Formal Modeling and Analysis of Cognitive Agent Behavior

Alexei Sharpanskykh (sharp@few.vu.nl) and Jan Treur (treur@few.vu.nl)

Department of Artificial Intelligence, Vrije Universiteit Amsterdam,
De Boelelaan 1081a, NL-1081 HV Amsterdam, The Netherlands
tel.: +31.20.598.7756
tel.: +31.20.598.7763
fax: +31.20.598.7653

Abstract

From an external perspective, cognitive agent behavior can be described by specifying (temporal) correlations of a certain complexity between stimuli (input states) and (re)actions (output states) of the agent. From an internal perspective the agent's dynamics can be characterized by direct (causal) temporal relations between internal, mental states of the agent. The latter type of specifications can be represented in a relatively simple, executable format, which enables different types of analysis of the agent's behavior. In particular, simulations of the agent's behavior under different (environmental) circumstances can be explored. Furthermore, by applying verification techniques, automated analysis of the consequences of the agent's behavior can be carried out. To enable such types of analysis when only given an external behavioral specification, this has to be transformed first into some type of executable format. An automated procedure for such a transformation is proposed in this paper. The application of the transformation procedure is demonstrated for a number of cases, showing examples of the types of analysis as mentioned for different forms of behavior.

Keywords: modeling of behavior, analysis, cognitive agents, simulation, verification, model checking

1. Introduction

The behavior of a cognitive agent can be considered both from an external and an internal perspective. From the external perspective, behavior of the agent can be described by temporal relationships of a certain complexity between its input (stimuli) and output (actions) states over time, expressed in some (temporal) language, without any reference to internal or mental states of the agent. Such relationships are called input-output correlations by Kim (1996, pp. 87-91). Within Philosophy of Mind such a view is considered within the perspective of behaviorism (Kim, 1996). The states of the agent are required to be publicly observable and the statements that describe these states should be intersubjectively verifiable (Heil, 2000). According to the apologists of behaviorism Watson (1913) and Skinner (1953), internal states of the agent (mental or inner states) are considered to be methodologically intractable and unnecessary, since they are based on a personal subjective experience and evaluations and can not be used for analysis and predictions of the agent behavior. Descriptions from an external perspective can be successfully used for modeling relatively simple types of behavior (e.g., stimulus-response behavior (Skinner, 1935)). For less simple types of behavior (e.g., adaptive behavior based on conditioning (Balkenius & Moren, 1999)) an external behavioral specification often consists of more complex temporal relations, relating behavior at a certain point in time to a possibly large number of inputs in the past (e.g., a training program), that can not be directly used for simulations or other types of analysis.

From the internal perspective the behavior of the agent can be characterized by a specification of more direct (causal) temporal relations between mental states of the agent, based on which an externally observable behavioral pattern is generated. Such a perspective is taken within functionalism (Kim, 1996). From this perspective mental states are described by their functional or causal roles. These can be specified in simple, executable formats. A mental state is characterized by its direct temporal or causal relations with input, output and other mental states. Functionalism was originally formulated by Putnam in terms of a 'Turing machine' (Putman, 1975), an abstract machine to give a mathematically precise definition of an algorithm or an automatic procedure. However, in general other executable (temporal) languages can be applied to specify functional roles.

From the viewpoint of analysis, executability is an important advantage of an internal specification over an external one. An external specification of behavior that is more complex than stimulus-response behavior, involves temporal expressions of a form of complexity that makes it

useless as a basis for any simulation experiment or other analysis method. Executable specifications, on the other hand, enable different types of automated analysis of the agent's behavior, for example, by simulations of different scenarios of the agent's behavior or by verifying certain global properties of an agent in its environment. To enable automated analysis of an external behavioral specification, the possibly complex temporal relationships between input states and output states over time have to be reformulated in terms of a simpler executable format. In practice, such a reformulation process is by no means trivial. For example, it may involve certain creativity concerning additional intermediate states that have to be postulated and direct temporal relations between such states that have to be hypothesized. Moreover, it is hard to ensure by human activity only that the reformulated specification is equivalent in a certain sense to the original one (and does not describe just another process). This paper focuses mainly on how this issue may be addressed.

The challenge addressed is to obtain a standard method for this reformulation process and to provide automated support for this method, with guaranteed outcome equivalent to the original specification. As a solution a standard procedure is proposed for (automated) transformation of an external behavioral specification first into a synthetic executable specification using postulated intermediate states, and subsequently into a general state transition system format. The executable specification is based on direct executable temporal relations between certain (postulated) states. These states play roles comparable to sensory representation memory states and preparation states of an agent. The type of internal memory states of an agent used (and shown to suffice) are memory states based on the agent sensing (observations) of objects and processes in his/her environment and of his/her own behavior (e.g., actions). Furthermore, it is postulated that before performing an action an agent creates an internal preparation state. While simple types of agent behavior (e.g., variants of stimulus-response behavior) are based on a limited number of (unrelated) internal states, more complex types (e.g., motivation-based, goal-directed, adaptive) require more complex patterns of temporal relations between (multiple) internal states. In the approach presented this is addressed by allowing the memory states to represent complex temporal relations. So, they are used not only to represent world states, but also temporal patterns that occurred in the past. In this way reasoning of an agent about his/her previous experience enables to generate proactive, motivation-based or goal-directed behavior as well.

The justification that the proposed transformation method indeed provides an executable specification which is equivalent to the original specification, is based on the theorem (see Section 5) that an external behavioral specification entails any dynamic property if and only if the generated executable internal specification entails the same property.

Based on the generated executable specification different types of (automated) analysis can be performed. First, a developed simulation software tool applied to the generated transition system specification of the agent's behavior can be used to generate traces representing changes of internal (mental) states and actions of the agent over time, according to different environmental scenarios. Second, the generated transition system specification is also useful to analyze the consequences of the agent's behavior under such environmental scenarios.

For a given specification of externally observable behavior, an interesting however not easily solvable problem is how to determine the (logical) consequences of this behavior in different environmental circumstances. For example, to which extent different types of behavioral repertoires of an animal situated in different food-related circumstances in the environment ensure (or entail) the animal's well-being. Within Computer Science, quite useful and efficient model checking techniques have been developed to determine consequences of a given system specification; e.g., (Clarke & Grumberg & Peled, 1999). By performing model checking it is possible to determine automatically if a system model, usually specified in a transition system format, entails some dynamic property, specified by more complex temporal formulae. Using model checking techniques this paper contributes an automated approach for analyzing the consequences of a behavioral specification of an agent in its environment. To be able to use model checking techniques, a behavioral specification has to be given in a simple, executable format (as a transition system). To address this issue the proposed approach includes an automated procedure for the transformation of executable specifications of agent behavior into the input format of the SMV model checking tool (McMillan, 1993) that is used for the analysis of logical consequences of the agent's behavior.

In the next section the concepts for formal modeling of externally observable agent behavior and for specifying an executable internal specification are introduced. Next, in Section 3 the transformation procedure from an external into an executable internal specification and subsequently into a general description of a finite state transition system is described in some detail. The explanation of the procedure is illustrated by a running example. After that the proposed approach is applied for a number of cases concerning different types of analysis of agent behavior. More specifically, in Section 4 simulation of different scenarios of agent behavior is considered, and in Section 5 an automated approach for the analysis of the consequences of an agent's behavior is described. The paper ends with a discussion.

2. Formal Modeling of Agent Behavior

From an external perspective an agent can be seen as an autonomous entity that interacts with a dynamic environment via its *input* and *output* (interface) states. At its input the agent receives observations from the environment whereas at its output it generates actions that can change a state of the environment.

2.1. States

An agent state at a certain point in time as used here is an indication of which of the state properties of the agent and its environment are true (hold) at that time point. Externally observable state properties of the agent are formalized as first-order predicate logic terms using the interaction state ontology InteractionOnt(A) (e.g., for observations and actions). In general, an ontology is defined as a specification (in order-sorted logic) of a vocabulary that comprises finite sets of sorts, constants within these sorts, and relations and functions over these sorts. A sort is a set of objects of the same type: e.g., the sort AGENT is the set of all particular instances of agents conceptualized in a model. InteractionOnt can be seen as a union of input and output state ontologies of the agent (resp., InputOnt and OutputOnt) to define corresponding input and output agent state properties. Generally speaking, an input ontology determines what types of information are allowed to be transferred to the input of an agent (or of the environment), and an output ontology defines what kinds of information can be generated at the output of an agent (or of the environment). InputOnt includes the unary predicate for specifying observations of agents called observed: STATPROP, where STATPROP is the set of all state properties defined using InputOnt. For example, for the constant tree the observation of an agent can be defined: observed(tree). For specifying the actions performed by agents the ontology OutputOnt includes the predicate performing_action: ACTION, where the sort ACTION consists of all actions that may be performed by agents. For example, the execution of the action open_window by an agent can be specified as performing_action(open_window).

2.2. Expressing Dynamic Properties

To characterize the dynamics of the agent, dynamic properties relate properties of states at certain points in time. Consider this externally observable behavior of an agent:

> "At any point in time if agent A observes food present at position p, then there exists a later point in time, at which agent A goes to p."

To express such dynamic properties, and other, more sophisticated ones, the Temporal Trace Language (TTL) is used (Jonker & Treur, 2002). TTL is a variant of order-sorted predicate logic (Manzano, 1996) and has some similarities with situation calculus (Reiter, 2001) and event calculus (Kowalski & Sergot, 1986).

The language TTL includes special sorts, such as: TIME (a set of linearly ordered time points), STATE (a set of all state names of an agent system), TRACE (a set of all trace names), and STATPROP (a set of all state property names). A trace can be seen as a temporally ordered sequence of states, i.e., a trajectory. Each state in a trace γ has a name and corresponds to a unique time point (e.g., state(γ, t1), state(γ, t2), etc). The sort TRACE contains a set of individual traces that describe particular developments (or histories) of an agent system.

Further in the paper we shall use t with subscripts and superscripts for variables of the sort TIME; and γ with subscripts and superscripts for variables of the sort TRACE.

A state of an agent is related to a state property via the satisfaction relation |= formally defined as a binary infix predicate (or by holds as a binary prefix predicate). For example, "in the output state of agent A in trace γ □□□at time t property p holds" is formalized by state(γ , t, output(A)) |= p. Sometimes, when the indication of an agent aspect is not essential, this relation will be used without the third argument: state(γ, t) |= p.

Both state(γ , t, output(A)) and p are terms of the TTL language. In general, TTL terms are constructed by induction in a standard sorted predicate logic way from variables, constants and functional symbols typed with TTL sorts. Dynamic properties are expressed by TTL-formulae defined by:

(1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROP, then holds(v_1, u_1) is an atomic TTL formula.

(2) If τ_1, τ_2 are terms of any TTL sort, then $\tau_1 = \tau_2$ is an atomic TTL formula.

(3) If t_1, t_2 are terms of sort TIME, then $t_1 < t_2$ is an atomic TTL formula.

(4) The set of well-formed TTL-formulae is defined inductively in a standard way based on atomic TTL-formulae using Boolean connectives and quantifiers.

