Formalization in ASP

Our formalization of activity recognition in Answer Set Programming captures the domain transition system of Definition 1 by means of a set of rules using the predicate  $holds(F, T)$  (see e.g. (Baral 2003)). We have also adapted rules defining intention for elementary actions (Baral and Gelfond 2005) and activities (Gelfond 2006). The notion of *justified* activities requires a substantial elaboration with rules to recognize intention of actions at earlier times with respect to a given history statement. In the ASP formalization below, we assume that actions contain at most one primitive action. In other words, there is no concurrency and the

empty action, i.e., nothing occurs, is allowed. This permits us to treat actions as primitive objects rather than sets and makes the presentation simpler. It is not difficult, however, to extend the formalization with general concurrent actions.

Let us describe the main components of the ASP formalization. Although many rules below seem to be unsafe, we assume implicit domain predicates are used to make them safe. For computing answer sets, we use the Smodels construct `#domain` to specify a domain for all the variables that appear in the rules below. We omit domain predicates for readability but it should be kept in mind that they are needed and included in order to make all the rules safe. We start with the component that, given a statement *intended*(*c*, *i*) in the history, conjectures that some activity *s* that has *c* in its sequence is in progress.

```
inprogress(S, I) :-
    component(C, K, S),
    intended(C, I),
    K <= I,
    not other_justified(C, I, S).

other_justified(C, I, S) :-
    component(C, K, S),
    justified(C, I, S1),
    neq(S, S1).
```

Conjecturing that activities are in progress possibly leads to concluding that the conjectured activity was intended at some point:

```
intended(S, I) :-
    inprogress(S, I),
    not inprogress(S, I-1).

intended(S, I) :-
    inprogress(S, I),
    ends(S, I).
```

Next we describe the component that captures the notion of justified actions, starting with self-justified actions:

```
justified(C, I, C) :-
    inprogress(C, I),
    purposeful(C).
```

The following two rules capture item 2 of Definition 4:

```
justified(C, I, C) :-
    inprogress(C, I),
    component(C, K, S),
    inprogress(S, I),
    not justified_before(C, S, I).

justified_before(C, S, I2) :-
    start_to_now(S, I1, I2),
    justified(C, I, S),
    not other_justified(C, I, S),
    ends(C, I3),
    I1 <= I, I < I3, I3 <= I2.
```

Another part of our formalization captures reasoning about which actions are in progress and which are intended. This part can be divided into two components: given a history statement about time point *i*, one component does inference about what holds at time points preceding *i* and the

other inference about time points later than *i*. We start with the rules for reasoning about later time points.

The following two rules directly encode the definition of *in progress*:

```
inprogress(S, I) :- starts(S, I).

inprogress(S, I) :-
    inprogress(S, I-1),
    not ends(S, I).
```

In (Gelfond 2006), the formalization of intended actions includes rules for drawing conclusions about later time points with respect to history statements. In addition to the use of *inprogress* and some other differences, we must generalize those rules to take into account reasoning about *justified* actions.

```
intended(C, I) :-
    starts(S, I),
    component(C, 1, S).

intended(C2, I) :-
    inprogress(S, I),
    component(C2, K, S),
    component(C1, K-1, S),
    ends(C1, I),
    justified(C1, I-1, S).
```

The component for inference at earlier time points includes a rule saying that if an activity is in progress and it justifies one of its components that is not the first, then it is in progress in the previous time point:

```
inprogress(S, I) :-
    inprogress(S, I+1),
    intended(A, I+1),
    component(A, K, S),
    K > 1,
    justified(A, I+1, S).
```

Given history statements about the occurrence of an action, it is possible to draw inferences about the intentions of the agent before the action was executed, especially if the action is not self-justified. The following rules say, roughly, that if the action is a component of an activity and the activity has an earlier component, then either the earlier component is intended in the preceding time point or the action that occurred was intended in the preceding time point.

```
intended(A1, I) :-
    inprogress(S, I+1),
    occurs(A2, I+1),
    component(A2, K, S),
    component(A1, K-1, S),
    not -occurs(A1, I),
    not intended(A2, I).

intended(A2, I) :-
    inprogress(S, I+1),
    occurs(A2, I+1),
    component(A2, K, S),
    component(A1, K-1, S),
    not intended(A1, I).
```

Finally, the following rules capture conditions (3–5) in Definition 5:

Condition (3):

```

:- inter(C, I, I1),
   not full_just(C, I, I1).

full_just(C, I, I1) :-
   full_justified(C, C1, I, I1).

full_justified(C, C1, I, I1) :-
   justified(C, I, C1).

full_justified(C, C1, I, I1) :-
   justified(C, I, C1),
   I < I1,
   full_justified(C, C1, I+1, I1).

