In this paper the formal theoretical basis used for transformation of a non-executable external behavioral specification for an agent system into an executable format, required for enabling verification techniques, is explained in detail.

An external behavioral specification for the agent is specified using the Temporal Trace Language (TTL), which syntax and semantics are explained in Section 1.

In the general case an external behavioral agent specification consists of complex temporal relations expressed as dynamic properties in non-causal (and also non-executable) format, which does not allow direct application of automatic verification or simulation. In particular, in order to enable verification and validation of more general dynamic properties against the dynamic properties that constitute the agent behavioral model, an external behavioral description should be replaced by one in a simpler format, closer to a finite state transition system format required, for example, by model checking techniques. In order to obtain this, an agent behavioral description is replaced by one in executable (temporal) format. The justification of such substitution is based on the theorem that a behavioral specification entails a certain dynamic property if and only if the generated executable specification entails the same property. The proof for this theorem and other formal theoretical results are given in Section 2.

Moreover, for the purposes of practical verification by means of model checking techniques, an automated translation from a behavioral specification based on executable temporal logical properties into a finite state transition system description has been developed. The details of a translation procedure are explained in Section 3.

Furthermore, the procedure for translating from the state transition system description into the model specification format for the SMV model checker that is used for verification, as well as the complexity issues of the translated specification are considered in Section 4.

In Sections 5 the application of the proposed approach is illustrated by a paradigmatic example.

1. TTL Syntax and Semantics

The language TTL, short for Temporal Trace Language, is a variant of order-sorted predicate logic. Whereas standard multi-sorted predicate logic is a language to reason about static properties only, TTL is an extension of such languages with facilities for reasoning about the dynamic properties of arbitrary systems expressed by static languages.

1.1 The Language of TTL

For expressing TTL-formulae ontologies are used. In logical terms, an ontology is a signature that specifies the vocabulary of a language to represent and reason about a system.

**Definition 1.1 (TTL Signature)**

A signature consists of the symbols of the following classes:

1. For representing and reasoning about objects of different ontological kinds a number of syntactical sorts are introduced. Among them are several standard sorts: TIME (a set of all time points), STATE (a set of all state names), TRACE (a set of all trace names; a trace can be considered as a timeline), STATPROP (a set of all state property expressions) and STATOM (a set of all state atoms; STATOM is a subset of STATPROP). All other sorts, which depend on an application domain, represent subsorts of the general sort OBJECT. Furthermore, for every sort S a sort S_VARS exists, which contains constant names of all possible variables of sort S. A union of all sorts S_VARS for all sorts S constitutes a sort STATE_VARIABLE.

2. Countably infinite number of individual variables of each sort. We shall use t with subscripts and superscripts for variables of the sort TIME; γ with subscripts and superscripts for variables of the sort TRACE; s with subscripts and superscripts for variables of the sort STATE.
(3) a set of constants \( C \) of each sort, among which \( \text{true} \) and \( \text{false} \) of the sort \( \text{TRUTH\_VALUE} \).

(4) a set of function symbols \( \Phi \), among which:

