1 Topic and approach

The purpose of the present work is to characterize the behavior pattern of *deliberated retry* in logicist terms. Deliberated retry is where an agent pursues goals by trying actions or action sequences that are likely to achieve the goal, and where the agent responds to failure by trying an alternative plan or action sequence. (We identify plans with action sequences). The paper proposes a first-order theory using a narrative time-line approach in which each model is a history of the world where the agent's successive actions exhibit deliberated retry. This work is intended to be used in the design of autonomous agents, and in particular in our WITAS project which is concerned with the control system of an intelligent airborne vehicle (UAV).

A logic of this kind must of course build on existing work in logics of actions and change, but it also imposes some specific requirements on the host logic. Besides the characterization of deliberated retry, which is addressed in this article, it must also characterize the success and failure of actions, it must account for exogenous events that influence the execution of an action, and it must allow actions to be described on different levels of resolution. The present section discusses the integration of these different aspects of the entire agent architecture on the logic level.

1.1 Goals vs high-level actions

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. By this approach we avoid the need to introduce goals as a separate type of entities, and we introduce the possibility (although it will not be used in the present paper) of having goals and actions on several levels.

A goal-directed action is then viewed as 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.

1.2 Additional requirements on a logic of autonomous agent behavior

A logic that describes the behavior of an autonomous agent, such as the WITAS vehicle, must have the following expressiveness besides characterizing goal-directed behavior:

- *Hierarchical actions.* This is a natural requirement for any nontrivial robotic behavior. It is also technically necessary when goals are represented as high-level actions, as described in the previous subsection.
- *Concurrent actions.* In a hierarchical action structure, a high-level action is concurrent with its lower-level realizations.
- *Success and failure of actions.* It is when an action fails that the goal-directed behavior needs to "try again".
- *Hybrid specification of actions,* that is, specifications where both continuous and discrete change is accounted for. This is a necessity in any robotic system that has to handle continuously changing resources, such as "remaining fuel" and "remaining battery power".
- *Multi-level specification of actions,* where one and the same action is described both in a concise way (for example, only preconditions and postconditions) and in a detailed way (for example, successive maneuvers). The detailed specification will often be a hybrid one and include characterization of piecewise continuous change.
- *Exogenous events,* that is, events that are due to phenomena in the robot's environment, and that are not the direct results of the robot's actions. In multi-level action specifications, low-level specifications tend to obtain their high degree of detail because they must specify reactions to exogenous events. Also, exogenous events are often the reason for action failure, so a characterization of goal-directed behavior, with its requirement to characterize action failure, must also characterize exogenous events.

For example, consider an intelligent UAV that is to fly over traffic scenes, and that is required to 'understand' what happens in those scenes, and to take appropriate action towards certain goals, such as for assisting one or more cooperating ground vehicles in their missions. Such a robotic system would require all the features in the list. The application 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 where you can land". There are quite a number of exogenous events that may cause the failure of an action in this scenario or require it to be replanned, for example, hostile actions of the ground vehicles, obstructions to vision or to communication, restrictions on fuel or other resources in the UAV, and so on.

Several of our earlier articles contain results that complement the present article in these respects. 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 [8] and how to combine it with specifications of the effects of actions [9]. 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 and for relating the global, precondition/ postcondition type description of the action to the one on the hybrid level [11, 12].

In the present article we only consider the relationship between lower-level and higher-level actions *inside the deliberative layer*, whereas the previous articles showed how the hybrid, low-level specification of an action may be interpreted as a controller that reacts to external events. Those methods and the methods described here will eventually need to be integrated. The more limited scope of the present article is in order to take one thing at a time.

There is yet another way that exogenous events may influence a system of the kind considered here, namely, by causing the system to adopt additional goals. This aspect is not directly addressed in the present work, because of an architectural consideration. The pattern of deliberated retry, which is modelled here, is viewed as a behavioral process that takes inputs of two kinds: (a) the decisions of a higher, deliberative layer in the robot, (b) indications from lower layers in the robot to the effect that the execution of a particular action has succeeded or failed. It is the role of low-level action specifications to indicate when a certain exogenous event causes an action to fail. Correspondingly, it is the role of the higher, deliberative layer to adopt new goals according to exogenous events in combination with general policies or desires. (In addition. the higher layer may decide to "fail" an action in the sense of giving it up).