```

Condition (4):

```

:- ends(S, I+1),
   intended(C, I),
   justified(C, I, S),
   not other_justified(C, I, S),
   length(S, K),
   not component(C, K, S).

```

Condition (5):

```

:- intended(C, n).

```

In the above last rule,  $n$  is a constant defined to be the maximum length of the trajectories to be considered when solving an activity recognition problem.

The above set of rules will be denoted by  $\Pi_{ar}$ .

The encoding of a history  $H$  and an action description  $AD$  is as follows. Statements  $intended(c, i)$ ,  $intended(s, i)$ ,  $happened(a, i)$  and  $observed(l, i)$  are encoded directly as facts  $intended(a, i)$ ,  $intended(s, i)$ ,  $happened(a, i)$  and  $observed(l, i)$ , respectively. The set of such facts obtained from a history  $H$  is denoted by  $\pi(H)$ .

The translation of a domain description  $AD$  is denoted by  $\pi(AD)$  and contains the following rules and facts. For each named activity  $(s, \langle c_1, \dots, c_m \rangle)$  we include the facts:

```

activity(s).
length(s, m).
component(c_1, 1, s).
...
component(c_m, m, s).

```

As mentioned earlier, there is a fact  $purposeful(c)$  for each purposeful action or named activity  $c$ .

Dynamic causal laws, static causal laws and executability propositions are translated as follows:

For each dynamic causal law  $causes(a, l_0, [l_1, \dots, l_n])$  in  $AD$ :

```

holds(l_0, I+1) :-
   occurs(a, I),
   holds(l_1, I),
   ...
   holds(l_n, I).

```

For each static causal law  $caused(l_0, [l_1, \dots, l_n])$  in  $AD$ :

```

holds(l_0, I) :-
   holds(l_1, I),
   ...
   holds(l_n, I).

```

Finally, for each executability proposition  $impossible\_if(a, [l_1, \dots, l_n])$  in  $AD$ :

```

-occurs(a, I) :-
   holds(l_1, I),
   ...
   holds(l_n, I).

```

We also include in  $\pi(AD)$  a rule for the common sense law of inertia:

```

holds(L, I+1) :-
   holds(L, I),
   not holds(-L, I+1).

```

For reasoning about intended atomic actions, we use the following set of rules, denoted by  $\Pi_I$ :

```

%% Observed actions are intended
intended(A, I) :- happened(A, I).
occurs(A, I) :- happened(A, I).

```

```

occurs(A, I) :-
   intended(A, I),
   not -occurs(A, I).

```

```

intended(A, I+1) :-
   intended(A, I),
   -occurs(A, I),
   not -intended(A, I+1).

```

```

starts(A, I) :-
   intended(A, I),
   not intended(A, I-1).

```

```

starts(A, I) :-
   intended(A, I),
   ends(A, I).

```

```

ends(A, I+1) :- occurs(A, I).

```

```

inprogress(A, I) :- starts(A, I).

```

```

inprogress(A, I) :-
   inprogress(A, I-1),
   not ends(A, I).

```

The rest of the rules in the formalization are minor auxiliary rules.

Throughout the paper we have been focusing on intention and occurrence of actions and have sidelined reasoning about the value of fluents in the states of the trajectory. Incompleteness of information about fluents is another interesting aspect of the activity recognition problem we are considering. We leave that side of the problem for the full version of this paper. The following result assumes a history with complete initial state, i.e. for every fluent  $f$  there is a statement  $observed(f, 0)$  or  $observed(\neg f, 0)$ .

The complete formalization of an activity recognition problem in answer set programming is the program  $\pi(AD, H) = \Pi_I \cup \Pi_{ar} \cup \pi(AD) \cup \pi(H)$ . The models of a program  $\pi(AD, H)$  induce trajectories as follows.