2.3. External Behavior Specification

Typically a behavior specification from an external perspective consists of a number of dynamic properties that express how the agent copes with different situations it encounters in its environment. Here situations are meant as sequences (extending over time) of states or events. A simple example is the following dynamic property

"in any trace γ, if at any point in time t agent A observes that it is dark in the room, whereas earlier at t1 light was on in this room, then there exists a point in time t2 after t such that at t2 in the trace γ agent A switches on a lamp".

which is expressed in formalized form as:

[[state(γ, t, input(A)) |= observed(dark_in_room) & ∃t1<t [state(γ, t1, input(A)) |= observed(light_on)]

⇒ ∃t2 ≥ t state(γ, t2, output(A)) |= performing_action(switch_on_light)]

An external behavioral specification φ consists of a number of dynamic properties which have a format based on temporal patterns described by a past statement, interval statement and a future statement. For simplicity, the future part has a format that prevents non-determinism in behavior.

Definition (Past, Interval and Future Statements)

a) A *past statement* for a trace γ and a time point t over state ontology Ont is a temporal statement $\varphi_p(\gamma,t)$, such that each time variable different from t is restricted to the time before t: for every time quantifier for a time variable s a restriction of the form $s \leq t$, or $s < t$ is required within the statement.

b) A *future statement* for a trace γ and a time point t over state ontology Ont is a temporal statement $\varphi_f(\gamma,t)$, such that for every time quantifier for a time variable s, different from t a restriction of the form $s \geq t$, or $s > t$ is made.

c) An *interval statement* for a trace γ and time points t_1 and t_2 over state ontology Ont is a temporal statement $\varphi(\gamma, t_1, t_2)$ in TTL, that is a past statement for t_2 and a future statement for t_1.

Based on these definitions an external behavioral specification of an agent is defined as follows.

Definition (External Behavioral Specification)

An *external behavioral specification* for an agent system consists of dynamic properties $\varphi(\gamma, t)$ expressed in TTL of the form $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)]$, where $\varphi_p(\gamma, t)$ is a past statement and $\varphi_f(\gamma, t)$ is a future statement over the interaction ontology. The future statement is represented in the form of a conditional action: $\varphi_f(\gamma, t) \Leftrightarrow \forall t_1 > t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{act}(\gamma, t_1)]$, where $\varphi_{cond}(\gamma, t, t_1)$ is an interval statement over the interaction ontology, which describes a condition for some specified action(s) and $\varphi_{act}(\gamma, t_1)$ is a (conjunction of) future statement(s) for t_1 over the output ontology of the form $state(\gamma, t_1+c) \models performing\_action(a)$, for some integer constant c and action a.

When the past formula $\varphi_p(\gamma, t)$ is true, within γ at time t a potential to perform one or more action(s) occurs. This potential is actualized at time t_1 when the condition formula $\varphi_{cond}(\gamma, t, t_1)$ is true, which leads to the action(s) being performed in γ at the time point(s) t_1+c indicated in $\varphi_{act}(\gamma, t_1)$.

2.4. Internal Dynamics Specification

An executable specification of the internal dynamics of an agent consists of a set of executable dynamic properties, representing temporal relations between a number of postulated internal (or mental) states. Internal states of an agent A are described using a postulated internal state ontology InternalOnt(A). It is often assumed that an agent maintains a memory in the form of some internal model of the history; some even go that far to speculate that intelligence is mainly based on that; cf. (Hawkins, 2004). It is assumed that internal memory states are formed based on sensing (or observation). The state ontology InternalOnt includes sorts and functions for defining memories about input states. Notice that within such states representing memory, statements about time are made. To relate time within such a state property to time external to states, the function symbol present_time is used. Here the properties of correctness and uniqueness are assumed:

Uniqueness of time

This expresses that present_time(t) is true for at most one time point t:

$$\forall t, t'' \; state(\gamma, t) \models present\_time(t'') \Rightarrow \forall t', t' \neq t'' \; \neg state(\gamma, t) \models present\_time(t')$$

Correctness of time

This expresses that present_time(t) is true for the current time point t:

$$\forall t \; state(\gamma, t) \models present\_time(t)$$

As an example, memory(t, observed(a)) expresses that the agent has memory that it observed at time point t a state property a. Furthermore, it is postulated that before performing an action an agent creates an internal preparation state. For example, preparation_for(b) specifies preparation of an agent to perform action b. Both memory and preparation states belong to a type of states, which are much better understood and physically grounded in neurobiological research (Dudai, 1990) than intentional states such as beliefs, desires, and intentions. This motivates the choice that has been made with respect to the internal representations.

Each dynamic property in the internal specification of an agent's dynamics is specified in one of the *executable* forms given in Table 1.

Table 1: Executable Format

If a conjunction of state properties X holds in trace γ at time point t, *(Step Property)* $\forall t\ state(\gamma,t) \models X$	**then** a(nother) conjunction of state properties Y will hold in trace γ at time point t+c, with c>0 an integer constant $\Rightarrow\ state(\gamma, t+c) \models Y$
If a conjunction of state properties X and a condition property C hold in trace γ at time point t, *(Conditional Persistency Property)* $\forall t\ state(\gamma, t) \models X \wedge C$	**then** this conjunction of state properties X will hold in trace γ at the next time point $\Rightarrow\ state(\gamma, t+1) \models X$
If a conjunction of state properties X holds in trace γ at time point t, *(State Relation Property)* $\forall t\ state(\gamma, t) \models X$	**then** a(nother) conjunction of state properties Y will hold in trace γ at the same time point t $\Rightarrow\ state(\gamma, t) \models Y$

3. Transformation into Executable Format

The procedure described in this section achieves the transformation of an external behavioral specification for an agent into executable format and subsequently into the representation of a finite state transition system.

Let φ(γ, t) be a non-executable dynamic property from an external behavioral specification for agent A, expressed using ontology InteractionOnt(A), for which an executable representation should be found, then the transformation procedure is specified as follows.

The Transformation Procedure

(1) Identify executable temporal properties, which describe transitions from interaction states to memory states.

(2) Identify executable temporal properties, which describe transitions from memory states to preparation states for performing an action.

(3) Specify executable properties, which describe the transition from preparation states to the corresponding action performance states.

(4) From the executable properties, identified during the steps 1-3, construct a part of the specification π(γ, t), which describes the internal dynamics of agent A, corresponding to the property φ(γ, t).

(5) Apply the steps 1-4 to all properties in the external behavioral specification of the agent A. In the end add to the executable specification the dynamic properties, which were initially specified in executable form using an ontology, different than InteractOnt(A).

(6) Translate the identified during the steps 1-5 executable rules into the transition system representation.

The details of the described procedure are explained by means of an example, in which delayed-response behavior of a laboratory mouse is analyzed; e.g., Hunter (1912); Allen & Bekoff (1997).

Initial situation

The initial situation for the conducted experiment is as follows: the mouse is placed in front of a transparent screen that separates it from a piece of food that is put behind the screen. The mouse is able to observe the position of food and of the screen. At some moment after food has been put, a cup is placed covering the food, which makes food invisible for the mouse. After some time the screen is raised and the animal is free to go to any position. If the mouse comes to the position, where the food is hidden, then it will be capable to lift up the cup and get the food.

The behavioral specification for the conducted experiment consists of environmental properties and externally observable behavioral properties of the mouse. For the purposes of illustration of the proposed transformation procedure the dynamic property that describes the delayed-response behavior of the mouse has been chosen. Informally this property expresses that the mouse goes to the position with food if it observes that there is no screen and at some point in the past the mouse observed food and since then did not observe the absence of food. According to the definition of an external behavioral specification the considered property can be represented in the form $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)]$, where $\varphi_p(\gamma, t)$ is a formula

$$\exists t2<t\ [state(\gamma, t2, input(mouse)) \models observed(food) \wedge \forall t3,\ t \geq t3 > t2\ state(\gamma, t3, input(mouse)) \models not(observed(not(food)))]$$

and $\varphi_f(\gamma, t)$ is a formula

$$\forall t4 > t\ [state(\gamma, t4, input(mouse)) \models observed(not(screen)) \Rightarrow state(\gamma, t4+c, output(mouse)) \models performing\_action(goto\_food)\ ]$$

with $\varphi_{cond}(\gamma, t, t4)$ is

$$state(\gamma, t4, input(mouse)) \models observed(not(screen))$$

and $\varphi_{act}(\gamma, t4)$ is

$$state(\gamma, t4+c, output(mouse)) \models performing\_action(goto\_food)$$

where t is the present time point with respect to which the formulae are evaluated.

Step 1. ***From interaction states to memory states***

General idea

The formula $\varphi_{mem}(\gamma, t)$ obtained by replacing all occurrences in $\varphi_p(\gamma, t)$ of subformulae of the form state(γ, t') |= p by state(γ, t) |= memory(t', p) is called the *memory formula for* $\varphi_p(\gamma, t)$.

Thus, a memory formula defines a sequence of past events (i.e., a history) (e.g., observations of an external world, actions) for the present time point t. The time interval for generation of an internal memory state of an agent from its observation is assumed to be incommensurably smaller than time intervals between external events (i.e., stimuli). Therefore, in the proposed model both an observation state and a corresponding memory state are created at the same time point.

By a rewriting process (for the formal details for the considered procedure we refer to Sharpanskykh. & Treur (2005)) $\varphi_{mem}(\gamma, t)$ is equivalent to some formula $\delta^*(\gamma, t)$ of the form state(γ, t) |= $q_{mem}(t)$, where $q_{mem}(t)$ is called *the normalized memory state formula for* $\varphi_{mem}(\gamma, t)$, which uniquely describes the present state at the time point t by a certain history of events. Moreover, q_{mem} is the state formula $\forall t'$ [present_time(t') $\Rightarrow$ $q_{mem}(t')$].

