

Linköping Electronic Articles in
Computer and Information Science
Vol. 2(1997): nr 19

Logic Based Modelling of Goal-Directed Behavior

Erik Sandewall

Linköping University Electronic Press
Linköping, Sweden

<http://www.ep.liu.se/ea/cis/1997/019/>

*Published on December 19, 1997 by
Linköping University Electronic Press
581 83 Linköping, Sweden*

**Linköping Electronic Articles in
Computer and Information Science**

ISSN 1401-9841

Series editor: Erik Sandewall

©1997 Erik Sandewall

Typeset by the author using L^AT_EX

Formatted using étendu style

Recommended citation:

*<Author>. <Title>. Linköping Electronic Articles in
Computer and Information Science, Vol. 2(1997): nr 19.
<http://www.ep.liu.se/ea/cis/1997/019/>. December 19, 1997.*

This URL will also contain a link to the author's home page.

*The publishers will keep this article on-line on the Internet
(or its possible replacement network in the future)
for a period of 25 years from the date of publication,
barring exceptional circumstances as described separately.*

*The on-line availability of the article implies
a permanent permission for anyone to read the article on-line,
to print out single copies of it, and to use it unchanged
for any non-commercial research and educational purpose,
including making copies for classroom use.*

*This permission can not be revoked by subsequent
transfers of copyright. All other uses of the article are
conditional on the consent of the copyright owner.*

*The publication of the article on the date stated above
included also the production of a limited number of copies
on paper, which were archived in Swedish university libraries
like all other written works published in Sweden.
The publisher has taken technical and administrative measures
to assure that the on-line version of the article will be
permanently accessible using the URL stated above,
unchanged, and permanently equal to the archived printed copies
at least until the expiration of the publication period.*

*For additional information about the Linköping University
Electronic Press and its procedures for publication and for
assurance of document integrity, please refer to
its WWW home page: <http://www.ep.liu.se/>
or by conventional mail to the address stated above.*

ABSTRACT

We address the problem of characterizing goal-directed robotic behavior using a logic of actions and change. Our approach is based on distinguishing two kinds of actions: *procedural* actions which are defined in a mechanistic way, and *goal-directed* actions which are performed through a process involving tries, possibly failures, and corrective action and new tries until the goal has been reached. (The definition of procedural actions may be done external to the logic, for example through differential equations, or through a conventional programming language). For both kinds of actions, the logic expresses explicitly whether the action *succeeds* or *fails*. Each execution of a goal-directed action is also characterized by a number of *breakpoints* where some sub-action has been completed and a new sub-action for getting to the desired goal is selected. The logic is used for characterizing the selection of sub-actions at breakpoints, and the success or failure of the goal-directed action in terms of the success or failure of the sub-actions.

The article describes how goal-directed actions can be modelled by an extension of existing results on logics of actions and change.

ACKNOWLEDGEMENTS

This research was conducted within the Wallenberg Laboratory for Information Technology and Autonomous Systems (WITAS), which is supported by the Wallenberg Foundation.

SUMMARY

We address the problem of characterizing goal-directed robotic behavior using a logic of actions and change. The logicist formalization is required to be such that the set of models is exactly the set of acceptable goal-directed behaviors in the application at hand, each model being a representation of a possible history of the world. It is not required from the logic that it should *represent* psychologically related concepts, such as goals or intentions, but merely that the resulting behavior obtained from the formalization should have the characteristics of being goal-directed.

Frame of reference and motivation

We view intelligent robotic behavior as something which has both a reactive level and a deliberative level. The deliberative level is in charge of such things as receiving instructions from the commander, pursuing goals, predicting the possible outcomes of intended actions, avoiding dangers, diagnosing faults, planning means of acquiring knowledge, and so on. The underlying, reactive level may be characterized by stimulus-response behaviors and by program-like composition of simpler behaviors into more complex ones.

Logic is clearly relevant for the design and analysis of the deliberative level of such a design. However, the logic being used then must be able to characterize its different functionalities, including its goal-directedness. It must also be able to characterize the underlying, reactive level, not in every detail, but with sufficient precision for the needs of the deliberative layer.

Procedural vs goal-directed actions

The present article focuses on the characterization of goal-directed behavior as one aspect of an intelligent robot. However, we do not introduce “goals” as a separate logical construct. Instead, our approach is to distinguish two kinds of actions: *procedural* actions which are defined using a subroutine, and *goal-directed* actions which are performed through a process involving tries, possibly failures, and corrective action and new tries until the goal has been reached. Procedural actions may therefore be used for representing reactive-level behaviors to the deliberative layer; goal-directed actions characterize processes within the deliberative layer itself.

Success vs failure of actions

For both kinds of actions (goal-directed and procedural ones), the logic expresses explicitly whether the action *succeeds* or *fails*. Each execution of a goal-directed action is also characterized by a number of *breakpoints* where some sub-action has been failed, and another sub-action is selected for one more attempt to arrive at the desired goal. The logic is used for characterizing the selection of sub-actions at breakpoints, and the success or failure of the goal-directed action in terms of the success or failure of the sub-actions.

The term “goal” is used in the sense used in AI planning, that is, as the concrete goal (goal state) that is to be achieved by a plan = a sequence of actions. We do not use the word in the sense of “general goal” (goal in life).

Formalism for actions and change

We use a logic of time and action, that is, a multi-sorted first-order logic where points in time is one of the sorts. The following predicates are used:

$H(t, p)$: the “propositional fluent” p holds at time t .

$X(t, f)$: the fluent f is occluded at time t

$G(s, a)$: the action a is invoked (“go”) at time s

$A(s, a)$: the action a is applicable at time s

$D_s([s, t], a)$: the action a is executed successfully over the time interval $[s, t]$: it starts at time s and terminates successfully at time t .

$D_f([s, t], a)$: the action a is executed but fails over the time interval $[s, t]$: it starts at time s and terminates with failure at time t .

In fact, these predicates are viewed as abbreviations for underlying, more elementary constructs indicating the applicability and the failure of actions.

Actions are composed using the conventional operators such as sequential composition ($;$), conditionals, etc.

The article proposes a set of axioms that characterize possible structures using these and related predicates.

Formalism for goal-directed behavior

A few specialized relations are introduced for characterizing the goal-directed behavior. The key relation is:

$Option(s', g, s, a)$: while pursuing the goal set by invoking the goal-directed action g at time s' , one is now at a timepoint s (a breakpoint), and one considers performing the action a . This action may be a composite one, which means that it has the character of a plan. Several follow-up actions may be considered at the same point in time.

The article proposes a set of axioms that characterize goal-directed behavior in terms of this predicate and the ones mentioned under the previous heading.