**Definition 6** Let  $A$  be a subset of the literals of a given program  $\pi(AD, H)$ .  $A$  is said to define the trajectory

$\langle \sigma_0, a_1, \sigma_1, \dots, \sigma_n, a_n \rangle$  if  $\sigma_i = \{l \mid \text{holds}(l, i) \in A\}$  and  $\text{occurs}(a_j, j) \in A$  for all  $0 \leq i \leq n$  and  $1 \leq j \leq n$

**Theorem 1** For an action description  $AD$  and history  $H$  with complete initial state, a trajectory  $P$  without concurrency (i.e. all actions are singletons or empty sets) is a model of  $H$  iff  $P$  is defined by an answer set of the program  $\pi(AD, H)$ .

**Example 3** The following are the results of some sample runs with the Cooking domain. Using the following facts:

```
intended(make_fettuccini, 1) .
happened(mix_chicken_marinara, 3) .
```

and setting the max (plus 1) trajectory length constant `const n=5` we obtain an answer set that contains:

```
inprogress(cfm, 1)
inprogress(cfm, 2)   inprogress(ccm, 2)
inprogress(ccm, 3)   inprogress(cfm, 3)
inprogress(cfm, 4)
justified(make_fettuccini, 1, cfm)
justified(make_marinara, 2, ccm)
justified(make_marinara, 2, cfm)
justified(mix_chicken_marinara, 3, ccm)
justified(mix_fettuccini_marinara, 3, cfm)
justified(mix_fettuccini_marinara, 4, cfm)
occurs(make_fettuccini, 1)
occurs(make_marinara, 2)
occurs(mix_chicken_marinara, 3)
occurs(mix_fettuccini_marinara, 4)
```

Now let us try the following history:

```
happened(make_marinara, 2) .
happened(make_marinara, 4) .
```

With `const n=5` we get no answer sets, as expected. If we increase the constant to 6 we get two answer sets. In one, *ccm* happens twice and with no action occurring at 1:

```
inprogress(ccm, 2)
inprogress(ccm, 3)
inprogress(ccm, 4)
inprogress(ccm, 5)
occurs(make_marinara, 2)
occurs(mix_chicken_marinara, 3)
occurs(make_marinara, 4)
occurs(mix_chicken_marinara, 5)
```

In the second answer set, *cfm* is in progress from 1 to 3 and *ccm* from 4 to 5.

```
inprogress(cfm, 1)
inprogress(cfm, 2)
inprogress(cfm, 3)
inprogress(ccm, 4)
inprogress(ccm, 5)
occurs(make_fettuccini, 1)
occurs(make_marinara, 2)
occurs(mix_fettuccini_marinara, 3)
occurs(make_marinara, 4)
occurs(mix_chicken_marinara, 5)
```

Setting `const n=7` gives us two additional answer sets:

```
inprogress(ccm, 2)
inprogress(ccm, 3)   inprogress(cfm, 3)
inprogress(ccm, 4)   inprogress(cfm, 4)
```

```
inprogress(ccm, 5)   inprogress(cfm, 5)
inprogress(ccm, 6)
occurs(make_marinara, 2)
occurs(make_fettuccini, 3)
occurs(make_marinara, 4)
occurs(mix_fettuccini_marinara, 5)
occurs(mix_chicken_marinara, 6)
```

and

```
inprogress(ccm, 2)
inprogress(ccm, 3)   inprogress(cfm, 3)
inprogress(ccm, 4)   inprogress(cfm, 4)
inprogress(ccm, 5)   inprogress(cfm, 5)
                                     inprogress(cfm, 6)
occurs(make_marinara, 2)
occurs(make_fettuccini, 3)
occurs(make_marinara, 4)
occurs(mix_chicken_marinara, 5)
occurs(mix_fettuccini_marinara, 6)
```