2 Representing success and failure

A logic for characterizing goal-oriented behavior must be able to express that an action has 'succeeded' or 'failed', since a very important aspect of such behavior is that the agent should try again when one attempt to achieve the given goal has failed. We obtain our logic by extending an existing, narrative time-line logic[1] with a few additional constructs. The present section describes this logic in reasonably precise terms while omitting some of the routine and tedious details.

2.1 Standard Time and Action Logic

The following is the basic notation that we inherit from earlier work. Three primary predicates are used. The predicates H for *Holds* and D for *Do* are defined as follows. $\mathsf{H}(t, p)$ says that the "propositional fluent" ([2]) p holds at time t. In other words, p is reified and $\mathsf{H}(t, p)$ is the same as $p(t)$ in the case where p is atomic. $\mathsf{D}([s, t], a)$ 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 $\mathsf{H}(t, f\!:\!v)$ where $f\!:\!v$ is the proposition saying that the fluent f has the value v. Fluent-

[1] A first-order, multi-sorted logic where time, represented by real numbers, is one of the sorts.

[2] We 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 word 'fluent' both for the function and its designator, we follow that practice here.

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 $\mathsf{H}(1998, ageof(john) : 36)$.

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. $\mathsf{X}(s, f)$ 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[3]. 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 $\mathsf{H}(t,p)$ and $\mathsf{D}([s,t],a)$, respectively. The change is made in order to emphasize more strongly that we are dealing with a fairly standard first-order logical theory.

2.2 Ontology for invocation and success

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. To a first approximation, *invocation* of an action causes it to begin its *execution*, which ends with either *success* or *failure*. However, the matter is complicated by the requirement to represent that it is sometimes impossible to execute an action. In our approach, invocation of an action is possible at any time, but the invocation does not necessarily lead to the execution of the action. In particular, it does not 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. (The reasons for these ontological choices will be briefly explained below).

2.3 Syntax for invocation and success

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

[3]However, it also appears that it is straightforward to generalize the time domain so that it also accounts for the case of branching time.

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

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

fail, where $\mathsf{H}(t, fail(a))$ says that the action a terminated with failure at time t. $\mathsf{H}(t, fail(a))$ 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 $\mathsf{D}([s, s], test(p)))$, and that satisfies

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

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

The following abbreviations are introduced:

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

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

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

$\mathsf{D}_{\mathrm{f}}([s, t], a)$ for $\mathsf{D}([s, t], a) \wedge \mathsf{H}(t, fail(a))$: 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.

$\mathsf{D}_{\mathrm{c}}([s, t], a)$ for $\exists u[\mathsf{D}([s, u], a) \wedge t \leq u]$: 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).

$\mathsf{D}_{v}(s, a)$ for $\mathsf{G}(s, a) \wedge (\neg\mathsf{H}(s, app(a)) \vee \exists s'\exists t[\mathsf{D}([s', t], a) \wedge s' < s < t])$: 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).

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.

2.4 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:

$$\mathsf{D}([s, t], a) \rightarrow \mathsf{H}(s, inv(a)) \wedge \neg\mathsf{D}_{v}(s, a)$$

This implies:

$$\mathsf{D}([s, t], a) \rightarrow \mathsf{H}(s, inv(a)) \wedge \mathsf{H}(s, app(a)) \wedge$$

$$\neg\exists s'\exists t[\mathsf{D}([s', t], a) \wedge s' < s < t]$$

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

$$\mathsf{H}(s, inv(a)) \rightarrow \exists t[s \leq t \wedge \mathsf{D}([s,t], a)]$$

$$\vee \mathsf{D}_v(s,a) \vee Composite(a)$$

The predicate *Composite* will be introduced in subsection 2.7; 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 invocation of an action does not result in its execution.

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

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

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

$$\mathsf{H}(s, app(test(p)))$$

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

S6. All other actions execute over extended periods of time: never immediately, except for actions of the form $test(p)$:

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

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

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

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

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

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 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 $\mathsf{D}([s_i, t_i], a)$ 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 $\mathsf{H}(t, fail(a))$ which invocation the failure referred to. This consideration is also the reason for the choice manifested in axiom 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* $\mathsf{D}([s,t],a)$ *and* $\mathsf{H}(u, fail(a))$ *for some* u *in* $(s,t]$, *then* $t = u$.