Entailment methods

A logic characterizing goal-directed use of actions must deal with multiple knowledge sources, including action laws characterizing the effects of actions, observations of the state of the world, plans for achieving specific goals, and so forth. Each of these aspects of the system may call for some kind of nonmonotonicity in order to be sufficiently selective. It is not trivial how to combine the different types of nonmonotonicity that are required. The present article proposes and uses a general approach for dealing with this problem.

ARTICLE

1 Topics and motivation

In the traditions of situation calculus and STRIPS, actions are considered as transitions from one state to the next. In implementation terms, this corresponds to an architecture where the cognitive or ‘deliberative’ layer of the robot is able to invoke actions much like a program invokes a subroutine: the action-subroutine is executed until it terminates, and then the cognitive level regains control.

This point of view is not sufficient for real-world robotics. For practical purposes, the execution of an action must be able to be influenced by external events at any point during the interval of time where it is being executed. These ‘external events’ may be events in the physical world surrounding the robot. They may also be additional decisions in the deliberative layer which operates concurrently with, although in a logical sense ‘above’ the operative definition of the action, or they may be defined by the execution of other actions that are concurrent with the one at hand.

From a knowledge representation point of view, we are faced with the need of characterizing processes which operate concurrently and which are of two different kinds: the continuous or hybrid (i.e. mixed continuous and discrete) process in the lower levels of the robot architecture, and the reasoning processes in the higher, deliberative layer(s). The reasoning process must have an adequate model of the underlying hybrid process; the hybrid process must be defined in such a way that it can accept the input from the reasoning process, and the model of the hybrid process used by the reasoning process must be able to represent this capability. (We use the term “hybrid” so as to include “continuous” as a special case).

When addressing this difficult representational problem, it is natural to focus first on some specific cases of deliberative behavior. In this article we address the case of *goals* in the sense of the word that is used in the planning and problem-solving literature, that is, conditions on the state of the world that are to be achieved by a suitable sequence of actions. (Sometimes the word “intention” is used with roughly this meaning).

1.1 Defining actions on the level of hybrid processes

While goal-directed actions is the main topic here, we also need an approach to the representation of actions on the continuous or hybrid level. In particular, one important aspect of goal-directed behavior is the capability of recognizing that an action has failed, and of deciding to try something else instead. On the deliberative level it is difficult to say very much about action failure, besides the fact that either the action succeeds or it doesn’t, and the effects in each case. On the other hand, the detailed definition of an action that is found on the hybrid level specifies under which conditions the action succeeds and achieves its intended result state, and when it fails.

The continuous or hybrid level may be characterized by sets of differential equations which are associated with applicability conditions, so that different equations are stated to apply in different segments of the state space at hand. In earlier papers, we have shown how to import this representation into a logical framework [7] and how to combine it with specifications of the effects of actions [8]. More recently, we have also shown how

to relate high-level and low-level definitions of actions in such a framework, and in particular how to use the hybrid-level action description for defining success and failure of actions [11, 12].

Concretely speaking, there are quite a number of intervening events that may cause the failure of an action in a physical world. Let us take some examples from our own application: the present work has been done within a project on methods for intelligent UAV's. The task of the UAV is to fly over traffic scenes, to understand what happens in those scenes, and to take appropriate action towards certain goals, for example for assisting one or more cooperating ground vehicles in their missions. Our application therefore involves actions that are performed over periods of time (for example, "follow that car", "accompany the cooperating car to its destination", or "find a place satisfying certain conditions". Reasons why such actions can fail include the actions of the ground vehicles, obstructions to vision or to communication, restrictions on fuel or other resources in the UAV, etc.

From our earlier work, we have methods for relating the hybrid-level specification of an action to any number of environmental events which may influence or distract it, and for relating the global, precondition/ post-condition type description of the action to the one on the hybrid level.

From the point of view of the hybrid specification of the action, it makes no difference if it is stopped because of an intervention by a person who commands the robot (which counts as an external event as modelled in our earlier work), or because the deliberative layer at some point realized an upcoming danger and decided on an interrupt. Therefore, we can now address the question of modelling high-level behavior that reasons about and uses the success and the failure of subordinate actions.

1.2 If you fail, then try again

We shall consider goals as the end states of particular kinds of actions, namely *goal-directed* actions. Therefore, the question of characterizing goal-directed behavior is reduced to the question of how goal-directed actions are decomposed into procedural ones.

A goal-directed action is an open-ended process: the robot sets out to achieve a certain goal, for example to find and retrieve a particular object, or to obtain more fuel. It chooses certain lower-level actions which are likely to achieve the goal, but these lower actions may succeed or fail; if they fail then the robot diagnoses the error and tries to achieve the same goal in some other way. At the lowest level in this structure one finds the procedural actions, but above them there is a structure of goal-directed actions, possibly on several levels. They are related and held together by decision-like entities and other events, for example decision to try, faults, diagnoses, and decisions about retries. In some cases this structure will be quite simple; in other cases it may be very complex.

The problem addressed here is how such goal-directed behavior can be characterized in a logic of actions and change. We propose an extended action logic whose expressive power is sufficient for this purpose. Concretely, the characteristic property of this logic of action and change is that for a well chosen axiomatization of a scenario, the models represent exactly those histories where the robot exhibits the appropriate, goal-directed behavior. In that sense, the proposed logic is a characterization of goal-directed behavior.

One limitation of the present article is that we only consider the relationship between lower-level and higher-level actions inside the deliberative

layer. In the end it will be necessary to deal with influences from external events, the deliberative layer, and concurrent events in an integrated fashion, requiring a closer integration with the earlier work, but we begin here with the simpler case where the latter two aspects are omitted for the time being.

2 Background formalism

We represent both procedural and goal-directed actions using a *narrative time-line approach*, that is, a first-order multi-sorted logic where time, represented as the real numbers, is one of the sorts. The syntax of the base logic is quite similar to what has been used before both by us and by several others. The present section summarizes the formalism that we share with the previous work.

2.1 Standard Time and Action Logic

We use three primary predicates. The predicates H for *Holds* and D for *Do* are defined as follows. $H(t, p)$ says that the “propositional fluent”¹ p holds at time t . In other words, p is reified and $H(t, p)$ is the same as $p(t)$ in the case where p is atomic. $D([s, t], a)$ says that the action a is performed over exactly the closed temporal interval $[s, t]$. Open and semiopen intervals are denoted (s, t) , $[s, t)$, and $(s, t]$ as usual.

Non-propositional fluents are also admitted, using the notation $H(t, f : v)$ where $f : v$ is the proposition saying that the fluent f has the value v . Fluent-valued functions are allowed, and one of their uses is to define fluents for properties of objects. Thus $ageof(p)$ may be the fluent for the age of the person p , used as in $H(1998, ageof(john) : 36)$.