## Related Work

There is a substantial body of work on activity or plan recognition. The work of (Kautz and Allen 1996), from which we adapted the cooking example, was among the first to use some kind of action formalism, albeit a simple one and without considering an explicit notion of intended actions. There are some recent approaches based on Hidden Markov Models (Bui, Venkatesh, and West 2002; Blaylock and Allen 2003). These approaches cannot handle multiple activities occurring simultaneously as in our framework. Perhaps closer to our action based approach is the recent work of (Demolombe and Hamon 2002; Demolombe and Fernandez 2005). This work is based on a different action formalism, the situation calculus, and instead of notions of purposeful and justified actions, that framework relies on an explicit specification of which actions must *not occur* at particular points of an activity if it is to be recognized from the history. This makes the definitions of activities more complicated and less elaboration tolerant. The approach of (Goultiaeva and Lespeance 2007) extends that of (Demolombe and Hamon 2002) by reformatizing the problem in a way that allows incremental recognition of plans. Both of these approaches require the history to be a complete prefix of an activity in order to recognize it. Thus they are not able to solve the problems in Example 3 because the history is missing some of the earlier occurring actions. These frameworks also require observations of actions to be made in the order the actions occur and cannot handle shared actions between activities. On the other hand, they can define more complex activities than sequences. A more general notion of activity is in our plans for future work.

It is worth pointing out as well that none of the above frameworks allow in the history statements of intention to execute an action or activity, since they do not employ a formal notion of intended actions. Finally, another unique feature of our approach is the direct implementation of the answer set programming formalization through execution by answer set finders like Smodels (Niemelä and Simons 1997). Nevertheless, several of the above frameworks have features we would like to integrate into our framework, e.g. activities

of a more general form than sequences, and probabilities or qualitative information useful for choosing among alternative solutions to an activity recognition problem.

## Conclusion

We have introduced a new approach to activity recognition that is based on a formal theory of actions and a notion of intended actions. Our approach is based on using knowledge about the intention and the occurrences of non-purposeful actions to conjecture that more complex purposeful activities may be occurring. In addition to  $\mathcal{A}$ -type action language domain descriptions with a transition system-based semantics, we provide a formalization in answer set programming that can be readily used to solve problems by submitting the formalization to an answer set finder such as Smodels (Niemelä and Simons 1997). We have illustrated our approach with problems in a meal cooking domain and included some sample results obtained with Smodels.

This framework can be extended in many possible ways. For instance, if one considers a case where the observed agent's actions can fail, then we would need to recognize or predict that the agent repeated the execution of an action because the first try failed, and that because it failed, its intention to execute that action persisted after the first try. Another extension to consider is where the observed agent may abandon an intended activity in the middle of the execution and start executing some other activity. We plan to look at these and many other interesting possibilities in future work.

## References

- Baral, C., and Gelfond, M. 2000. Reasoning agents in dynamic domains. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Kluwer.
- Baral, C., and Gelfond, M. 2005. Reasoning about intended actions. In *Procs. of the 20th National Conference on Artificial Intelligence (AAAI'05)*, 689–694.
- Baral, C. 2003. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press.
- Blaylock, N., and Allen, J. 2003. Corpus-based statistical goal recognition. In *Proceedings Procs. of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*.
- Bui, H.; Venkatesh, S.; and West, G. 2002. Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research*.
- Demolombe, R., and Fernandez, A. 2005. Intention recognition in the situation calculus and probability theory frameworks. *Computational Logic in Multi Agent Systems* 358–372.
- Demolombe, R., and Hamon, E. 2002. What does it mean that an agent is performing a typical procedure? a formal definition in the situation calculus. In *Procs. of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, 905–911.
- Geib, C. W., and Steedman, M. 2007. On natural language processing and plan recognition. In *Procs. of the International Joint Conference on Artificial Intelligence (IJCAI'07)*.
- Gelfond, M. 2006. Going places - notes on a modular development of knowledge about travel. In *Procs. of the AAAI Spring Symposium "Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering"*.
- Goultiaeva, A., and Lespeance, Y. 2007. Incremental plan recognition in an agent programming framework. In *Procs. of AAAI 2007 Workshop on Plan, Activity, and Intent Recognition (PAIR'07)*.
- Kautz, H., and Allen, J. F. 1996. Generalized plan recognition. In *Procs. of 13th National Conference on Artificial Intelligence (AAAI'96)*, 32–37.
- Niemelä, I., and Simons, P. 1997. Smodels—an implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. 4th Int'l Conf. on Logic Programming and Non-Monotonic Reasoning*, 420–429.