Example

For the considered example $q_{mem}(t)$ for $\varphi_{mem}(\gamma, t)$ is specified as:

$\exists t2$ [memory(t2, observed(food)) $\wedge$
$\forall t3, t \geq t3 > t2$ memory(t3, not(observed(not(food))))]

Additionally, memory state persistency properties are composed for all memory atoms. For example, for the atom memory(t2, observed(food)) the corresponding persistency property is defined as:

$\forall t''$ state(γ, t'', internal(mouse)) |= memory(t', observed(food)) $\Rightarrow$
state(γ, t''+1, internal(mouse)) |= memory(t', observed(food))

Rules that describe creation and persistence of memory atoms are given in *the executable theory from observation states to memory states* $Th_{o\rightarrow m}$. For the considered example:

$\forall t'$ state(γ, t', input(mouse)) |= observed(food) $\Rightarrow$
state(γ, t', internal(mouse)) |= memory(t', observed(food))

$\forall t'$ state(γ, t', input(mouse)) |= not(observed(not(food))) $\Rightarrow$
state(γ, t', internal(mouse)) |= memory(t', not(observed(not(food))))

$\forall t'$ state(γ, t', input(mouse)) |= observed(not(food)) $\Rightarrow$
state(γ, t', internal(mouse)) |= memory(t', observed(not(food)))

$\forall t''$ state(γ, t'', internal(mouse)) |= memory(t', observed(food)) $\Rightarrow$
state(γ, t''+1, internal(mouse)) |= memory(t', observed(food))

$\forall t''$ state(γ, t'', internal(mouse)) |= memory(t', not(observed(not(food)))) $\Rightarrow$
state(γ, t''+1, internal(mouse)) |= memory(t', not(observed(not(food))))

$\forall t''$ state(γ, t'', internal(mouse)) |= memory(t', observed(not(food))) $\Rightarrow$
state(γ, t''+1, internal(mouse)) |= memory(t', observed(not(food)))

Step 2. ***From memory states to preparation states***

General idea

Obtain $\varphi_{cmem}(\gamma, t, t_1)$ by replacing all occurrences in $\varphi_{cond}(\gamma, t, t_1)$ of state(γ, t') |= p by state(γ, t_1) |= memory(t', p). The condition memory formula $\varphi_{cmem}(\gamma, t, t_1)$ contains a history of events, between the time point t, when $\varphi_p(\gamma, t)$ is true and the time point t_1, when the formula $\varphi_{cond}(\gamma, t, t_1)$ becomes true. Again by a rewriting process $\varphi_{cmem}(\gamma, t, t_1)$ is equivalent to the formula state(γ, t_1) |= $q_{cond}(t, t_1)$, where $q_{cond}(t, t_1)$ is called *the normalized condition state formula for* $\varphi_{cmem}(\gamma, t, t_1)$. Moreover, $q_{cond}(t)$ is the state formula $\forall t'$ [present_time(t') $\Rightarrow$ $q_{cond}(t, t')$].

Example

For the considered example $q_{cond}(t, t4)$ for $\varphi_{cmem}(\gamma, t)$ is obtained as: memory(t4, observed(not(screen))) and $q_{cond}(t)$: $\forall t'$ [present_time(t') $\Rightarrow$ memory(t', observed(not(screen)))].

Obtain $\varphi_{prep}(\gamma, t_1)$ by replacing in $\varphi_{act}(\gamma, t_1)$ any occurrence of state(γ, t_1+c) |= performing_action(a) by state(γ, t_1) |= preparation_for(action(t_1+c, a)), for some number c and action a. The preparation state is created at the same time point t_1, when the condition for an action $\varphi_{cond}(\gamma, t, t_1)$ is true. By Lemma 1 $\varphi_{prep}(\gamma, t_1)$ is equivalent to the state formula state(γ, t_1) |= $q_{prep}(t_1)$, where $q_{prep}(t_1)$ is called *the normalized preparation state formula for* $\varphi_{cond}(\gamma, t_1)$. Moreover, q_{prep} is the state formula $\forall t'$ [present_time(t')] $\Rightarrow$ $q_{prep}(t')$]. For the considered example $q_{prep}(t4)$ is composed as preparation_for(action(t4+c, goto_food)).

Rules, which describe generation and persistence of condition memory states, a transition from the condition to the preparation state, and the preparation state generation and persistence, are given in *the executable theory from memory states to preparation states* $Th_{m\to p}$. For the considered example:

$\forall t'$ state(γ, t', input(mouse)) |= observed(not(screen)) $\Rightarrow$
state(γ, t', internal(mouse)) |= [memory(t', observed(not(screen))) $\wedge$ stimulus_reaction(observed(not(screen)))]

$\forall t''$ state(γ, t'', internal(mouse)) |= memory(t', observed(not(screen))) $\Rightarrow$
state(γ, t''+1, internal(mouse)) |= memory(t', observed(not(screen)))

$\forall t'$ state(γ, t') |= $\forall t''$ [present_time(t'') $\rightarrow$ $\exists t2$ [memory(t2, observed(food)) $\wedge$ $\forall t3$, t'' $\geq$ t3 > t2 memory(t3, not(observed(not(food))))]] $\Rightarrow$

state(γ, t') |= ∀t''' [present_time(t''') → [∀t4 > t''' [memory(t4, observed(not(screen))) → preparation_for(action(t4+c, goto_food))]]]

∀t', t state(γ, t') |= [∀t''' [present_time(t''') → [∀t4 > t''' [memory(t4, observed(not(screen))) → preparation_for(action(t4+c, goto_food))]]] ∧

∀t'' [present_time(t'') → memory(t'', observed(not(screen)))] ∧ stimulus_reaction(observed(not(screen)))]

⇒

state(γ, t', internal(mouse)) |= ∀t4 [present_time(t4) → preparation_for(action(t4+c, goto_food))]

∀t' state(γ, t') |= [stimulus_reaction(observed(not(screen))) ∧ not(preparation_for(action(t'+c, goto_food)))]

⇒

state(γ, t'+1) |= stimulus_reaction(observed(not(screen)))

∀t' state(γ, t', internal(mouse)) |= [preparation_for(action(t'+c, goto_food)) ∧ not(performing_action(goto_food))] ⇒

state(γ, t'+1, internal(mouse)) |= preparation_for(action(t'+c, goto_food)).

The auxiliary atoms stimulus_reaction(a) are used to reactivate agent preparation states for generating recurring actions.

Step 3. ***From preparation states to action states***

General idea

The preparation state preparation_for(action(t_1+c, a)) is followed by the action state, created at the time point t_1+c. Rules that describe a transition from preparation to action states are given in *the executable theory from the preparation to the action state(s)* $Th_{p\to a}$.

Example

For the considered example the following rule holds:

∀t' state(γ, t', internal(mouse)) |= preparation_for(action(t'+c, goto_food)) ⇒

state(γ, t'+c, output(mouse)) |= performing_action(goto_food).

Step 4. ***Constructing an executable specification***

An executable specification π(γ, t) for agent A is defined by a union of the dynamic properties from the executable theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to a}$, identified during the steps 1-3. For the purposes of simulations of agent behavior the non-executable external behavioral specification is replaced by the executable behavioral specification.

Step 5. ***Constructing an executable specification for the whole external behavioral specification of an agent***

Other non-executable dynamic properties from the agent behavioral specification are substituted by executable ones by applying the same sequence of steps 1-4. In the end the executable properties for generating observation states from the states of the external world are added:

$\forall t'$ state(γ, t', world) |= [food $\wedge$ not(cup)] $\Rightarrow$
state(γ, t', input(mouse)) |= observed(food)

$\forall t'$ state(γ, t', world) |= [not(food) $\wedge$ not(cup)] $\Rightarrow$
state(γ, t', input(mouse)) |= observed(not(food))

$\forall t'$ state(γ, t', world) |= not(screen) $\Rightarrow$
state(γ, t', input(mouse)) |= observed(not(screen))

$\forall t'$ state(γ, t', world) |= screen $\Rightarrow$
state(γ, t', input(mouse)) |= observed(screen)

It is assumed that an observation state is generated at the same time point, when a corresponding state of the external world is active.

Step 6. ***Translation of an executable specification into a description of a transition system***

General idea

For the purposes of practical analysis (e.g., by performing simulation and verification) a specification based on executable temporal logical properties generated by the procedure described in the previous Section is translated into a finite state transition system model. The translation is based on the fact that a computation (in our case the execution of temporal logical properties) is essentially an (infinite) sequence of states (Vardi, 1996). Therefore, similarly to Vardi (1996), given an executable temporal specification one can construct a finite state transition system that generates the set of traces (by all possible executions of transition rules) equivalent to the set produced by all possible execution of temporal logical properties from the specification.

In computer science a finite state transition system is often described by a tuple $\langle Q, Q_0, \Sigma, \rightarrow \rangle$, where Q is a finite set of states of an agent, $Q_0 \subseteq Q$ is a set of initial states, Σ is a set of labels or events, which trigger the transition and $\rightarrow \subseteq Q \times \Sigma \times Q$ is a set of transitions. Such a representation often assumes an explicit denotation for every state in a transition system, which can be very numerous. However, a more compact representation, close to the production systems style, in the form of a set of transition rules with variables is possible (Arnold, 1994).

Definition (General Representation of a Finite State Transition System)
Let Ont be a state ontology consisting of sorts, constants, functions and predicates. Let At(Ont) be the set of (many-sorted predicate logic) atoms over Ont (possibly with variables). A general representation for a finite state transition system over Ont consists of transition rules of the form [P $\twoheadrightarrow$ N], where P is a proposition based on atoms from At(Ont), and N is a conjunction of atoms from At(Ont). The meaning is that when a certain instance of P by a certain variable assignment is true in a state, then the instance of N by the same variable assignment will be true in the next state; here $\twoheadrightarrow$ is a symbol for the transition between the two states.

Such a general representation for a finite state transition system has as an advantage that it does not depend on any particular implementation (e.g., verification or simulation tools). However, as this generic format describes states and transitions between them, it can be relatively easy translated into specialized languages of existing tools, based on the finite state transition system representation (e.g., the input format of the SMV model checker).

To translate the executable specification constructed at Step 5 from the theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to o}$ into the finite state transition system format, for each rule from the executable specification the corresponding transition rule should be created. Let us first consider the formulae from the theory $Th_{o\to m}$. To relate states of a transition system to the timeline used in these rules the unary predicate present_time is used. The atom present_time(t) being true in a given state indicates that t is the time in this state. Furthermore, the assumption from $Th_{o\to m}$ that an observation state and a corresponding memory state are created at the same time point should be preserved. Thus, the time increment rules are defined as:

present_time(0) $\wedge \neg$p $\twoheadrightarrow$ present_time(1)
present_time(t) $\wedge \neg q_{mem} \wedge \neg$p $\twoheadrightarrow$ present_time(t+1)
Now, when a relation between states and time points is established, the rules defined in the $Th_{o\to m}$ can be easily translated into the transition system format as it is shown in Table 2.

Table 2: Translation of the formulae from the executable theory $Th_{o\to m}$ into the corresponding finite state transition rules

Rule from the executable theory $Th_{o\to m}$	Corresponding transition rules
Memory state creation rule $\forall t'$ state(γ, t') $\models$ p $\Rightarrow$ state(γ, t') $\models$ memory(t', p)	present_time(t) $\wedge$ p $\twoheadrightarrow$ memory(t, p)
Memory persistence rule $\forall t''$ state(γ, t'') $\models$ memory(t', p) $\Rightarrow$ state(γ, t''+1) $\models$ memory(t', p)	memory(t, p) $\twoheadrightarrow$ memory(t, p)

Next, let us translate the properties from $Th_{m\to p}$. The time increment rules are created similarly to the $Th_{o\to m}$ case based on the assumption from $Th_{m\to p}$ that a preparation state is generated at the same time point, when the condition for an output is true.

present_time(t) $\wedge$ q_{cprep} $\wedge$ $\neg q_{cond}$(t) $\wedge$ $\neg$p $\twoheadrightarrow$ present_time(t+1)

present_time(t) $\wedge$ q_{prep} $\twoheadrightarrow$ present_time(t+1)

Then, the rules defined in the $Th_{m\to p}$ are translated into the transition system format in a straightforward manner as it is shown in Table 3.