The third predicate, X is pronounced *occludes* and is used for characterizing exceptions from the assumption of continuity of the value of fluents. Continuity includes persistence as a special case, for discrete-valued fluents. $X(s, f)$ expresses that at time s , the value of the fluent f is not required to be continuous or to persist.

In all cases, s and t are timepoints (usually s for starting time and t for termination time) and a is an action.

In narrative time-line approaches, each model of the axioms characterizes one possible history in the world, not a tree of possible histories. Alternative histories are represented by different models. Therefore, a timepoint t is sufficient for identifying the state of the world at time t in the present model.

Several earlier publications by ourselves and others in our group have used the notations $[t]p$ and $[s, t]a$ for what is here written $H(t, p)$ and $D([s, t], a)$, respectively. The change is made in order to emphasize more strongly that we are dealing with a fairly standard first-order logical theory.

¹We have previously tried to maintain a terminological distinction between *fluent* as a *function* from timepoints to corresponding values, and a *feature* as a formal object that designates a fluent. With that terminology, the p and f that occur in the second argument of H are features, not fluents. Similarly, the functions *inv*, *app*, and *fail* that will be introduced later in this section, are functions from actions to features. However, since it is so common to use the ‘fluent’ both for the function and its designator, we follow that practice here.

2.2 Representing success and failure of actions

Since the performance of a goal-directed action involves trying lower-level actions which may succeed and fail, and to proceed accordingly, we need a notation for dealing with the applicability, success, and failure of actions. The following notions will be used. *Invocation* of an action is possible at any time. However, the invocation does not necessarily lead to the *execution* of the action. In particular, invocation does not lead to execution if the action is inapplicable by definition (for example, turning on the light in a room where there is no light) or if the action is already executing. The latter condition means that the same action can not execute over two overlapping but non-equal intervals of time. Once an execution does execute, it must either *succeed* or *fail*. The distinction between success and failure is done on the following pragmatic grounds: planning goal achievement is done using the assumption that actions succeed, and using knowledge about their results when they do succeed. The case where an action fails is dealt with on a case-by-case basis once the failure has occurred.

Each action has a temporal duration, which must be an interval that is greater than a single point except for some specific cases defined below. Note, in particular, that when an action is not applicable, it is considered not to execute; it is not considered to fail instantly.

Two representations will be used for the expression of success, failure, and applicability of actions. In one, we use specially constructed fluents; in the other, variants of the D predicate that distinguish between action success and action failure. The former representation is considered as the basic one, and the latter is introduced as abbreviations or 'macros'.

The following are three functions from actions to propositional fluents:

inv, where $H(s, inv(a))$ says that the action a is invoked at time s . At all other times, $H(s, inv(a))$ is false.

app, where $H(s, app(a))$ says that the action a is applicable at time s .

fail, where $H(t, fail(a))$ says that the action a terminated with failure at time t . $H(t, fail(a))$ is false at all times when the action is not executing, or when it is executing but not terminating, or when it is terminating successfully.

In addition, we need one function from propositional fluents (properly speaking, propositional features) to actions:

test, where $test(p)$ or $test(f : v)$ is an action that is always applicable, whose duration is always instantaneous (expressed by $D([s, s], test(p))$), and that satisfies

$$H(s, fail(test(p))) \leftrightarrow \neg H(s, p)$$

In other words, $test(p)$ succeeds at time s iff p is true at s .

The following abbreviations are introduced:

$G(s, a)$ for $H(s, inv(a))$: the action a is invoked ("go") at time s

$A(s, a)$ for $H(s, app(a))$: the action a is applicable at time s

$D_s([s, t], a)$ for $D([s, t], a) \wedge \neg H(t, fail(a))$: the action a is executed successfully over the time interval $[s, t]$; it starts at time s and terminates with success at time t .

$D_f([s, t], a)$ for $D([s, t], a) \wedge H(t, fail(a))$: the action a is executed but fails over the time interval $[s, t]$; it starts at time s and terminates with failure at time t .

$D_c([s, t], a)$ for $\exists u[D([s, u], a) \wedge t \leq u]$: the action a is being executed; the execution started at time s and has not been terminated before time t . (It may terminate at t or later).

$D_v(s, a)$ for $G(s, a) \wedge (\neg H(s, app(a)) \vee \exists s' \exists t [D([s', t], a) \wedge s' < s < t])$: the action a is invoked at time t but it is either not applicable, or already executing at that time. (This is the case where invocation of the action does not initiate an execution).

An additional variant of this relation will be introduced below.

For both D_s and D_f , s is the time when the action was invoked, and t is the exact time when it concludes with success or failure.

2.3 Axiomatic characterization

The following set of axioms characterizing the obvious properties of these relations is an adaptation of the axioms reported in [11]. The adaptation is because we here introduced *inv*, *app*, and *fail* as the basic notions; whereas previously the relations G , D_s , etc were considered as basic.

S1. If an action is being executed, then it must have been invoked and be applicable and non-executing at invocation time:

$$D([s, t], a) \rightarrow H(s, inv(a)) \wedge \neg D_v(s, a)$$

This implies:

$$D([s, t], a) \rightarrow H(s, inv(a)) \wedge H(s, app(a)) \wedge \neg \exists s' \exists t [D([s', t], a) \wedge s' < s < t]$$

S2. If an action is invoked, then it is executed from that time on, unless it is inapplicable, already executing, or composite:

$$H(s, inv(a)) \rightarrow \exists t [s \leq t \wedge D([s, t], a)] \vee D_v(s, a) \vee Composite(a)$$

The predicate of *Composite* will be introduced in subsection 2.6; for the present context it can be taken as always false. The reason for excluding this case in this axiom is that for composite actions, there are some additional obstacles where the action the invocation of an action does not result in its execution.

S3. An action can not take place during overlapping intervals:

$$D([s, t], a) \wedge D([s', t'], a) \wedge s \leq s' < t \rightarrow s = s' \wedge t = t'$$

S4,S5. Actions of the form *test*(p) are always applicable, and instantaneous:

$$H(s, app(test(p)))$$

$$D([s, t], test(p)) \rightarrow s = t$$

S6. All other actions execute over extended periods of time: immediately, except for actions of the form *test*(p):

$$D([s, t], a) \rightarrow s < t \vee \exists p [a = test(p)]$$

S7. Actions only fail at the end of their execution:

$$H(t, fail(a)) \rightarrow \exists s [D([s, t], a)]$$

S8. Definition of success for actions of the form *test*(a):

$$D([s, s], test(p)) \rightarrow (H(s, fail(test(p))) \leftrightarrow \neg H(s, p))$$

Several of these axioms capture desirable properties directly. For others, all the consequences are not immediately obvious. One useful consequence is the following theorem, previously mentioned with in [11] for a somewhat different axiomatization:

Theorem 1 *In any model for the axiom S3, let $\{[s_i, t_i]\}_i$ be the set of all intervals such that $D([s_i, t_i], a)$ for a specific action a . Then there is some ordering of these intervals such that $s_i < s_{i+1}$ and $t_i \leq s_{i+1}$ for all i .*

Proof. Suppose the proposition does not hold, and choose an order of the pairs such that $s_i \leq s_{i+1}$, and where each pair only occurs once. Also, choose j so that either $s_j = s_{j+1}$, or $s_j < s_{j+1} < t_j$. If no such j is to be found, then the ordering already satisfies the condition in the proposition.