Conversely, if $\mathsf{D_s}([s,t],a)$, then $\mathsf{H}(u, fail(a))$ 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 $\mathsf{H}(u, fail(a))$ 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 $\mathsf{H}(u, fail(a))$ 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.

2.5 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:

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

Ordinary action laws specify the action's effects when it succeeds. They are therefore written as usual and with $\mathsf{D_s}$ 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:

$\mathsf{A}(s,a) \leftrightarrow \mathsf{H}(s,\varphi_a)$
$\mathsf{D_s}([s,t],a) \rightarrow \mathsf{H}(t,\psi_a \wedge \omega_a)$
$\mathsf{A}(s,a) \wedge \mathsf{D_c}([s,t],a) \wedge u \in [s,t) \rightarrow \mathsf{H}(u,\psi_a) \wedge \neg\mathsf{H}(u,\omega_a)$
$\mathsf{D_c}([s,t],a) \wedge \neg\mathsf{H}(t,\psi_a) \rightarrow \mathsf{D_f}([s,t],a)$

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

2.6 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

$$\mathsf{H}(s,p \wedge p') \leftrightarrow \mathsf{H}(s,p) \wedge \mathsf{H}(s,p')$$

and similarly for the other operators.

2. Composite fluents are always occluded, so that

$$\mathsf{X}(s, p \wedge p')$$

and similarly for the other operators. 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.

2.7 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, $\mathsf{D_s}$, etc. This is as follows for ;

$$\begin{array}{l}
\mathsf{G}(s, a_1; a_2) \rightarrow \mathsf{G}(s, a_1) \wedge \\
\quad (\mathsf{D}_v(s, a_1) \rightarrow \mathsf{D}_v(s, a_1; a_2)) \wedge \\
\quad (\mathsf{D_f}([s,t], a_1) \rightarrow \mathsf{D_f}([s,t], a_1; a_2)) \wedge \\
\quad (\mathsf{D_s}([s,t], a_1) \rightarrow \mathsf{G}(t, a_2) \wedge \\
\quad\quad (\mathsf{D}_v(t, a_2) \rightarrow \mathsf{D_f}([s,t], a_1; a_2)) \wedge \\
\quad\quad (\mathsf{D_f}([t,u], a_2) \rightarrow \mathsf{D_f}([s,u], a_1; a_2)) \wedge \\
\quad\quad (\mathsf{D_s}([t,u], a_2) \rightarrow \mathsf{D_s}([s,u], a_1; a_2)))
\end{array}$$

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

$$\begin{array}{l}
\mathsf{G}(s, try(a_1, a_2)) \rightarrow \mathsf{G}(s, a_1) \wedge \\
\quad (\mathsf{D_s}([s,t], a_1) \rightarrow \mathsf{D_s}([s,t], try(a_1, a_2)) \wedge \\
\quad (\mathsf{D_f}([s,t], a_1) \rightarrow \mathsf{G}(t, a_2) \wedge \\
\quad\quad (\mathsf{D}_v(t, a_2) \rightarrow \mathsf{D_f}([s,t], try(a_1, a_2)) \wedge \\
\quad\quad (\mathsf{D_f}([t,u], a_2) \rightarrow \mathsf{D_f}([s,u], try(a_1, a_2)) \wedge \\
\quad\quad (\mathsf{D_s}([t,u], a_2) \rightarrow \mathsf{D_s}([s,u], try(a_1, a_2)))) \wedge \\
\quad (\mathsf{D}_v(s, a_1) \rightarrow \mathsf{G}(s, a_2) \wedge \\
\quad\quad (\mathsf{D}_v(s, a_2) \rightarrow \mathsf{D}_v(s, try(a_1, a_2)) \wedge \\
\quad\quad (\mathsf{D_f}([s,t], a_2) \rightarrow \mathsf{D_f}([s,t], try(a_1, a_2)) \wedge \\
\quad\quad (\mathsf{D_s}([s,t], a_2) \rightarrow \mathsf{D_s}([s,t], try(a_1, a_2))))
\end{array}$$

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, $\mathsf{D_s}$ etc are not in themselves the primary ones. The Annex (see URL at the beginning of the article) contains an axiomatization that relates ; and try to inv, app, $fail$, and $test$ that is easier to work with, 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

$$\mathsf{D_s}([s,s], test(p); test(q)) \leftrightarrow$$

$$\mathsf{D_s}([s,s], test(p)) \wedge \mathsf{D_s}([s,s], test(q))$$

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 Deliberated retry

We proceed now to the phenomenon of *deliberated 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 deliberated 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 2.7. 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.