Table 3: **Translation of the rules from the executable theory $Th_{m\to p}$ into the corresponding finite state transition rules**

Rule from the executable theory $Th_{m\to p}$	Corresponding transition rules
Memory state creation rule $\forall t'$ state(γ, t') $\models$ p $\Rightarrow$ state(γ, t') $\models$ [memory(t', p) $\wedge$ stimulus_reaction(p)]	present_time(t) $\wedge$ p $\twoheadrightarrow$ [memory(t, p) $\wedge$ stimulus_reaction(p)]
Memory persistence rule $\forall t''$ state(γ, t'') $\models$ memory(t', p) $\Rightarrow$ state(γ, t''+1) $\models$ memory(t', p)	memory(t, p) $\twoheadrightarrow$ memory(t, p)
Conditional preparation generation rule $\forall t'$ state(γ, t') $\models$ q_{mem} $\Rightarrow$ state(γ, t') $\models$ q_{cprep}	q_{mem} $\twoheadrightarrow$ q_{cprep}
Preparation state creation rule $\forall t'$, t state(γ, t') $\models$ [q_{cprep} $\wedge$ q_{cond}(t) $\wedge$ $\bigwedge$ stimulus_reaction(p)] p $\Rightarrow$ state(γ, t') $\models$ q_{prep}	present_time(t') $\wedge$ q_{cprep} $\wedge$ q_{cond}(t) $\wedge$ $\bigwedge$ stimulus_reaction(p) $\twoheadrightarrow$ q_{prep} p

Preparation state persistence rule $\forall$t' state(γ, t') |= [preparation_for(output(t'+c, a)) $\wedge$ $\neg$ output(a)] $\Rightarrow$ state(γ, t'+1) |= preparation_for(output(t'+c, a))	preparation_for(output(t+c, a)) $\wedge$ $\neg$output(a) $\twoheadrightarrow$ preparation_for(output(t+c, a))
Stimulus reaction state persistence rule $\forall$t' state(γ, t') |= [stimulus_reaction(p) $\wedge$ $\neg$preparation_for(output(t'+c, a))] $\Rightarrow$ state(γ, t'+1) |= stimulus_reaction(p)	present_time(t') $\wedge$ stimulus_reaction(p) $\wedge$ $\neg$preparation_for(output(t'+c, a)) $\twoheadrightarrow$ stimulus_reaction(p)

The executable theory from preparation to output $Th_{p\rightarrow o}$ contains only one formula that relates a preparation state at the time point t' to an output state at the time point t'+c; its translation is given in Table 4.

Table 4: Translation of the rule from the executable theory $Th_{p\rightarrow o}$ into the corresponding finite state transition rule

Rule from the executable theory $Th_{p\rightarrow o}$	Corresponding transition rules
Output generation rule $\forall$t' state(γ, t') |= preparation_for(output(t'+c, a)) $\Rightarrow$ state(γ, t'+c) |= output(a)	preparation_for(output(t+c, a)) $\wedge$ present_time(t+c-1) $\twoheadrightarrow$ output(a)

Example

The executable properties from the executable specification, translated into the transition rules for the considered example are given below:

food $\wedge$ not(cup) $\twoheadrightarrow$ observed(food)

not(food) $\wedge$ not(cup) $\twoheadrightarrow$ observed(not(food))

screen $\twoheadrightarrow$ observed(screen)

not(screen) $\twoheadrightarrow$ observed(not(screen))

present_time(t) $\wedge$ observed(food) $\twoheadrightarrow$ memory(t, observed(food))

present_time(t) $\wedge$ not(observed(not(food))) $\twoheadrightarrow$
memory(t, not(observed(not(food))))

present_time(t) $\wedge$ observed(not(food)) $\twoheadrightarrow$
memory(t, observed(not(food)))

present_time(t) $\wedge$ observed(not(screen)) $\twoheadrightarrow$

memory(t, observed(not(screen))) ∧ stimulus_reaction(observed(not(screen)))

memory(t, observed(food)) ⇸ memory(t, observed(food))

memory(t, not(observed(not(food)))) ⇸
memory(t, not(observed(not(food))))

memory(t, observed(not(food))) ⇸
memory(t, observed(not(food)))

memory(t, observed(not(screen))) ⇸
memory(t, observed(not(screen)))

present_time(t) ∧ ∃t2 [memory(t2, observed(food)) ∧ ∀t3, t ≥ t3 > t2 memory(t3, not(observed(not(food))))] ⇸
conditional_preparation_for(action(goto_food))

present_time(t) ∧ conditional_preparation_for(action(goto_food)) ∧ memory(t, observed(not(screen))) ∧ stimulus_reaction(observed(not(screen))) ⇸
preparation_for(action(t+c, goto_food))

present_time(t) ∧ stimulus_reaction(observed(not(screen))) ∧ not(preparation_for(action(t+c, goto_food))) ⇸
stimulus_reaction(observed(not(screen)))

preparation_for(action(t+c, goto_food)) ∧ not(performing_action(goto_food)) ⇸
preparation_for(action(t+c, goto_food))

preparation_for(action(t+c, goto_food)) ∧ present_time(t+c-1) ⇸
performing_action(goto_food).

The described transformation procedure was implemented in Java™, with an input (an external behavioral specification) and an output (an executable specification and finite transition system descriptions) files specified in textual format.

The generated finite state transition system representation is used in this paper for performing two types of analysis: by running simulations of different types of agent behavior, and by analyzing the consequences of the agent's behavior by model checking techniques.

4. Analysis of Agent Behavior by Simulation

In this Section the proposed transformation procedure is applied for simulating the delayed response and the adaptive behavior of an agent. For performing simulations a special software tool has been developed. Based on a specification of agent behavior in form of a transition system and using a sequence of external events (i.e., stimuli) as input, the program generates a trace (i.e., a sequence of agent states over time). The generated in such way traces can be used for analysis of external and internal dynamics of the agent in different experimental settings.

4.1 Simulation of the Delayed-response Behavior of the Agent

First, let us consider in more detail the example of the delayed-response behavior of the agent (a laboratory mouse), briefly introduced in Section 3. The behavior specification for this case is given below; it consists of environmental properties and externally observable behavioral properties of the agent.

Environmental properties:

EP1: At some time point food has been put at the position p1, after some time a cup has been placed upon food and after that the screen is raised

$\exists t1, t2, t3\ t2>t1\ \&\ t2<t3\ state(\gamma, t2) \models cup\_at(p1)\ \&\ state(\gamma, t1) \models food\_at(p1)\ \&\ state(\gamma, t3) \models not\_screen$

EP2: Food stays at the position where it has been put until it has been taken away or the agent is satisfied

$\forall t4\ state(\gamma, t4) \models [\ food\_at(X)\ \&\ not(mouse\_sat)\ \&\ not(food\_taken\_away\_from(X))\ ] \Rightarrow$
$state(\gamma, t4+1) \models food\_at(X)$, where $X \in \{p1, p2\}$

EP3: After the screen has been raised, it will never be drawn down again

$\forall t5\ state(\gamma, t5) \models not\_screen \Rightarrow state(\gamma, t5+1) \models not\_screen$

EP4: After placing the cup it will not be removed

$\forall t6\ state(\gamma, t6) \models cup\_at(X) \Rightarrow state(\gamma, t6+1) \models cup\_at(X)$, where $X \in \{p1, p2\}$

Properties that define the externally observable behavior of the mouse:

BP1: The mouse is able to observe presence (absence) of screen.

$\forall t7\ state(\gamma, t7) \models X \Rightarrow \exists t8\ t8>t7\ state(\gamma, t8, input(mouse)) \models observed(X)$,
where $X \in \{not\_screen, screen\}$

BP2: The mouse is always able to observe presence or absence of food if the cup is not covering it.

$\forall t9\ state(\gamma, t9) \models X\ \&\ not(cup\_at(Y)) \Rightarrow \exists t10\ t10>t9\ state(\gamma, t10, input(mouse)) \models observed(X)$,
where $X \in \{food\_at(Y), not(food\_at(Y))\}$ and $Y \in \{p1, p2\}$

BP3: The mouse is able to observe that food is taken away if the cup is not covering it.

$\forall t11\ state(\gamma, t11) \models food\_taken\_away\_from(X)\ \&\ not(cup\_at(X)) \Rightarrow$
$\exists t12\ t12>t11\ state(\gamma, t12, input(mouse)) \models observed(food\_taken\_away\_from(X))$,
where $X \in \{p1, p2\}$

BP4: The mouse always arrives at the position where it goes.

$\forall$t13 state(γ, t13, output(mouse)) |= performing_action(goto(X)) $\Rightarrow$
$\exists$t14 t14>t13 state(γ, t14) |= mouse_at(X),
where X$\in$ {p1, p2}

BP5: If the mouse is at the position with food, then it will be eventually satisfied (after consuming food).

$\forall$t15 state(γ, t15) |= mouse_at(X) & food_at(X) $\Rightarrow$ $\exists$t16 t16>t15 state(γ, t16) |= mouse_sat,
where X $\in$ {p1, p2}

BP6: The mouse consumes food completely.

$\forall$t17 state(γ, t17) |= mouse_sat & mouse_at(X) $\Rightarrow$ state(γ, t17+1) |= not(food_at(X))

BP7: If mouse found the position with food, it stays there.

$\forall$t18 state(γ, t18) |= mouse_at(X) & food_at(X) $\Rightarrow$ $\forall$t19 t19>t18 mouse_at(X)

BP8: Delayed-response behavior of the mouse

The mouse goes to the position with food if and only if it observes that there is no screen and at some point in the past the mouse observed food and since then did not observe the absence of food.

$\forall$t20 [state(γ, t20, input(mouse)) |= observed(not_screen) &
$\exists$t21< t20 state(γ, t21, input(mouse)) |= observed(food_at(X)) &
$\forall$t22, t20 $\geq$ t22 > t21 state(γ, t22, input(mouse))|= not(observed(not(food_at(X))))] $\Rightarrow$
$\exists$t23, t23>t20 state(γ, t23, output(mouse)) |= performing_action(goto(X)),
where X$\in$ {p1, p2}

The complete specification of the finite state transition system generated for this specification is given in (Sharpanskykh & Treur, 2005). This transition system was used to simulate a scenario of the animal's behavior with the following events in the environment: food is put at position p1 (time point 0), the screen separating the animal from food is present (time points 0-3), a cup is put at position p1, covering the food (time point 2), and the screen is removed (time point 4). The results of the simulation in the form of a trace (i.e., a sequence of states) are given in Table 5.

Table 5: Simulation trace illustrating delayed-response of the agent.