However, the case $s_j = s_{j+1}, t_j \neq t_{j+1}$ contradicts axiom (S3). The case $s_j < s_{j+1} < t_j$ also contradicts axiom (S3). This concludes the proof. QED.

The value of this observation is that through it, it makes sense to use the feature $fail(a)$ for characterizing the success or failure of an action with extended duration. If theorem 1 were not to hold, then it would not be clear from $H(t, fail(a))$ which invocation the failure referred to. This consideration is also the reason for the choice manifested in action S1: if an action a is invoked while it is already in the midst of executing, then it is not represented as “failing”, since this would confuse matters with respect to the already executing instance. Instead, we use the convention that it is invoked, possibly applicable, but it does not get to execute from that starting time.

We also obtain at once:

Theorem 2 *In any model for the axioms S1 – S8, if $D([s, t], a)$ and $H(u, fail(a))$ for some u in $(s, t]$, then $t = u$. Conversely, if $D_s([s, t], a)$, then $H(u, fail(a))$ does not hold for any u in $(s, t]$.*

Informally, we can think of each model in dynamical terms as a possible history in the world being described, and what this theorem says is that if an action is invoked and begins to execute, then if $H(u, fail(a))$ becomes true at some timepoint u during the execution, the action halts and ends with failure, and if it is able to proceed until its normal ending without $H(u, fail(a))$ becoming true at any time, then it ends with success.

Any use of this logic will naturally be concerned with the effects of actions. In the Features and fluents approach and its successors, this is specified using action laws, which in particular make use of the occlusion predicate, and in combination with assumptions of persistence.

2.4 Examples

As an example of the use of this notation, here is the formula stating that a condition φ guarantees that an action always succeeds:

$$H(s, \varphi) \wedge G(s, a) \rightarrow \exists t[D_s([s, t], a)]$$

Ordinary action laws specify the action’s effects when it succeeds. They are therefore written as usual and with D_s on the antecedent side: if preconditions apply and the action is performed successfully, then the postconditions result.

As another simple example, consider the case of actions which are described in terms of a precondition, a prevail condition, and a postcondition, where the postcondition is at the same time the termination condition for the action [13]. The prevail condition must be satisfied throughout the execution of the action; if it is violated then the action fails. Simple pre/ post/

prevail action definitions can be expressed as follows, if φ_a is the precondition of the action a , ω_a is the postcondition, and ψ_a is the prevail condition:

$$\begin{aligned} A(s, a) &\leftrightarrow H(s, \varphi_a) \\ D_s([s, t], a) &\rightarrow H(t, \psi_a \wedge \omega_a) \\ A(s, a) \wedge D_c([s, t], a) &\rightarrow H([s, t], \psi_a \wedge \neg\omega_a) \\ D_c([s, t], a) \wedge \neg H(t, \psi_a) &\rightarrow D_f([s, t], a) \end{aligned}$$

The traditional case of only pre- and postconditions is easily obtained by selecting ψ_a as tautology.

2.5 Composition operators for propositional fluents

The standard propositional operators such as \neg and \wedge will be used for composing propositional fluents (properly speaking: features). Composition is defined in a Herbrand style, so composite fluents are only equal if they have been equally formed. We have to specify how such composite fluents behave in relation to each of the predicates and functions that can take fluents as arguments:

1. The behavior of composite fluents with respect to the *Holds* predicate is defined by

$$H(s, p \wedge p') \leftrightarrow H(s, p) \wedge H(s, p')$$

and similarly for the others.

2. Composite fluents are always occluded, so that

$$X(s, p \wedge p')$$

and similarly for the others. This means that assumptions of continuity and persistence are only applied to the level of elementary fluents.

3. The action *test*(p) for composite p must then again be described with respect to how it relates to predicates and functions that take actions as arguments. All that was said about *test* above continues to hold, of course: it is always applicable, its duration is always instantaneous, and it succeeds or fails depending on whether p currently holds or not. No additional axiom is needed or appropriate for the special case where p is composite.

2.6 Action composition operators

The definitions of procedural actions must sometimes be constructed by composition of simpler actions. This calls for the use of operators such as $;$ for the sequential composition of actions, a conditional operator, and an operator that composes actions representing successive tries. Sequential composition is such that if the first action fails, then the whole action has failed, otherwise it is up to the second action. Successive-try composition, on the other hand, is defined so that if the first action *succeeds*, then the whole actions has succeeded; if the first action fails, then it is up to the second action to succeed or fail.

These action composition operators are best described in terms of their relationships with the derived predicates G , D_s , etc. This is as follows for $;$

$$\begin{aligned} G(s, a_1; a_2) &\rightarrow G(s, a_1) \wedge \\ & (D_v(s, a_1) \rightarrow D_v(s, a_1; a_2)) \wedge \\ & (D_f([s, t], a_1) \rightarrow D_f([s, t], a_1; a_2)) \wedge \\ & (D_s([s, t], a_1) \rightarrow G(t, a_2) \wedge \\ & (D_v(t, a_2) \rightarrow D_f([s, t], a_1; a_2)) \wedge \\ & (D_f([t, u], a_2) \rightarrow D_f([s, u], a_1; a_2)) \wedge \\ & (D_s([t, u], a_2) \rightarrow D_s([s, u], a_1; a_2))) \end{aligned}$$

Then, $try(a_1, a_2)$ denotes successive-try composition:

$$\begin{aligned}
& G(s, try(a_1, a_2)) \rightarrow G(s, a_1) \wedge \\
& (D_s([s, t], a_1) \rightarrow D_s([s, t], try(a_1, a_2))) \wedge \\
& (D_f([s, t], a_1) \rightarrow G(t, a_2) \wedge \\
& (D_v(t, a_2) \rightarrow D_f([s, t], try(a_1, a_2))) \wedge \\
& (D_f([t, u], a_2) \rightarrow D_f([s, u], try(a_1, a_2))) \wedge \\
& (D_s([t, u], a_2) \rightarrow D_s([s, u], try(a_1, a_2)))) \wedge \\
& (D_v(s, a_1) \rightarrow G(s, a_2) \wedge \\
& (D_v(s, a_2) \rightarrow D_v(s, try(a_1, a_2))) \wedge \\
& (D_f([s, t], a_2) \rightarrow D_f([s, t], try(a_1, a_2))) \wedge \\
& (D_s([s, t], a_2) \rightarrow D_s([s, t], try(a_1, a_2))))
\end{aligned}$$

Notice that if one omits the case of D_v , then the specifications of $;$ and try are symmetrical.