With this approach, goals can simply be viewed as high-level actions. This is a difference from earlier work which sometimes used a modal operator for specifying goals. Here, we manage with a simpler approach. The variable symbol a will be used for low-level actions, and g for high-level actions, or goals. When explicitly or implicitly quantified, these two variables will 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.

The goal-directed process is a deductive one, where successive conclusions are drawn from the available set of premises. As was discussed in section 1.2, this process can be influenced both by the robot agent and by lower-level processes that execute actions according to their low-level specifications. In both cases, the influence is performed by adding or "stating" additional propositions to the set of premises in the goal-directed process. The robot agent can influence that process in the following ways:

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

Notice that in all cases, the agent exercises its control by making statements of the form $\mathsf{H}(t, p)$ where p has the property of being true at singular points in time, and false everywhere else. It is appropriate to think about such fluents as *signals*.

It is assumed that the agent 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 2.7.

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

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.

In implementation terms, the predicate *Option* encapsulates the system's replanner: if the high-level action g was invoked at time s', and one attempt to achieve the goal has just failed at the present time s, then $Option(s', g, s, a)$ shall be true for exactly those (composite) actions a that will take the robot the remaining distance to the given goal. Similarly, $Realize(s, g, a)$ shall be true for at most one a for given s and g, namely for that a which is the standard way of achieving goal g from the state of the world at time s.

What 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

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

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

Finally, we need one more variant of the D predicate, this one defined directly and not as an abbreviation. The formula $\mathsf{D}_{\mathrm{b}}([s, t], g)$ 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 $\mathsf{G}(s, g)$ is true, $\mathsf{D}_{\mathrm{b}}([s, t], g)$ 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 it first; in this paper we limit our attention to it. The ontology described above is characterzied by the following axioms, in addition to those defined in earlier sections.

G1. $\mathsf{G}(s, g) \rightarrow \mathsf{D}_{\mathrm{b}}([s, s], g)$

G2. $\mathsf{D}_{\mathrm{b}}([s, t], g) \wedge \forall a[\neg Option(s, g, t, a)] \rightarrow$
$(s = t \rightarrow \mathsf{D}_v(s, g)) \wedge$
$(s < t \rightarrow \mathsf{D}_{\mathrm{f}}([s, t], g))$

G3. $\mathsf{D}_{\mathrm{b}}([s, t], g) \wedge Option(s, g, t, a) \rightarrow$
$\mathsf{D}_{\mathrm{f}}([s, t], g) \vee$
$\exists a'[\mathsf{G}(t, a') \wedge Option(s, g, t, a') \wedge$
$(\mathsf{D}_{\mathrm{s}}([t, t'], a') \rightarrow \mathsf{D}_{\mathrm{s}}([s, t'], g)) \wedge$

$$(\mathsf{D}_{\mathrm{f}}([t,t'],a') \rightarrow \mathsf{D}_{\mathrm{b}}([s,t'],g))]$$

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 possiblity 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. $\mathsf{D}_{\mathrm{f}}([s,t'],g) \wedge Option(s,g,t,a) \rightarrow$
$(\mathsf{D}_{\mathrm{c}}([t,t'],a) \rightarrow \mathsf{D}_{\mathrm{f}}([t,t'],a))$

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 an *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 are 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 is likely to become 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 The role of deliberated retry in the overall architecture of the robotic agent

The present section has descrbied the logical structure of the agent's subsystem for deliberated retry, that is, for pursuing goals by finding a method, and trying again if that method fails. Subsection 1.2 described other parts of the overall logical architecture that the deliberated retry behavior needs to interact with.

A subsystem for *planning* and *plan revision* is obviously required as an additional subsystem in such an architecture. That subsystem is represented by the relation *Options* in the present work, which means that it is held open whether replanning is to be done by e.g. an inference engine, 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.

We have indicated how the process specified here obtains its inputs as formulae, provided to it by the higher deliberative layer of the robot and by the low-level execution of actions. The outputs of the present process are likewise communicated as formulae, in particular, by formulae requesting the invocation of a particular action. Since these actions may be composite ones, or plans, it means 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.