0: present_time(0) world(0,food_at(p1)) world(0,screen)	9: present_time(4) world(4,not(screen))
1: observed(0,food_at(p1)) observed(0,screen)	10: observed(4,not(screen))
2: memory(observed(0,food_at(p1))) memory(observed(0,screen))	11: memory(observed(4,not(screen))) stimulus_reaction(observed(not(screen)))
3: conditional_preparation_for(action(goto(p1)))	12: preparation_for(action(5,goto(p1)))
4: present_time(1)	13: not(stimulus_reaction(observed(not(screen))))
5: present_time(2) world(2,cup_at(p1))	14: present_time(5) performing_action(goto(p1))
6: observed(2,cup_at(p1))	15: not(preparation_for(action(5,goto(p1))))
7: memory(observed(2,cup_at(p1))	16: present_time(6) world(6,mouse_at(p1))
8: present_time(3)	17: present_time(7) world(7,mouse_sat)

Furthermore, a transition system representation can be used for construction of graphical models of agent dynamics. A graphical model for the considered example is shown in Figure 1. The state description literals with names started with a capital letter denote variables, which allow a concise representation of sets of states. For example, the label mem(T, obs(not(screen))) represents the set that includes every state corresponding to some time point t, in which the state property mem(t, obs(not(screen))) holds. An AND-relation between states requires all state properties of the states in the relation to be true in order to carry out the corresponding transition. A persistent state once activated, remains active at every time point in the future, i.e. the state properties of a persistent state hold for every time point in the future. The model in Figure 1 has been built manually. However, tools exist such as the one described in van Ham, van de Wetering, and van Wijk (2002), which allow for automatic visualization of finite state transition systems and can be used for graphical analysis of executable models. The graphical representation is particularly useful for the analysis of large transition systems. Usually such systems comprise a large number of transition rules specified without any particular order that do not provide a clear and ordered overview on the dynamics of a system. A graphical counterpart of a transition system makes temporal and causal relations between states of a system explicit and allows tracking different development paths of the system. Furthermore, the existing tools allow zooming into particular parts of a transition system to investigate relations between particular states.

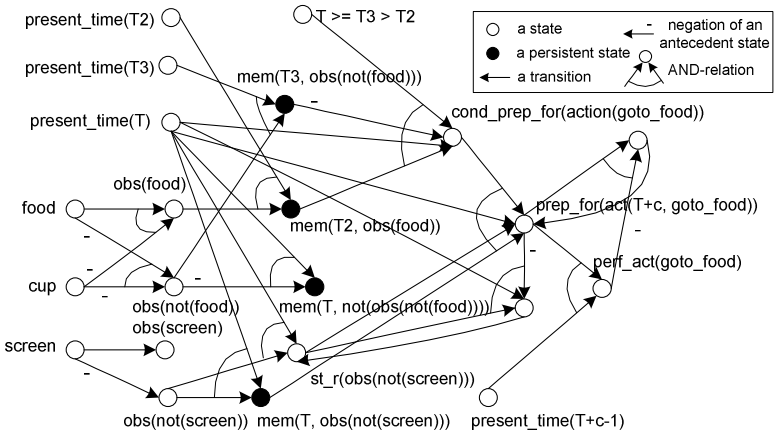

Figure 1: Graphical model, which describes delayed-response behavior in executable form.

4.2 Simulation of Adaptive Agent Behavior

In the second simulation example the adaptive behavior of *Aplysia Californica* (a sea hare) is considered. In neurobiology *Aplysia* has been often used for investigating classical and operant conditioning (Carew & Walters & Kandel, 1981). Consider a slightly simplified classical conditioning experiment of the *Aplysia*'s defensive withdrawal reflex. Before a learning phase a strong noxious stimulus (an electric shock) on the *Aplysia*'s tail produces a defensive reflex (a contraction), while a light tactile stimulus on *Aplysia*'s siphon does not lead to contraction. Formally:

$\forall t9 \le t$ state(γ, t9, input(aplysia)) |= observed(tail_shock) $\Rightarrow$
state(γ, t9+c, output(aplysia)) |= performing_action(contraction)

During the learning phase a light tactile stimulus on the Aplysia's siphon is repeatedly paired with an electric shock on its tail. After a few trials (for this example three temporal pairings are assumed) the animal reacts by contraction to the light tactile stimulus. The property that describes the learning process of the animal from the external perspective can be represented in the form $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)]$, where $\varphi_p(\gamma, t)$ is the formula:

$\exists t2, t3, t4, t5, t6, t7$ [t2 < t3 & t3 < t4 & t4 < t5 & t5 < t6 & t6 < t7 & t7 < t &
state(γ, t2, input(aplysia)) |= observed(touch_siphon) &
state(γ, t3, input(aplysia)) |= observed(tail_shock) &
state(γ, t4, input(aplysia)) |= observed(touch_siphon) &
state(γ, t5, input(aplysia)) |= observed(tail_shock) &
state(γ, t6, input(aplysia)) |= observed(touch_siphon) &
state(γ, t7, input(aplysia)) |= observed(tail_shock)]

and $\varphi_f(\gamma, t)$ is the formula

$\forall t8 \geq t$ [state(γ, t8, input(aplysia)) |= observed(touch_siphon) $\Rightarrow$
state(γ, t8+c, output(aplysia)) |= performing_action(contraction)]

with $\varphi_{cond}(\gamma, t, t8)$ is

$\forall t8 \geq t$ state(γ, t8, input(aplysia)) |= observed(touch_siphon)

and $\varphi_{act}(\gamma, t8)$ is

state(γ, t8+c, output(aplysia)) |= performing_action(contracts).

For this experiment c is assumed to be equal to two time units in a relative time scale.

Using the automated procedure, from the external behavioral specification of *Aplysia* a transition system was generated. This transition system was used to simulate a scenario of the animal's behavior with the following stimuli: touch the siphon (time points 0, 5, 9 and 15) and shock on the tail (time points 1, 6 and 10). The results of the simulation in form of a partial trace are given in Table 6.

Table 6: Partial simulation trace illustrating adaptive behavior of *Aplysia Californica*.

0: present_time(0) world(0,touch_siphon)	11: not(preparation_for(action(3,contracts)))
1: not(world(0,touch_siphon)) observed(0,touch_siphon)	.....
2: memory(observed(0,touch_siphon)) not(observed(0,touch_siphon)) stimulus_reaction(observed(touch_siphon))	39: present_time(15) world(15,touch_siphon)
3: present_time(1) world(1,tail_shock)	40: not(world(15,touch_siphon)) observed(15,touch_siphon)
4: not(world(1,tail_shock)) observed(1,tail_shock)	41: memory(observed(15,touch_siphon)) not(observed(15,touch_siphon)) stimulus_reaction(observed(touch_siphon))
5: memory(observed(1,tail_shock)) not(observed(1,tail_shock)) stimulus_reaction(observed(tail_shock))	42: preparation_for(action(17,contracts))
6: conditional_preparation_for(action(contracts))	43: not(stimulus_reaction(observed(touch_siphon))) not(stimulus_reaction(observed(tail_shock)))
7: preparation_for(action(3,contracts))	44: present_time(16)
8: not(stimulus_reaction(observed(touch_siphon))) not(stimulus_reaction(observed(tail_shock)))	45: present_time(17) performing_action(contracts)
9: present_time(2)	46: not(preparation_for(action(17,contracts)))
10: present_time(3) performing_action(contracts)	47: present_time(18)

In the given trace the process of conditioning starts at the state 0 (time point 0) and finishes at the state 36 (time point 12). After that the animal reacts to a light tactile stimulus (state 39) by producing a defensive reflex (states 42-45).

A graphical model for the example of classical conditioning for Aplysia Californica's defensive withdrawal reflex is shown in Figure 2.

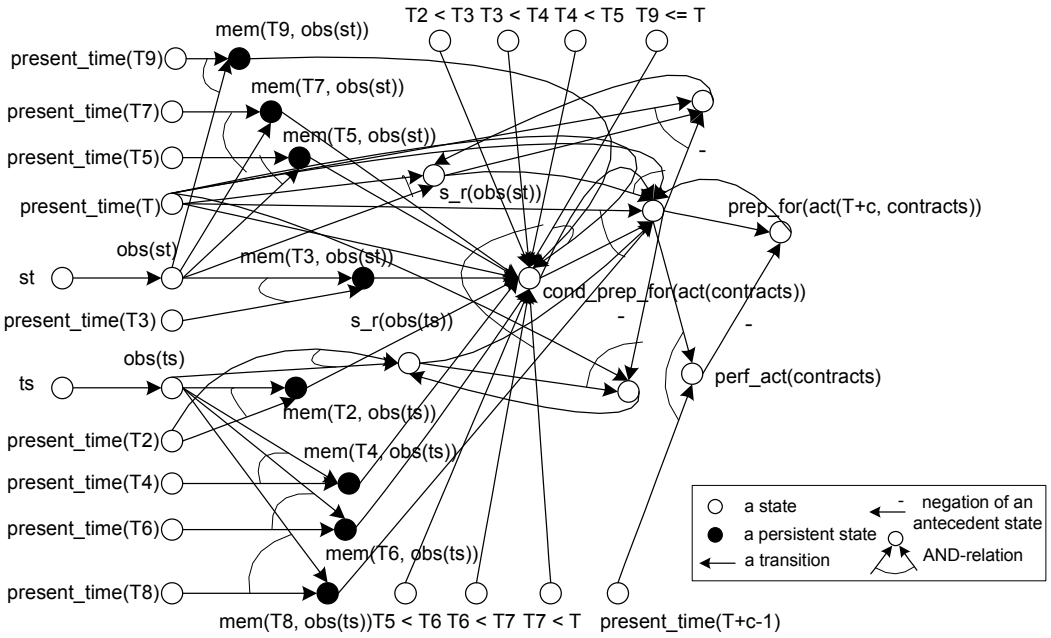

Figure 2: A graphical model for the example of classical conditioning for Aplysia Californica's defensive withdrawal reflex

5. Analysis of the Consequences of Agent Behavior by Model Checking

The proposed approach for analysis of the consequences of the agent behavior is based on the statement that the logical consequences of a certain external behavior specification are the logical consequences of the corresponding internal executable specification. This statement is supported by the following theorem.

Theorem[1]

If the internal dynamics specification $\pi(\gamma, t)$ corresponds (by the transformation above) to the external behavioral specification $\varphi(\gamma, t)$, and $\psi(\gamma, t)$ is a dynamic property of the agent in its environment, then $\psi(\gamma, t)$ is entailed by $\varphi(\gamma, t)$ if and only if $\psi(\gamma, t)$ is entailed by $\pi(\gamma, t)$:

$$\forall\gamma\ [\pi(\gamma, t) \Rightarrow \psi(\gamma, t)] \quad \Leftrightarrow \quad \forall\gamma\ [\varphi(\gamma, t) \Rightarrow \psi(\gamma, t)]$$

The consequences of the generated executable specification are easier to determine because of the simpler format of the internal dynamics specification. Furthermore, the process of analysis of such consequences can be automated by model checking techniques. For this purpose the SMV model checking tool is used in this paper. The SMV uses efficient algorithms to analyze a model of an agent system and the Computational Tree Logic (CTL) (McMillan, 1993) is used for properties (e.g., properties concerning well-being) to check. CTL is branching-time logic, meaning that its model of time is a tree-like structure in which different paths in the future are possible, any one of which might be actually realized. A particular use of CTL will be demonstrated by an example in this Section.

Moreover, the language for model specification in the SMV is similar to the executable format of agent behavioral specifications, which facilitates the automatic translation of the description of a finite state transition system, generated by the procedure introduced in Section 3, into the SMV input format. A specification in SMV is a plain text file that consists of two main parts: (1) a specification of a transition system and (2) a set of properties to be checked on the transition system specification expressed in CTL.