Although these specifications are easy to follow, they are not appropriate as axioms, since the operators G , D_s etc are not in themselves the primary ones. The Annex contains an axiomatization that relates $;$ and try to inv , app , $fail$, and $test$ and from which the specifications above can be inferred.

The action composition operators represent a kind of “programming language” for procedural actions. Actions which are defined in this way are however still not goal-directed in the sense discussed in the initial section. We shall proceed to goal-directed actions in the next section.

Composition of actions of the form $test(p)$ can be reduced to propositional combinations, for example

$$D_s([s, s], test(p); test(q)) \leftrightarrow D_s([s, s], test(p)) \wedge D_s([s, s], test(q))$$

We choose to *define* composition of such actions in terms of equality:

$$test(p); test(q) = test(p \wedge q)$$

$$try(test(p), test(q)) = test(p \vee q)$$

The definitions of “if” and “while” are obtained in similar manner, and are detailed in the Annex. (Note that “if p then a else b ” can not be expressed using the operators defined so far).

The predicate $Composite(a)$ is defined so that it is true for actions a that are formed using the functions $;$, try , etc., except for the ones where all components are of the form $test(p)$ so that they are equal to non-composite actions by what has just been said. It is straight-forward to write out the axioms for the definition of $Composite$.

3 Characterizing successive tries in logic

We proceed now to the phenomenon of *deliberative retry*, which is characteristic of high-level actions: if something goes wrong, then try again, but before you do that, consider carefully what different options are available. The weighing of possible alternatives is what differentiates deliberative retry from the preprogrammed successive-try composition defined above.

3.1 Ontology

The goal-directed behavior that we wish to characterize in the logic is as follows. At each point in time, the robot is engaged in no, one, or more *processes*, each of which is an instance of goal-directed behavior. Each process goes on for an interval of time, then it ends and once ended, the

same process can not restart. At each point in time within its duration, the process is carrying out a *plan*, which is an action and in the general case a composite action, formed using the action composition operators that were defined in section 2.6. Each constituent action may succeed or fail, which also defines the success or failure of the plan. If the current plan fails, then another plan is found, if possible, for achieving the current goal. If the current plan succeeds, then the current process succeeds. Therefore, there is an implicit assumption that the choice of plans and the definition of success of plans is such that the success of the plan guarantees that the goal has been achieved. If no applicable plan exists, then the process fails.

The robot controls this process in the following ways:

- The robot invokes a goal-directed process by stating the formula $H(s, inv(g))$.
- The robot discontinues an on-going process by stating the formula $H(t, fail(g))$.
- The robot discontinues an on-going action within a process by stating the formula $H(t, fail(a))$.
- The robot selects the new plan to be used within a process when a current plan has failed. This is done by stating the formula $H(s, inv(a))$ where a is often a composite action.

Notice that in all cases, the robot exercises its control by making statements of the form $H(t, p)$ where p has the property of being true at singular points in time, and false everywhere else.

It is assumed that the robot exercises these capabilities correctly, so that e.g. it does not discontinue processes that are not in course, it only selects plans that are guaranteed to achieve the goal if successful, etc.

For simplicity, we assume here that each process is linear in the sense that it is not able to spawn other, concurrent processes. This is in line with the fact that no action composition operator for concurrent execution was introduced in subsection 2.6.

Possible concurrent occurrences of the same action are assumed to behave in line with the formalization in the preceding sections. This means that if several concurrent processes request the same action a to be invoked at the same time, then this can be done, but only one instance of the action is invoked. If that instance fails, then the failure affects all the invoking processes. On the other hand, if a process invokes an action a at a time where the same action is already in course in another, concurrent process, then the new invocation of the action a falters (no new execution is initiated).

3.2 Additional formalism

Previous authors have sometimes used a modal operator for specifying goals. Here, we manage with a simpler approach, essentially because we have not set out to characterize goals, but only to characterize goal-directed behavior.

The basic idea has already been mentioned: We distinguish actions of several levels. The lowest level of action is the procedural one, the one which is implemented as a program or other routine behavior. Working towards a goal is represented as a higher level action, often realized by performing several low-level actions in succession.

Thanks to this approach, we only need to make some simple additions to the background formalism that was briefly reviewed above. We introduce

a few specialized predicates, besides the general-purpose predicates H, D, and X. The new predicates are:

- $Option(s', g, s, a)$ which says that while performing the high-level and goal-directed action g that was invoked at time s' , at time s the goal has not yet been achieved, and the world is in a state where the action a is executable, and where its successful execution will achieve the goal.
- $Realize(s, g, a)$ which says that at time s , the standard way of performing the (high-level) action g is to initiate the (low-level) action a .

What about if a proposed action or plan a is nondeterministic and *possibly* achieves the goal, but is not guaranteed to do so? In this case, one can always use $a; test(p)$ as the last argument of $Option$, where p is the desired goal condition. The action $a; test(p)$ will execute a , and if it succeeds then it tests whether p is true and if so it succeeds, otherwise the whole action fails.

It is assumed that these relations satisfy

$$Option(s', g, s, a) \rightarrow H(s, app(a))$$

$$Realize(s, g, a) \rightarrow Option(s, g, s, a)$$

Furthermore, if $achieve(p)$ is the generic goal-directed action having the property

$$D_s([s, t], achieve(p)) \rightarrow H(t, p)$$

that is, the action succeeds when p has been achieved, then

$$Option(s', achieve(p), s, a)$$

should be true for every a satisfying

$$Option(s', achieve(p), s, a) \wedge D_s([s, t], a) \models H(t, p)$$

The variable symbol a will be used for low-level actions, and g for high-level actions, and when explicitly or implicitly quantified they only range over those respective subtypes. At present, these subtypes are kept distinct, but we foresee a generalization where actions may be decomposed successively through several levels. The predicate D and the functions on actions (inv , app , $succ$) apply equally to high-level and low-level actions.

Finally, we need one more variant of the D predicate, this one defined directly and not as an abbreviation. The formula $D_b([s, t], g)$ will express that the goal-directed action g was invoked at time s , that at time t it has not yet succeeded, and that time t is a *breakpoint* in the sense that one of the attempts to achieve the goal has just failed, and the robot is considering what to do next. For given s and g where $G(s, g)$ is true, $D_b([s, t], g)$ will be true for all t which are breakpoints during the process of trying to reach the goal specified by g , and for no other t .