a) for each \( n \geq 1 \), a finite or countably infinite number of function symbols of type \( \text{OBJECT}^n \rightarrow \text{STATPROP} \)
b) a binary function symbol \( \text{comp\_aspect} \) of type \( \text{ASPECT\_COMPONENT} \times \text{COMPONENT} \rightarrow \text{COMPONENT\_STATE\_ASPECT} \), where a sort \( \text{ASPECT\_COMPONENT} \) is a set of the agent aspects (i.e., inputs, outputs, and internal); a sort \( \text{COMPONENT} \) is a set of all agent names; and a sort \( \text{COMPONENT\_STATE\_ASPECT} \) is a set of all names of aspects of all agent states.
c) a function symbol \( \text{state} \) of type \( \text{TRACE} \times \text{TIME} \times \text{COMPONENT\_STATE\_ASPECT} \rightarrow \text{STATE} \). Sometimes \( \text{state} \) will be used as a binary functional symbol of type \( \text{TRACE} \times \text{TIME} \rightarrow \text{STATE} \).
d) a binary function symbol \( \text{truth\_value} \) of type \( \text{STATE} \times \text{STATOM} \rightarrow \text{TRUTH\_VALUE} \)
e) a binary function symbol \( \land \) of type \( \text{STATPROP} \times \text{STATPROP} \rightarrow \text{STATPROP} \)
   - the same for \( \lor \) (or), \( \rightarrow \) (implication) and \( \leftrightarrow \) (equivalence) function symbols
f) a unary function symbol \( \text{not} \) of type \( \text{STATPROP} \rightarrow \text{STATPROP} \)
g) a binary function symbol \( \forall \) (for all) of type \( \text{STATE\_VARIABLE} \times \text{STATPROP} \rightarrow \text{STATPROP} \)
   - the same for the \( \exists \) (exists) function symbol

(5) a set of predicate symbols \( P \), among which:

a) a predicate symbol \( \text{holds} \) (\( \models \)) of type \( \text{STATE} \times \text{STATPROP} \).
b) \( = \) : an identity relation on arbitrary sorts
c) \( < \) : \( \text{TIME} \times \text{TIME} \) is the earlier than relation on time

**Definition 1.2 (TTL Signature)**

Let \( L_{\text{TTL}=\langle \text{OBJECT}, \text{STATE\_VARIABLE}, \text{TIME}, \text{STATE}, \text{TRACE}, \text{STATPROP}, C, \Phi, P \rangle} \) be a multi-sorted signature for the TTL language. With each variable \( x \in \text{STATE\_VARIABLE} \) a sort \( S \) is associated, written as \( x:S \). Then the terms and formulas of the language \( L \) are defined as follows.

**Definition 1.3 (TTL Terms)**

The terms of any sort \( S \) are inductively defined by:

1. If \( x:S \in \text{STATE\_VARIABLE} \), then \( x \) is a term of \( L_{\text{TTL}} \)
2. If \( c:S \in L_{\text{TTL}} \) is a constant symbol, then \( c \) is a term of \( L_{\text{TTL}} \)
3. If \( f \in L_{\text{TTL}} \) is an \( n \)-place function symbol and \( \tau_1, \ldots, \tau_n \) are terms of \( L_{\text{TTL}} \), then \( f(\tau_1, \ldots, \tau_n) \) is a term of \( L_{\text{TTL}} \)

**Definition 1.4 (TTL Formulæ)**

TTL-formulæ are defined inductively as follows:

A. The set of \( \text{atomic TTL-formulæ} \) is defined as:

1. If \( v_i \) is a term of sort \( \text{STATE} \), and \( u_i \) is a term of the sort \( \text{STATPROP} \), then \( \text{holds}(v_1, u_i) \) is an atomic TTL formulæ.
2. If \( \tau_1, \tau_2 \) are terms of any sort, then \( \tau_1 = \tau_2 \) is an atomic TTL formulæ. (further we shall use this predicate in form \( \tau_1 = \tau_2 \))
3. If \( t_1, t_2 \) are terms of sort \( \text{TIME} \), then \( t_1 < t_2 \) is an atomic TTL formulæ. (further we shall use this predicate in form \( t_1 < t_2 \), furthermore we shall use \( t_1 \leq t_2 \) for \( t_1 < t_2 \land t_1 = t_2 \))

B. The set of \( \text{well-formed TTL-formulæ} \) is defined as

1. Any atomic TTL-formulæ is a well-formed TTL-formulæ.
2. If \( F \) and \( G \) are well-formed TTL-formulæ, then so are \( \neg F \), \( (F \lor G) \), \( (F \Rightarrow G) \) and \( (F \equiv G) \).
3. If \( F \) is a well-formed TTL-formulæ containing \( x:S \) as a free variable, then \( (\forall x:S \ F(x(S))) \) and \( (\exists x:S \ F(x(S))) \) are well-formed TTL-formulæ.