A transition system (or model) specification in SMV consists of a number of sections. In the section labeled VAR the names and types of the variables used in the model are defined. The type associated with a variable is either Boolean, scalar, or an array. In the second section labeled ASSIGN the initial values of variables are defined (i.e., the values that the variables have in the initial state) and the transition rules between states are specified. The transition rules are specified by case-expressions that define the change of values of the variables of the transition system as follows:

```
next (var) := case
                    boolean_expression: val;
              esac
```

[1] The proof for this theorem is given in Sharpanskykh & Treur (2005)

All case-expressions are evaluated in every state. When boolean_expression on the left-hand side of ":" of some transition rule is evaluated to true in some state, then the corresponding variable var will receive the value val in the next state.

For the translation of an executable specification of agent behavior into a SMV specification a dedicated procedure has been developed and implemented. This procedure is applied for every dynamic property in an executable behavioral specification as follows: First, the normalized memory state formula $q_{mem}(t)$ and the normalized condition state formula $q_{cond}(t, t_1)$ are processed by applying the steps 1-3 described below. After that conditional preparation generation rules are added by performing the step 4. Finally, the preparation and output state creation rules are generated by performing the step 5.

Step 1. For each occurrence of an existential quantifier of the form ∃t1 P(t1), where t1 is a time variable name and P(t1) is some function of the form memory(observed(t1, obs_event)), ¬memory(observed(t1, obs_event)), memory(output(t1, act_event)), or ¬memory(output(t1, act_event)), where obs_event and act_event are some atoms and for each occurrence of a universal quantifier of the form ∀t1 P(t1), create an atom (a label) t1 and add to the SMV specification the corresponding initialization rules.

Step 2. For each occurrence of the expression Q t1, t2 R t1 memory(observed(t1, obs_event)), where Q is either an existential or a universal quantifier, R is the comparison relation for the linear ordered time line: R∈{<, ≤}; t1 and t2 are time variables, add to the specification the following rule:

```
next(t1):= case
             t2 & obs_event: 1; //memory state creation
             !t2: 0;
             1: t1;   //persistence of memory
esac;
```

Similar rules should be added for the expressions Q t1, t2 R t1 memory(output(t1, act_event), Q t1, t2 R t1 ¬memory(observed(t1, obs_event)) and Q t1, t2 R t1 ¬memory(output(t1, act_event)).

Step 3. For each expression of the form ∃t1, t2 ∀t3 [t3 R t2 AND t1 R t3 AND memory(observed(t1, obs_event1)) AND memory(observed(t2, obs_event2)) & P3(t3)] if P3(t) is of the form memory(observed(t3, obs_event))

For t3 < t2 and t1< t3 add to the specification the following rules:

```
t3t1_eq: boolean ;
init(t3t1_eq):=0;
next(t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
          !obs_event2 & !t2 & t3t1_eq &
!obs_event3: 0;
            1: t1;
esac;
```

```
next(t3):= case
            !t1: 0;
            !obs_event2 & !t2 &
!obs_event3: 0;
            !obs_event2 & !t2 &
obs_event3: 1;
            1: t3;
esac;
```

The cases (ii) t3 < t2 and t1≤ t3; (iii) t3 ≤ t2 and t1< t3 and (iiii) t3 ≤ t2 and t1≤ t3 are dealt similarly.

Step 4. Add conditional preparation generation rules to the specification:

```
next(fmemN):= case   // N is a number of a dynamic property in the input specification
            ∧ti: 1; // conjunction of all labels, created based on φp(γ, t)
             i
            1: 0;
esac;
```

Step 5. For each action and communication a function output(act_event) in a formula $q_{bt}(t)$ add to the specification the following rules:

```
next(fprep_act):= case
            fmemN & ∧tj: 1;
                     j
                      1: 0;
esac;
```

```
next(act_event):= case
                  fprep_act: 1;
                          1: 0;
esac;
```

When an executable specification is translated into the SMV input format, the checking of a CTL property(ies) on this specification can be automatically performed using the SMV. As a result the tool generates an answer, if the specified property(ies) are satisfied by the model. If the property is not satisfied, a counterexample is provided. A counter-example shows a sequence of states that resulted in a state, in which the checked property is not satisfied. In such a way, the reason for the checking failure can be determined.

In this section the proposed analysis method is described and illustrated by an example, in which next to the delayed-response behavior (considered in Section 4.1) also the motivation-based behavior of the agent is analyzed. The specification for the delayed-response behavior (denoted here by φ_1) is given in Section 4.1. The specification for the motivation-based behavior (denoted here by φ_2) is constructed from the properties BP1-BP7, which are defined in Section 4.1, and additional properties BP9-BP12 given below.

BP9: Motivation-based behavior of the mouse (start at position p1)

If the mouse observes no screen and it is not satisfied, and at some time point in the past it observed food at position p1 and since then did not observe food at position p2, then the mouse will go to position p1.

$\forall t24$ [state(γ, t24, input(mouse)) |= observed(not_screen) & state(γ, t24) |= not(mouse_sat) &

$\exists t25, t25<t24$ state(γ, t25, input(mouse)) |= observed(food_at(p1)) &

$\forall t26, t26 \leq t24 \;\&\; t26 > t25$ state(γ, t26, input(mouse)) |= not(observed(food_at(p2)))] $\Rightarrow$

$\exists t27, t27>t24$ state(γ, t27, output(mouse)) |= performing_action(goto(p1))

BP10: Motivation-based behavior of the mouse (start at position p2)

If the mouse observes no screen and it is not satisfied, and at some time point in the past it observed food at position p2 and since then did not observe food at position p1, then the mouse will go to position p2.

$\forall t24$ [state(γ, t24, input(mouse)) |= observed(not_screen) &

state(γ, t24) |= not(mouse_sat) & $\exists t25, t25<t24$ state(γ, t25, input(mouse)) |= observed(food_at(p2)) &

$\forall t26, t26 \leq t24 \;\&\; t26 > t25$ state(γ, t26, input(mouse)) |= not(observed(food_at(p1)))] $\Rightarrow$

$\exists t27, t27>t24$ state(γ, t27, output(mouse)) |= performing_action(goto(p2))

BP11: Motivation-based behavior of the mouse (continue at position p2)

If the mouse is at position p1 and there is no food at p1 and the mouse is still not satisfied, then it will go to position p2 to continue its search for food

$\forall t28$ state(γ, t28) |= mouse_at(p1) & not(food_at(p1)) & not(mouse_sat) $\Rightarrow$

$\exists t29, t29>t28$ state(γ, t29, output(mouse)) |= performing_action(goto(p2))

BP12: Motivation-based behavior of the mouse (continue at position p1)

If the mouse is at position p2 and there is no food at p2 and the mouse is still not satisfied, then it will go to position p1 to continue its search for food

$\forall t30$ state(γ, t30) |= mouse_at(p2) & not(food_at(p2)) & not(mouse_sat) $\Rightarrow$

$\exists t31, t31>t30$ state(γ, t31, output(mouse)) |= performing_action(goto(p1))

Such a specification of behavior can be attributed, for example, to an animal that feels hunger.

Both types of behavior of the agent are analyzed in two different environmental experimental settings (E, resp. E') with an identical initial situation, described as follows:

The mouse is placed in front of a transparent screen that separates it from a piece of food that is put behind the screen. The mouse is able to observe the position of food and of the screen. At some moment after food has been put, a cup is placed covering the food, which makes food invisible for the mouse. After some time the screen is raised and the animal is free to go to any position. If the mouse comes to the position, where the food is hidden, then it will be capable to lift up the cup and get the food.

By means of the analysis method described below it is determined for each of the environmental settings and each type of behavior whether the combination will bring the agent well-being.

The Analysis Method

(a) By means of the translation procedure described in Section 3, each external behavioral specification φ_i is automatically translated into the corresponding executable internal dynamics specification π_i and related to it state transition system representation τ_i.

(b) The well-being properties ψ and ψ' to be checked are specified in CTL

(c) Using the state transition system representations τ_i, verification of each of the agent models with respect to properties ψ and ψ' is performed in the SMV model checker, resulting in confirmed or rejected entailment relations between the π_i and ψ and ψ'.

(d) Based on the theorem introduced at the beginning of this Section the confirmed or rejected entailment relations between the π_i and ψ and ψ' imply corresponding confirmed or rejected entailment relations between the φ_i and ψ and ψ'.

The environmental conditions E are defined by dynamic properties EP1-EP4 listed in Section 4.1. For this example, ψ is the following conditional well-being property (which is expressed conditionally for environmental conditions E):

for all traces, if the screen is removed and food is hidden under the cup, then the mouse will eventually be satisfied.

This property ψ can be expressed in Computation Tree Logic (CTL) (Clarke & Grumberg & Peled, 1999) required for verification in the SMV model checking tool as follows:

$$\boldsymbol{AG}\,(\text{not\_screen} \,\&\, \text{food} \,\&\, \text{cup} \rightarrow \boldsymbol{AF}\,\text{mouse\_sat})$$

where ***A*** is a path quantifier defined in CTL, meaning “for all computational paths”, ***G*** and ***F*** are temporal quantifiers that correspond to “globally” and “eventually” respectively.

The automatic verification in the SMV model checking tool showed that the property ψ expressing well-being under environmental conditions E is entailed by the model of the agent delayed-response behavior expressed by φ_1. The model of agent motivation-based behavior φ_2 also turns out to entail the general property ψ.

In the second experimental setting, described by environmental conditions E', the mouse observed food for some time at the position p1, after that one cup is put covering the food and another cup is put at the position p2, which is also behind the transparent screen. Thereafter, invisibly for the mouse, food is removed from position p1 and put under the cup at position p2. Later the screen is raised and the animal is free to go to any position. The environmental conditions E' are formalized by dynamic properties BP2-BP4, and by the property BP5:

EP5: At some time point food had been put at the position p1, after some time one cup had been placed upon food and another cup had been placed at the position p2; thereafter food has been taken away from p1 and has been put at p2 behind the cup, after that the screen is raised

$\exists t32, t33, t34, t35, t33>t32 \ \& \ t33<t34 \ \& \ t35>t34 \ state(\gamma, t33) \models [cup\_at(p1) \ \& \ cup\_at(p2)] \ \& \ state(\gamma, t32) \models food\_at(p1) \ \& \ state(\gamma, t34) \models [food\_taken\_away\_from(p1) \ \& \ food\_at(p2)] \ \& \ state(\gamma, t35) \models not\_screen$

The global property ψ' to be verified in this case expresses well-being under these environmental conditions E':

for all traces if the screen is removed and food is hidden behind the cup at position p2, then the mouse will eventually be satisfied,

or, in CTL:

$$\boldsymbol{AG}\ (not\_screen \ \& \ food\_at(p2) \ \& \ cup\_at(p2) \rightarrow \boldsymbol{AF}\ mouse\_sat)$$

The automated verification in SMV showed that the model of the agent behavior φ_1 for the delayed-response case does not entail property ψ' expressing well-being under environmental conditions E'. From the counter-example generated by the model checker it is visible that the animal went to the position p1, and did not find food there, and after that did not go anywhere else, which caused the failure of the property.