3.3 Axiomatization for the sequential case

The case of several concurrent processes has the particular complication that a proposed invocation of a step in a plan may falter because the same action is presently in the midst of executing. One must then decide whether to replan, or to wait until the action becomes available and then perform it,

or whether possibly it is sufficient to use the state of the world at the end of the present execution of the action. This choice is problem-dependent, which contributes to the complexity of concurrency in this context.

In the purely sequential case these problems do not occur, and we shall therefore treat them first; in this paper we limit our attention to them. The ontology described above is characterized by the following axioms, in addition to those defined in earlier sections.

- G1. $G(s, g) \rightarrow D_b([s, s], g)$
- G2. $D_b([s, t], g) \wedge \forall a[\neg Option(s, g, t, a) \rightarrow$
 $(s = t \rightarrow D_v(s, g)) \wedge$
 $(s < t \rightarrow D_f([s, t], g))$
- G3. $D_b([s, t], g) \wedge Option(s, g, t, a) \rightarrow D_f([s, t], g) \vee$
 $\exists a'[G(t, a') \wedge Option(s, g, t, a') \wedge$
 $(D_s([t, t'], a') \rightarrow D_s([s, t'], g)) \wedge$
 $(D_f([t, t'], a') \rightarrow D_b([s, t'], g))]$

These axioms are organized as a kind of “engine” for doing the goal-directed behavior, based on the notion of breakpoints, that is, points where a plan has failed and replanning has to take place. Axiom G1 says that immediately when a goal-directed action has been invoked, you are at a breakpoint for that action. Axiom G2 says that if you are at a breakpoint and no plan is available, then the process fails. Axiom G3 says that if you are at a breakpoint and some plan *is* available, then *some* plan (not necessarily the one mentioned in the antecedent) is invoked. It further says that if the selected plan succeeds, then the goal-directed action succeeds, and if the invoked plan fails, then the process is at a new breakpoint where replanning has to take place again.

Axiom G3 has an additional literal for the possibility that the goal-directed action fails. This literal is intended to cover the case that the robot decides to discontinue the process exactly when it is at a breakpoint. If the robot discontinues the goal while being within the execution of a plan, we need instead the following axiom:

- G4. $D_f([s, t'], g) \wedge Option(s, g, t, a) \wedge D_c([t, t'], a) \rightarrow D_f([t, t'], a)$

Note that this means that a success of a low level action can be redefined as failure by a high level action. On the other hand, if the robot chooses to fail the ongoing low-level action, the currently executing plan, or some segment of the currently executing plan, then the already defined axioms take care of it correctly.

3.4 Minimization of actions

The axioms that were specified in the previous section defined when certain actions and events must take place, including both the invocation and the failure of actions. They do not restrict the actions and events to the minimally necessary actions or the only motivated events, but presumably such minimization is intended. It would be contrary to the concept of goal-directed behavior to see the occurrence of unmotivated actions, and it would not make ontological sense to see the failure condition for actions trigger at arbitrary times and without reasons.

As usual, there are two ways of eliminating models containing unintended actions and events: by introducing additional axioms, or by an explicit minimization policy on models. In the concurrent case, it turns out to be quite difficult to eliminate all redundant actions by explicit axioms. One can get some of the way. For example, it is straightforward to write an axiom saying that every invocation of an action must be obtained from a *Option* statement. It is also not very difficult to write an axiom saying that if several alternative plans are proposed in the same case of *Option*, only one of them will be chosen. This appears to be sufficient for the non-concurrent case.

Suppose, however, that we have two concurrent processes which happen to have concurrent breakpoints; one of them considers plans *a* and *b*; the other one considers plans *a* and *c*. We would then accept that only plan *a* is selected, and possibly that plans *b* and *c* are selected, but certainly not that *a* and *b* is selected since *a* can do the job alone. To make matters worse, suppose we have three concurrent processes with shared breakpoint; one of them considers *a* and *b*, one considers *b* and *c*, and one considers *a* and *c*.

An axiomatization that deals correctly with these obstruse cases will presumably be quite complex, and unfortunately the complexity will largely be due to very odd cases. At the same time, it seems that the model-preferential specification can be made quite simple and concise: minimize $\{inv(a)\}_a$ chronologically.

A possible objection to such a principle might be that chronological minimization of effort may be very shortsighted, and sometimes it is important to look ahead and trade off current work against future comfort. However, this misses the point, since such tradeoffs can be done as appropriate in the replanning process and in the choice between alternative plans. The point made here is that *once the plans have been selected* in the participating processes, one only invokes a minimal set of actions at each point in time.

3.5 Characterizing the action processes

A complete cognitive robotics system needs to reason about the effects of actions, so it needs access to action laws (sometimes called action effect laws). In the context of a success/ failure distinction for actions, there is also a need for rules that specify the conditions under which actions may fail or are bound to fail.

None of these considerations have been made in the present article, because we shall use an approach where the treatment of those other aspects are dealt with separately and in a modular way. The present treatment can be limited to the question of the goal-directed behavior as such.

In particular, a software module that makes interpretive use of the present set of axioms will output invocations of actions within plans; it will require several kinds of inputs, including the information about the success or failure of actions and plans, and information about the “moves” of the robot agent as defined in section 3.1. All of this information can be communicated as simple logic formulae.

Notice, in particular, that the module being described here does not need to be involved at the beginning and end of every elementary action in a plan; it is sufficient for it to be involved at the breakpoint where replanning takes place. Notice also that the replanning process itself is encapsulated in the specification of *Option*, which means that it is held open whether replanning is to be done by an inference engine or by some other process, for example an algorithmic process or lookup in a plan library. The only

important thing from our point of view is that the fourth argument of the predicate *Option* can be determined when the first three predicates are given, together with the relation H , and that the relation *Option* satisfies the criterium for producing correct plans. Naturally, if *Option* is implemented by an inference based method such as deduction or abduction, then it must make use of action laws expressed in logic, as usual.

In summary, we have now showed how the invocation of a goal-directed action can invoke one or more procedural actions, and how the proper reactions to the success or failure of the latter can be specified in logic. The key notion in this formalisation is that it does not explicitly prescribe *the* next action to be taken in a particular failure situation; the forward deductive machinery is able to derive a number of candidates. The axioms in section 3.3 assure that exactly one of those will be chosen, assuming of course that at least some action is implied to be considered and that there is no overriding command.

The success and failure of the procedural actions is in turn defined on the level of continuous or hybrid description. The previous article [11] addressed how to establish the deliberative-level description of actions, including both the success case and the failure case, as logical consequences of the hybrid-level description.