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 goal-directedness that requires the use of the following sets of axioms:

- The set **S**, consisting of the axioms S1 – S8 in section 2.4, together with axioms characterizing composite properties (section 2.6) and composite actions (section 2.7 and annex).
- The set **G**, consisting of the axioms G1 – G4 in section 3.3, together with the two 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 setting, 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 up to the timepoint of comparison, 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 sixtuples $\langle \mathsf{H}, \mathsf{D}, \mathsf{X}, \mathsf{D_b}, Option, Realize\rangle$ in the obvious fashion. Furthermore, in each such interpretation, the H component is partitioned into four parts,

$$\mathsf{H} = \mathsf{H}_{ord} \cup \mathsf{H}_{inv} \cup \mathsf{H}_{fail} \cup \mathsf{H}_{app}$$

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 $\mathsf{D_b}$ to vary freely, except H_{app}, and using the axiom sets **S** (characterizing invocation of actions) and **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 $\mathsf{D_b}$ to vary freely and using the axiom set **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 $\mathsf{D_b}$, and using the knowledge sources **D**, **R**, and **G** in order to constrain H_{inv}, $\mathsf{D_b}$, *Option*, and *Realize*.
- A final set of models is constructed as the classical model set for the knowledge source **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 catalog of different entailment methods and their respective properties has been published in [10]. The third case is the one that needs to be further considered here. The fourth case requires no minimization of models, and is in that sense 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}, $\mathsf{D_b}$, etc. up to that time. It is intended that the proposed axioms shall characterize exactly the intended occurrences of $\mathsf{D_b}$ for given H etc, so no minimization of $\mathsf{D_b}$ 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 allow 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 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 that can be analyzed formally and for which one can prove some of their properties in a systematic fashion.

6 Related work

The material presented in section 2 ($\mathsf{D_s}$ and $\mathsf{D_f}$ 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 [10]. The approach to representing actions with extended duration by an instantaneous invocation, an instantaneous termination, and possibly some additional, instantaneous events within the course of the action, is also used in the "new situation calculus" work of Levesque, Reiter et al [6].

A recent workshop article by De Giacomo, Reiter, and Soutchanski [2] describes an approach to execution monitoring using the framework of the previous article. The purpose of their system is to take corrective action after the failure of one step in a composite action (a "plan"), which is of course almost identically the same topic as in our present work. By comparison, their approach appears to be more computation-oriented than ours: they describe recovery in procedural terms, and they do not define the required entailment method. Apart from these differences of emphasis, there are many similarities between their approach and ours.

The work by Kabanza et al [3] has also partly similar goals to the present ones, but differs 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 the one proposed here. 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.

The contributions of Levesque, Reiter, et al and of Kabanza are relatively recent ones. The topic addressed here is also 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 [7] 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.

One important success criterion for a logicist approach to agent behavior is that it ought to be able to characterize those behaviors that have been implemented and found useful in practical agent architectures. It is therefore no coincidence that the ontology and the agent behaviors in our approach shows many similarities with the one used in practical software architectures for autonomous agents, such as the RAPs (Reactive Action Packages) approach of Firby et al [1]. 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. 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 [4, 5] 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.

A more extensive discussion of related work can be found via the article's web page, which is referenced on the front page or in the publication history.

Acknowledgements

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

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] Giuseppe De Giacomo, Ray Reiter, and Mikhail Soutchanski. Execution monitoring of high-level programs. In *Working Papers of Common Sense'98*, pages 277–297, 1998.

[3] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95(1):67–113, 1997.

[4] Yves Lespérance et al. Foundations of a logical approach to agent programming. In *Proc. IJCAI 95 Workshop on Agent Theories, Architectures, and Languages*, 1995.

[5] Hector J. Levesque. What is planning in the presence of sensing? In *National Conference on Artificial Intelligence*, 1996.

[6] Hector J. Levesque, Raymond Reiter, Yves Lesérance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *jlp*, 31(1-3):59–84, April-June 1997.

[7] 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.

[8] Erik Sandewall. Combining logic and differential equations for describing real-world systems. In *Proc. International Conference on Knowledge Representation, Toronto, Canada*, 1989.

[9] 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.

[10] Erik Sandewall. *Features and Fluents. The Representation of Knowledge about Dynamical Systems. Volume I.* Oxford University Press, 1994.

[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 Rönnquist. 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.