Unlike the external behavior specification φ_1 that describes the delayed-response behavior of the agent, the specification φ_2 for the motivation-based behavior includes behavioral repertoire to deal with invisible food, expressed in the form of properties that turn out to ensure the entailment of global property ψ'. More specifically, φ_2 expresses the behavior that if the agent could not find food

at the position where it has seen it before, and the agent is still not satisfied, then the agent will search for food at another position p2. Formally this is expressed by:

$\forall$t5 state(γ, t5) |= mouse_at(p1) & not(food_at(p1)) & not(mouse_sat) $\Rightarrow$
    $\exists$t6, t6>t5 state(γ, t6, output(mouse)) |= performing_action(goto(p2))

$\forall$t7 state(γ, t7) |= mouse_at(p2) & not(food_at(p2)) & not(mouse_sat) $\Rightarrow$
    $\exists$t8, t8>t7 state(γ, t8, output(mouse)) |= performing_action(goto(p1))

The automated verification in SMV confirmed that the external behavioral specification φ_2 for the case of motivation-based behavior entails property ψ'.

From the results of verification of the external behavioral specifications φ_1 and φ_2 for both types of behavior in both experimental settings with respect to the entailment of properties ψ and ψ' (see Table 7) we draw the conclusion that the agent that manifests motivation-based behavior φ_2 fits more for surviving in the world, described by the two types of experimental conditions than the agent that has the delayed-response behavior φ_1.

Table 7: Outcomes of the Example Analysis

		well-being under different environmental conditions	
		ψ	ψ'
behavior type	delayed response φ_1	+	-
	motivation-based φ_2	+	+

6. Discussion

Behavior of organisms comes in a variety of forms and complexities. Simple forms of behavior such as stimulus-response patterns can be formalized in relatively simple terms, based on direct stimulus-action associations that can be considered as associations between an input state and a subsequent output state of the organism. A description of an organism's behavior in terms of such stimulus-action associations can directly be used as a basis to model and analyze this behavior. For more complex behavior, however, the picture is not so simple. To describe behavior from the external perspective, in general, an input-output correlation (cf. Kim, 1996) has to be specified which indicates how a pattern of input states over time relates to a pattern of output states over time. With increasing complexity of the behavior considered, specification of such an input-output correlation will become more complex, and not take the form of direct stimulus-action associations anymore. The question arises on how such more complex descriptions of behavior can be expressed

and handled, and, in particular, how such behavior can be analyzed, for example, by simulation and verification. The problem with specifications of input-output correlations for externally observable behavior is that for any type of behavior that is a bit more complex than stimulus-response behavior, such specifications have a temporal complexity that makes them useless as a basis for analysis by simulation or verification.

The solution for this problem developed in this paper is twofold. First, a formal language is put forward that allows specifying behavior from an external perspective in terms of dynamic properties involving input states and output states over time. Secondly, it is shown how an external behavior specification expressed in such a language can be automatically transformed into an equivalent executable specification that easily can be used to perform different types of analysis of agent behavior. This transformation creates a specification based on postulated internal states (in particular memory states and preparation states), and their direct temporal relationships. Here memory states do not simply represent of certain aspects of the world state, but they can provide representations of temporal relationships over various states in the past.

Further, the paper illustrates how based on generated executable specifications both simulations and the analysis of consequences of agent behavior can be performed. Alternative methods for temporal analysis of reactive systems are discussed in (Manna & Pnueli, 1995); also these methods can be applied, once an executable behavioral specification has been generated.

The analysis of consequences of agent behavior is performed by means of model checking techniques using the SMV model checker. The external behavioral specification is related to the executable SMV specification in the following linear way:

(1) for every quantified variable from a non-executable specification a variable and an appropriate rule for its update are introduced;

(2) for every nested quantifier an additional variable and an auxiliary executable rule are introduced, which establishes a relation between the quantified variables;

(3) for every observed atom from a past and a conditional formulae from dynamic properties, a corresponding memory state creation and a memory state persistence rule are introduced using the variables described in (1) and (2), and variables that correspond to external events;

(4) for every non-executable dynamic property auxiliary variables fmem and fprep (i.e., the variables that indicate truth values of $\varphi_{mem}(\gamma, t)$ and $\varphi_{prep}(\gamma, t_1)$ respectively) and corresponding update rules are introduced;

(5) for every action specified in $\varphi_{act}(\gamma, t_1)$ a variable and an appropriate update rule are introduced;

(6) for reactivation of agent preparation states the auxiliary variables and the update rules corresponding to observed atoms from $\varphi_{prep}(\gamma, t_1)$ are introduced.

Notice that an SMV-specification comprises constants, variables and state transition rules with limited expressiveness (e.g., no quantifiers). Furthermore, for expressing one complex temporal relation a large quantity (including auxiliary) of transition rules is needed. Specification of agent system behavior observed externally in the more expressive predicate-logic-based language TTL is much easier. TTL proposes an intuitive way of creating a specification of system dynamics, which still can be automatically translated into a state transition system description, as shown in this paper.

In general, the transformation into executable format can be achieved in different ways, depending on the format of an external behavioral specification of an agent system. For example, for translating agent behavioral specifications expressed in modal temporal logics into executable format, procedures described in (Fisher, 1996) can be used. This paper exploits a procedure to generate an executable internal behavioral specification from a more expressive external specification than is possible in modal temporal logics. The executable format introduced in this paper has similarities with the production rule representation formats used in existing cognitive architectures. For example, in the ACT-R architecture (Anderson, 1996) rules are stored in the procedural memory, which is essentially specified by a production system and can be easily expressed by formulae from the executable specification introduced in this paper.

Also, different languages may be used for describing internal executable specifications based on different types of internal states. For the approach chosen here an internal specification is expressed over memory states of an agent that are based on the agent sensing (i.e., observations) of not only his/her environment state, but also of patterns over time of this environment and his/her own behavior therein (e.g., actions). This relates to a thesis currently recognized in neurobiological research (Di Ferdinando & Parisi, 2004) that internal representations of an agent are based not only on the properties of the sensory input, but also on the properties of the actions with which the agent responds to this sensory input. The internal states can represent world states, but may also refer to more complex temporal patterns occurring in the past. This focus on internal representations for temporal patterns is also supported by other literature (e.g., (Damasio, 1999; Dennett, 1991, 2001, 2005; Pockett, Banks and Gallagher, 2006; Wegner, 2002, 2003)). This literature discusses that for crucial cognitive capabilities, certain mental states or brain states are exploited that can be interpreted as representing temporal information. Several authors put forward ideas on consciousness that incorporate this thesis. For example, Damasio (1999)'s notion of core consciousness is based on internal second-order representations of the process of body change upon

a stimulus. Furthermore, Wegner (2002, 2003)'s notion of conscious will is based on an internal representation of the temporal relationship between the occurrences of a thought and an action. Moreover, Dennett (2001, 2005) discusses how from an asynchronous distributed process that occurs in the brain consciousness emerges by temporal representation of the ordering of events.

Descriptions of memory states can also be compared to information chunks stored in memory considered in (Kokinov 2003). When a problem has to be solved, an organism retrieves relevant chunks from his/her memory and combines them into an episode representing a similar problem/situation occurred in the past. Using the terminology of the approach proposed in this paper, an episode corresponds to a (possibly complex) temporal property based on memory states and relations between them.

Sometimes, when a structure of a neurological circuit of an organism is known, it is possible to relate postulated internal states to certain real neurological states of an organism. The neurological model of *Aplysia Californica*, suggested by Roberts and Glanzman (2003) allows finding some correspondences between the postulated internal states described in the example of this paper and the real physical states of the organism. The observation states from our model can be related to activation states of sensory neurons, whereas the memory states (to some extent) can be put into correspondence with an enhancement of the strength of the synaptic connection between the sensory and motor neurons and with an associative increase in the excitability of the siphon sensory neurons of *Aplysia*.

However, the rules for the creation of internal states of an agent proposed in this approach are based on the idealized assumptions described above, which may lead to internal states that do not correspond in a direct manner to internal states actually occurring in certain biological organisms. If such a direct correspondence is aimed for, to ensure the biological plausibility of the models constructed using the proposed approach for specific forms of organisms (types of agents), the rules for creation of intrinsic states may be adjusted correspondingly.

Also other existing frameworks and approaches that include different types of mental states of an agent (e.g., BDI (Rao & Georgeff, 1991), KARO (van Linder & van der Hoek & Meyer, 1998), Schweiger Gallo & Gollwitzer (2007)) can be considered for internal representation. In particular, these frameworks recognize attitudes of agents such as desires, intentions, and goals. More specifically, in (Gollwitzer, 1999) it is shown that intentions can be implemented by if-then plans that describe when, where and how a goal set of an agent has to be put into action: "if situation x is encountered, then the agent will perform behavior y". However, such if-then plans can be specified using the approach proposed in this paper by temporal relations between externally observable and

internal states of an agent. In this case no introduction of supplementary internal concepts is required, and the intentional aspects of the agent behavior are implicitly realized through the temporal rules in the behavior specification of the agent. However, goal and intention concepts could also be considered explicitly by adding them to our ontology in order to get more transparency. However, this will add no essential expressivity, as they would be a renaming of already available complex expressions over our memory states; see also (Jonker, Treur, and Vries, 2002). In future work it will be investigated, which alternative or additional attitudes of agents could be included into the internal framework in order to more transparently represent certain specific types of complex behavior of an agent.

References

Allen, C., and Bekoff, M., (1997). *Species of Mind: the philosophy and biology of cognitive ethology*. MIT Press.

Anderson, J. R. (1996). ACT: A simple theory of complex cognition. *American Psychologist*, 51, 355-365.

Arnold, A. (1994). *Finite transition systems*. Semantics of communicating systems. Prentice-Hall.

Balkenius, C., & Moren, J. (1999). Dynamics of a classical conditioning model. *Autonomous Robots, 7*, 41-56.

Carew, T.J. & Walters, E.T., & Kandel, E.R. (1981). Classical conditioning in a simple withdrawal reflex in *Aplysia Californica*. *The Journal of Neuroscience, 1(12)*, 1426-1437.

Clarke, E.M., & Grumberg, E.M., & Peled, D.A. (1999). *Model Checking*, MIT Press, Cambridge Massachusetts, London England.

Damasio, A. (1999). *The Feeling of What Happens: Body, Emotion and the Making of Consciousness*. Harcourt Brace.

Dennett, D.C. (1991). *Consciousness Explained*. Little, Brown & Company. Published by Penguin Books.

Dennett, D.C. (2001). Are we Explaining Consciousness Yet? *Cognition, 79*, 221-237.

Dennett, D.C. (2005). *Sweet Dreams: Philosophical Obstacles to a Science of Consciousness*. MIT Press.

Di Ferdinando, A. & Parisi, D. (2004) Internal representations of sensory input reflect the motor output with which organisms respond to the input. In Carsetti A. (ed.): *Seeing, Thinking and Knowing*. Kluwer, Dordrecht, 115-141

Dudai Y. (1990). *The Neurobiology of Memory. Concepts, Findings, Trends.* Oxford: Oxford University Press.

Fisher, M. (1996). An Introduction to Executable Temporal Logics, *Knowledge Engineering Review 11(1)*, 3-36.

Fitting, M. (1996). *First-order Logic and Automated Theorem Proving*, 2nd edition, Springer-Verlag.