4 Entailment methods

The previous sections have described a logical machinery for goaldirectedness that requires the use of the following sets of axioms:

- The set **S**, consisting of the axioms S1 – S8 in section 2.3, together with axioms characterizing composite properties (section 2.5) and composite actions (section 2.6 and annex).
- The set **G**, consisting of the axioms G1 – G4 in section 3.3, together with the axioms in section 3.2.
- A set **R** of behavior rules, consisting of axioms using the predicates *Option* and *Realize* for specifying concrete behaviors.

However, it would not make sense to use them on a stand alone basis. They are intended to define goal-directed behavior in a deliberative context, where the following knowledge sources exist as well:

- A set **E** of action laws specifying the effects of actions when they succeed
- A set **A** of applicability laws, specifying when actions are applicable
- A flow **O** of observations, providing specific facts at specific points in time.
- A flow **D** of decisions by the robotic agent of the kinds specified in section 3.1.

If conventional logic were to be used, then it would be a trivial matter to combine these knowledge sources: one would merely take the union set of all the axioms, and use them for the deductive machinery. In the present set, however, the matter is more complicated since nonmonotonicity is involved. In particular, it is well known that the action laws **E** need to be used in a nonmonotonic context, if they are written in a reasonable way. We have also observed in an earlier paper [11] that the proper treatment of action

failure as a non-standard way of terminating actions requires the use of another kind of nonmonotonicity, and we have observed in subsection 3.4 of this article that minimization of actions calls for yet another kind of nonmonotonicity, at least when concurrency is involved. The question of how to combine the above mentioned knowledge sources is therefore not at all obvious. The present section will provide an answer to this important problem.

4.1 A model example

The general formulation of the problem at hand is the following: *given two or more logical knowledge sources, where each of them specifies some aspect of the dynamic behavior of a system, and where these aspects are interdependent so that the changes imposed by one knowledge source influences the continued development described by the other(s), how are those knowledge sources to be combined in the framework of nonmonotonic logics?* In order to address this question, we first describe a very simple case where the approach can be brought out clearly.

Consider therefore a system where there are two multi-valued fluents a and b for discrete time, and two knowledge sources A and B . A specifies the value of a at the next time-step depending on the values of a and b at the previous time-step, and B specifies the new value for b in the same way. Then, set up the logic so that each interpretation is a mapping from timepoints to corresponding values for a and b . In other words, each interpretation is a possible history of the world. Let $M(A)$ be the set of all interpretations *for arbitrary assignments to b* , and where each interpretation specifies the successive values for a according to its previous value and the value at hand for b . Let $M(B)$ be similar for the fluent b . Obviously, $M(A) \cap M(B)$ is the set of all histories of the world that develop according to the joint information of the two knowledge sources.

Suppose further that A and B are such that they need to be used in the context of a nonmonotonic logic. For example, A may be characterized by inertia or persistence, so that it is a set of rules specifying when the value of the fluent changes; there is a background assumption that if A does not specify any change, then the fluent stays constant from one timepoint to the next. It is well known that in this case, $M(A)$ can be conveniently written e.g. as $Min(<_a, Mod(A))$ where $Mod(A)$ is the set of classical models of A , $<_a$ is a preference relation on models which prefers inertia in the a component between models having the same b component, and Min is an operator reducing a set of models to the subset consisting of those members that are minimal with respect to the ordering in the first argument.

Suppose similarly that the knowledge source B has been written using the assumption of a normal value, so that the value of b at any time shall be the normal value unless a rule in B implies otherwise. This case can be dealt with by an approach similar to the one for A , except that another preference relation must be used.

It is now straightforward to see that the two knowledge sources, each having its own nonmonotonic entailment method, can be combined and that the set of selected models for the combination of A and B ought to be

$$Min(<_a, Mod(A)) \cap Min(<_b, Mod(B))$$

With this insight, we can return to the case at hand.

4.2 Entailment method for goal-directed behavior: simple case

We restrict our attention to the case where the agent only invokes goal-directed actions. It does not make an explicit choice between options; that choice is modelled as random (meaning that all choices are obtained as models), and the agent also does not fail actions on any level. Action failure is obtained as observations, that is, from the world at hand. Generalization to the case of more complex agent interactions appears to be fairly straightforward, and is planned to follow in a later contribution.

Interpretations are constructed as fourtuples $\langle H, D, X, D_b \rangle$ in the obvious fashion. Furthermore, in each such interpretation, the H component is partitioned into four parts,

$$H = H_{ord} \cup H_{inv} \cup H_{fail} \cup H_{app}$$

where H_{inv} contains value assignments for fluents of the form $inv(a)$, similarly for H_{fail} and H_{app} , and H_{ord} contains value assignments for all other fluents. In line with the approach of the previous subsection, we proceed as follows:

- One set of models is constructed by allowing H , X , and D_b to vary freely, except H_{app} , and using the axiom sets \mathbf{S} (characterizing invocation of actions) and \mathbf{A} (applicability of actions) for properly constraining H_{app} and D .
- Another set of models is constructed by allowing H_{inv} , H_{app} , D , and D_b to vary freely and using the axiom set \mathbf{E} (action laws) for obtaining proper execution of actions and proper effects of actions in all different cases of invocation that may arise. This will constrain X , H_{ord} , and H_{fail} .
- Another set of models is constructed allowing free variation of all components except H_{inv} and D_b , and using the knowledge sources \mathbf{D} , \mathbf{R} , and \mathbf{G} in order to constrain H_{inv} and D_b .
- A final set of models is constructed as the classical model set for the knowledge source \mathbf{O} , that is, the observations. It will only constrain H_{ord} .

The intersection of the model sets obtained in these four ways are clearly the desired ones, provided that each of the participating sets is selected correctly. For the first case, this is straight-forward. The second case is one which has already been studied extensively, and a catalogue of different entailment methods and their respective properties has been published in [9]. The third case is the one that needs to be further considered here; the fourth case is trivial.

But considered in this light, the solution to the selection of model set with respect to H_{inv} is also straight-forward: it must be merely a question of minimizing H_{inv} chronologically. In other words, at each point in time, only those actions are invoked whose invocation is necessary due to the given axioms and the situation at hand. Certainly there may be more than one choice of action to be invoked, and this is represented by obtaining several models.

The chronological minimization of H_{inv} at a timepoint t is of course done separately for different histories of H_{ord} , D_b , etc. up to that time. It is intended that the proposed axioms shall characterize exactly the intended

occurrences of D_b for given H etc, so no minimization of D_b ought to be required. The use of nochange axioms instead of chronological minimization seems to be straightforward for the non-concurrent case but problematic for the concurrent case, for the reasons that were discussed in subsection 3.4.