Hawkins, J. (2004). *On Intelligence*, Henry Gholt and Co Ltd.

Heil, J. (2000). *Philosophy of Mind*. Routledge.

Hunter, W.S. (1912). The delayed reaction in animals. *Behavioral Monographs, 2*, 1-85.

Jonker, C.M., & Treur J., & Wijngaards W.C.A. (2003). A temporal-modelling environment for internally grounded beliefs, desires, and intentions. *Cognitive Systems Research Journal, 4(3),* 191-210.

Jonker, C.M., & Treur, J. (2002) Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. *International Journal of Cooperative Information Systems*, *11*, 51-92.

Jonker, C.M., Treur, J., and Vries, W. de, (2002). Temporal Analysis of the Dynamics of Beliefs, Desires, and Intentions. *Cognitive Science Quarterly* (Special Issue on Desires, Goals, Intentions, and Values: Computational Architectures), vol. 2, 2002, pp.471-494.

Kim, J. (1996). *Philosophy of Mind*. Westview Press

Kokinov, B. (2003). The Mechanisms of Episode Construction and Blending in DUAL and AMBR: Interaction Between Memory and Analogy. In: Kokinov, B., Hirst, W. (ed.) *Constructive Memory*. Sofia: NBU Press.

Kowalski, R., & Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing, 4*, 67-95.

Manna, Z., & Pnueli A. (1995) *Temporal verification of reactive systems*, Springer-Verlag, Berlin Heidelberg New York.

Manzano, M. (1996). *Extensions of First Order Logic*, Cambridge University Press.

McMillan, K. (1993). *Symbolic Model Checking*, Kluwer Academic Publishers.

Pockett, S., Banks, W.P., and Gallagher, S. (eds.) (2006). *Does Consciousness Cause Behaviour?* MIT Press.

Priest, S. (1991). *Theories of the Mind*. Penguin.

Putman, H. (1975). *Mind, Language, and Reality: Philosophical papers*, vol.2. Cambridge: Cambridge University Press.

Rao, A. S. & Georgeff, M. P. (1991). Modeling agents within a BDI architecture. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR '91)*. Morgan Kaufmann, Cambridge, MA, 473-484.

Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge MA: MIT Press.

Roberts, A.C., & Glanzman, D.L. (2003). Learning in Aplysia: looking at synaptic plasticity from both sides. *Trends in Neurosciences*, *26*, 662-670.

Schweiger Gallo, I., & Gollwitzer, P. M. (2007). Implementation intentions: A look back at fifteen years of progress. *Psicothema*, *19*, 37-42.

Skinner, B.F. (1935). The generic nature of the concepts of stimulus and response. *Journal of General Psychology, 12*, 40-65.

Skinner, B.F. (1953). *Science and human behavior*. New York: Macmillan.

Sharpanskykh, A. & Treur, J. (2005). *Modeling of Agent Behavior Using Behavioral Specifications* (Tech. Rep. 06-02ASRAI; http://hdl.handle.net/1871/9123). Vrije Universiteit, Amsterdam.

van Ham, F., & van de Wetering, H., & van Wijk, J.J. (2002). Interactive Visualization of State Transition Systems. *IEEE Transactions on Visualization and Computer Graphics, 8(4)*, IEEE CS Press, 319-329.

van Linder, B.W., & van der Hoek, & Meyer, J.-J. Ch. (1998). Formalising Abilities and Opportunities of Agents, *Fundamenta Informaticae, 34(1-2)*, 53-101.

Vardi, M.Y. (1996). An automata-theoretic approach to linear temporal logic. In: *Proceedings of the VIII Banff Higher Order Workshop*, in: Lecture Notes in Computer Science, vol. 1043, Springer-Verlag, 238–266.

Watson, J. B. (1913). Psychology as the Behaviorist Views It. *Psychological review, 20*, 158-177.

Wegner, D.M. (2002). *The Illusion of Conscious Will*. MIT Press.

Wegner, D.M. (2003). The mind's best trick: how we experience conscious will. *Trends in Cognitive Science*, *7*, 65-69.

Table 1: Executable Format

If a conjunction of state properties X holds in trace γ at time point t, *(Step Property)* $\forall t\ state(\gamma,t) \models X$	**then** a(nother) conjunction of state properties Y will hold in trace γ at time point t+c, with c>0 an integer constant $\Rightarrow\ state(\gamma, t+c) \models Y$
If a conjunction of state properties X and a condition property C hold in trace γ at time point t, *(Conditional Persistency Property)* $\forall t\ state(\gamma\Box, t) \models X \wedge C$	**then** this conjunction of state properties X will hold in trace γ at the next time point $\Rightarrow\ state(\gamma, t+1) \models X$
If a conjunction of state properties X holds in trace γ $\Box$at time point t, *(State Relation Property)* $\forall t\ state(\gamma\Box, t) \models X$	**then** a(nother) conjunction of state properties Y will hold in trace γ at the same time point t $\Rightarrow\ state(\gamma\Box, t) \models Y$

Table 2: Translation of the formulae from the executable theory $Th_{o \to m}$ into the corresponding finite state transition rules

Rule from the executable theory $Th_{o \to m}$	Corresponding transition rules
Memory state creation rule $\forall t'\ state(\gamma, t') \models p \Rightarrow state(\gamma, t') \models memory(t', p)$	$present\_time(t) \wedge p \twoheadrightarrow memory(t, p)$
Memory persistence rule $\forall t''\ state(\gamma, t'') \models memory(t', p) \Rightarrow state(\gamma, t''+1) \models memory(t', p)$	$memory(t, p) \twoheadrightarrow memory(t, p)$

Table 3: **Translation of the rules from the executable theory $Th_{m\rightarrow p}$ into the corresponding finite state transition rules**

Rule from the executable theory $Th_{m\rightarrow p}$	Corresponding transition rules
Memory state creation rule $\forall t'$ state(γ, t') |= p $\Rightarrow$ state(γ, t') |= [memory(t', p) $\wedge$ stimulus_reaction(p)]	present_time(t) $\wedge$ p $\twoheadrightarrow$ [memory(t, p) $\wedge$ stimulus_reaction(p)]
Memory persistence rule $\forall t''$ state(γ, t'') |= memory(t', p) $\Rightarrow$ state(γ, t''+1) |= memory(t', p)	memory(t, p) $\twoheadrightarrow$ memory(t, p)
Conditional preparation generation rule $\forall t'$ state(γ, t') |= q_{mem} $\Rightarrow$ state(γ, t') |= q_{cprep}	q_{mem} $\twoheadrightarrow$ q_{cprep}
Preparation state creation rule $\forall t'$, t state(γ, t') |= [q_{cprep} $\wedge$ q_{cond}(t) $\wedge$ $\bigwedge$ stimulus_reaction(p)] p $\Rightarrow$ state(γ, t') |= q_{prep}	present_time(t') $\wedge$ q_{cprep} $\wedge$ q_{cond}(t) $\wedge$ $\bigwedge$ stimulus_reaction(p) $\twoheadrightarrow$ q_{prep} p
Preparation state persistence rule $\forall t'$ state(γ, t') |= [preparation_for(output(t'+c, a)) $\wedge$ $\neg$ output(a)] $\Rightarrow$ state(γ, t'+1) |= preparation_for(output(t'+c, a))	preparation_for(output(t+c, a)) $\wedge$ $\neg$output(a) $\twoheadrightarrow$ preparation_for(output(t+c, a))
Stimulus reaction state persistence rule $\forall t'$ state(γ, t') |= [stimulus_reaction(p) $\wedge$ $\neg$preparation_for(output(t'+c, a))] $\Rightarrow$ state(γ, t'+1) |= stimulus_reaction(p)	present_time(t') $\wedge$ stimulus_reaction(p) $\wedge$ $\neg$preparation_for(output(t'+c, a)) $\twoheadrightarrow$ stimulus_reaction(p)

Table 4: Translation of the rule from the executable theory $Th_{p\rightarrow o}$ into the corresponding finite state transition rule

Rule from the executable theory $Th_{p\rightarrow o}$	Corresponding transition rules
Output generation rule $\forall t'$ state(γ, t') |= preparation_for(output(t'+c, a)) $\Rightarrow$ state(γ, t'+c) |= output(a)	preparation_for(output(t+c, a)) $\wedge$ present_time(t+c-1) $\rightarrow\!\!\!\rightarrow$ output(a)

Table 5: Simulation trace illustrating delayed-response of the agent.

0: present_time(0) world(0,food_at(p1)) world(0,screen)	9: present_time(4) world(4,not(screen))
1: observed(0,food_at(p1)) observed(0,screen)	10: observed(4,not(screen))
2: memory(observed(0,food_at(p1))) memory(observed(0,screen))	11: memory(observed(4,not(screen))) stimulus_reaction(observed(not(screen)))
3: conditional_preparation_for(action(goto(p1)))	12: preparation_for(action(5,goto(p1)))
4: present_time(1)	13: not(stimulus_reaction(observed(not(screen))))
5: present_time(2) world(2,cup_at(p1))	14: present_time(5) performing_action(goto(p1))
6: observed(2,cup_at(p1))	15: not(preparation_for(action(5,goto(p1))))
7: memory(observed(2,cup_at(p1))	16: present_time(6) world(6,mouse_at(p1))
8: present_time(3)	17: present_time(7) world(7,mouse_sat)

Table 6: Partial simulation trace illustrating adaptive behavior of *Aplysia Californica*.

0: present_time(0) world(0,touch_siphon)	11: not(preparation_for(action(3,contracts)))
1: not(world(0,touch_siphon)) observed(0,touch_siphon)	……
2: memory(observed(0,touch_siphon)) not(observed(0,touch_siphon)) stimulus_reaction(observed(touch_siphon))	39: present_time(15) world(15,touch_siphon)
3: present_time(1) world(1,tail_shock)	40: not(world(15,touch_siphon)) observed(15,touch_siphon)
4: not(world(1,tail_shock)) observed(1,tail_shock)	41: memory(observed(15,touch_siphon)) not(observed(15,touch_siphon)) stimulus_reaction(observed(touch_siphon))
5: memory(observed(1,tail_shock)) not(observed(1,tail_shock)) stimulus_reaction(observed(tail_shock))	42: preparation_for(action(17,contracts))
6: conditional_preparation_for(action(contracts))	43: not(stimulus_reaction(observed(touch_siphon))) not(stimulus_reaction(observed(tail_shock)))
7: preparation_for(action(3,contracts))	44: present_time(16)
8: not(stimulus_reaction(observed(touch_siphon))) not(stimulus_reaction(observed(tail_shock)))	45: present_time(17) performing_action(contracts)
9: present_time(2)	46: not(preparation_for(action(17,contracts)))
10: present_time(3) performing_action(contracts)	47: present_time(18)

Table 7: Outcomes of the Example Analysis

		well-being under different environmental conditions	
		ψ	ψ'
behavior type	delayed response φ_1	+	-
	motivation-based φ_2	+	+

Figure 1: Graphical model, which describes delayed-response behavior in executable form.

Figure 2: A graphical model for the example of classical conditioning for Aplysia Californica's defensive withdrawal reflex