5 Discussion

5.1 Limitations of these results

The present treatment is incomplete in a number of ways, in particular:

- We have not yet treated the case of more general agent behavior, where the agent may discontinue goals and specific actions being executed towards the goals.
- We have not yet given complete treatment to the case of concurrent actions, although the formalism allows it and some parts of the present material also allows for that case.
- One particular aspect of the question of concurrent actions is that some of the axioms specified above do not completely characterize what must happen when the goal-directed process proposes to invoke an action that is already in the midst of executing in service of another goal.
- The formulation of a plausible underlying semantics, and a validation of the present approach with respect to that semantics has not yet been undertaken. This also means that the axiomatization proposed here is preliminary and is to be taken as a first proposal which needs to be subjected to additional “debugging”.

5.2 Continued work

The present article has been structured for electronic publication, and we will use it as a test example in that respect. In particular, we intend to use the present article as a base, and to report some of the continued work as extension notes rather than as separate, self-contained papers. We observe that the use of extension notes, which depend strongly on the underlying base article, is much more manageable in the electronic forum than what it was in paper-based publication, simply because it is now so easy to click one’s way from one article to another. The evolving structure of articles and extension notes will be accessible from the URL of the present article, which is quoted on the front page.

5.3 Conclusion

We have described a way of characterizing goal-directed robotic behavior by moderate extensions of a current logic of actions and change. In particular, our logical system is able to characterize the sequencing and the choice of successive sub-actions which are performed in order to achieve a given goal, and to infer the success or the failure of the overall goal from the success or failure of the sub-actions. We have also proposed a nonmonotonic entailment method for using the axiomatisation of the present article in conjunction with other, related knowledge sources.

We consider that work on these problems is a necessary step on the way to designing intelligent robotic agents wthat can be analyzed formally and for which one can prove some of their properties in a systematic fashion.

RELATED WORK

While the previous, ARTICLE section contained the substance of the present paper, the present section discusses how those results relate to other research. Its contents are also available in HTML form with clickable links to some of the cited articles. It is intended that the contents of the present section shall be extended and modified over time as additional related works are pointed out to us. For an up-to-date, on-line version of the present section, please use the URL mentioned on the front page of the present paper.

The present version of this section is incomplete.

The material presented in section 2 (D_s and D_f predicates and the definition of composite actions in that context) has been presented in an earlier paper by ourselves [11]. The same holds for the general approach to logics with explicit time, which has been systematically described in [9].

The previous work that is the closest to the present one in spirit is the work by Kabanza et al [2]. Their goals differ from ours in the sense that they study safety and liveness constraints, in the tradition of the theory of real-time systems. This leads them to an approach based on modal-temporal logic which is considerably more complex than ours. On the other hand, they only consider the world in terms of discrete state-transition functions, and therefore they do not have the grounding of actions (including action failure) in terms of physical models. This connection is an important aspect of our approach.

Kabanza's work is relatively recent. The topic addressed here is related to two strands of earlier research. One is the work on architectures for rational agents, in particular those contributions which attempted to characterize the architecture in precise and formal ways. Rao and Georgeff [5] have developed an abstract architecture for rational agents based on the concepts of belief, desire, and intention. Their architecture is defined in terms of an abstract interpreter and a formal language that is used for expressing beliefs, intentions, etc. The language includes a number of modal operators which lack a counterpart in our system, such as a belief operator and a concept of "inevitable". It also uses some constructs that we have inherited, in particular the distinction between success and failure of an action. However, their system does not apply the concept of success and failure to the achievement of a goal, and in general it lacks the additional expressiveness that is obtained by treating goals as goal-directed actions.

Tate's O-Plan2 system [15] is an extensive architecture for plan-guided systems, whose behavior is of course considerably more complex than has been modelled in our approach so far. However, O-Plan2 is only described in terms of its computational processes.

The other relevant strand of earlier work is in the area of goal-directed and plan-based multi-agent systems. In this case, the focus is on the modalities of knowledge and belief, and on communication and knowledge acquisition actions. Physical actions such as the robotic actions that we have discussed here are barely present or not at all. In particular, Shoham [14] has introduced a logical system for agent-oriented programming, and Levesque et al [3, 4] introduce an alternative, logical approach to agent programming. These approaches are more or less complementary to ours in the sense that a complete system should include both modal constructs (which they have

and we don't) and actions in the physical world as well as a capability for reasoning about the achievement or non-achievement of goals.

In more general terms, it is clear that the behaviors obtained from practical software architectures for autonomous agents, such as the RAPs (Reactive Action Packages) approach of Firby et al [1] are well in line with the behaviors characterized by our approach.

References

- [1] R. James Firby, Roger E. Kahn, Peter N. Prokopowicz, and Michael J. Swain. An architecture for vision and action. In *International Joint Conference on Artificial Intelligence*, pages 72–79, 1995.
- [2] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95(1):67–113, 1997.
- [3] Yves Lespérance et al. Foundations of a logical approach to agent programming. In *Proc. IJCAI 95 Workshop on Agent Theories, Architectures, and Languages*, 1995.
- [4] Hector J. Levesque. What is planning in the presence of sensing? In *National Conference on Artificial Intelligence*, 1996.
- [5] Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. In *International Conference on Knowledge Representation and Reasoning*, pages 439–449, 1992.
- [6] Erik Sandewall. A planning problem solver based on look-ahead in stochastic game trees. *Journal of the ACM*, 16(3), July 1969.
- [7] Erik Sandewall. Combining logic and differential equations for describing real-world systems. In *Proc. International Conference on Knowledge Representation, Toronto, Canada*, 1989.
- [8] Erik Sandewall. Filter preferential entailment for the logic of action in almost continuous worlds. In *International Joint Conference on Artificial Intelligence*, pages 894–899, 1989.
- [9] Erik Sandewall. *Features and Fluents. The Representation of Knowledge about Dynamical Systems. Volume I*. Oxford University Press, 1994.
- [10] Erik Sandewall. Assessments of ramification methods that use static domain constraints. In *International Conference on Knowledge Representation and Reasoning*. Morgan Kaufmann, 1996.
- [11] Erik Sandewall. Towards the validation of high-level action descriptions from their low-level definitions. *Artificial Intelligence Communications*, December 1996. Also Linköping University Electronic Press, <http://www.ep.liu.se/cis/1996/004/>.
- [12] Erik Sandewall. Relating high-level and low-level action descriptions in a logic of actions and change. In Oded Maler, editor, *Hybrid and Real-Time Systems*, pages 3–17. Springer Verlag, 1997.
- [13] Erik Sandewall and Ralph Rönquist. A representation of action structures. In *National Conference on Artificial Intelligence*, pages 89–97, 1986.
- [14] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.
- [15] Austin Tate. The emergence of standard planning and scheduling system components. In Christer Bäckström and Erik Sandewall, editors, *Current Trends in AI Planning*. IOS Press, 1994.