**1.2. The Semantics of TTL**

An interpretation of a TTL formulæ is defined by the standard interpretation of order sorted predicate logic formulæ.

**Definition 1.5 (Interpretation)**

An \( \text{interpretation} \) of a TTL formulæ is defined by a mapping \( I \) that:

1. associates each sort symbol \( S \) to a certain subdomain \( D_S \), and if \( S \subseteq S' \) then \( D_S \subseteq D_{S'} \);
2. the subdomains for standard sorts is given below:

\( D_{\text{OBJECT}} = D_{\text{STATE}} = D_{\text{TIME}} = D_{\text{STATOM}} = D_{\text{TRUTH\_VALUE}} = \{ \text{true}, \text{false} \} \)
The semantics of connectives and quantifiers is defined in the standard way.

Definition 1.6 (TTL Model)
The model \( M \) for the language TTL is a pair \( M = \langle I, V \rangle \), where:
- \( I \) is an interpretation function, and
- \( V \) is a variable assignment function, mapping each variable \( x : S \) to an element of \( D_x \).

Definition 1.7 (Meaning of TTL terms)
Let \( M = \langle I, V \rangle \) be a model for TTL. Then the meaning of a term \( \tau \in \text{TTL} \), denoted by \( \tau^M \), is inductively defined by:
1. \( (x : S)\|^M = V(x) \),
2. \( (c : S)^M = I(c) \),
3. \( f(\tau_1, \ldots, \tau_k)^M = I(f)(\tau_1^M, \ldots, \tau_k^M) \).

Definition 1.8 (Truth definition for TTL)
Let \( M = \langle I, V \rangle \) be a model for TTL. Then the truth definition of TTL is inductively defined by:
1. \( \models_M P(\tau_1, \ldots, \tau_k) \) iff \( I(P) (\tau_1^M, \ldots, \tau_k^M) = \text{true} \)
2. \( \models_M \neg \phi \) iff \( \not\models_M \phi \)
3. \( \models_M \phi \wedge \psi \) iff \( \models_M \phi \) and \( \models_M \psi \)
4. \( \models_M \forall x : S (\phi(x)) \) iff \( \models_M^{\forall x : S} \phi(x) \) for all \( v \in D_x \)

The semantics of connectives and quantifiers is defined in the standard way.

1.3. Axioms of TTL
(1) Substitution function: for all functional symbols \( f \in L_{\text{TTL}} \) and for all \( x : S \in L_{\text{TTL}} \) and \( x^* \in S_{\text{VARS}} \) \( \text{sub}^{x^*}_x f(x) = f(x^*) \)
(2) Congruence of traces:
\( \forall \gamma_1, \gamma_2 \ [\forall t \ [\text{state}(\gamma_1, t) = \text{state}(\gamma_2, t) \] \Rightarrow \gamma_1 = \gamma_2] \)
(3) Equality of states:
\( \forall s_1, s_2 \ [\forall a : \text{STATOMS} \ [\text{truth_value}(s_1, a) = \text{truth_value}(s_2, a) \] \Rightarrow s_1 = s_2] \)
(4) Truth value in a state:
\( \text{holds}(s, p) \Leftrightarrow \text{truth_value}(s, p) = \text{true} \)
(5) State property semantics
   a. \( \text{holds}(s, (p_1 \land p_2)) \Leftrightarrow \text{holds}(s, p_1) \land \text{holds}(s, p_2) \)
   b. \( \text{holds}(s, (p_1 \lor p_2)) \Leftrightarrow \text{holds}(s, p_1) \lor \text{holds}(s, p_2) \)
   c. \( \text{holds}(s, \neg(p_1)) \Leftrightarrow \neg\text{holds}(s, p_1) \)
For any constant variable name \( x^* \) from the sort \( S_{\text{VARS}} \):
   d. \( \text{holds}(s, (\exists x^* : S_{\text{VARS}}, F(x^*))) \Leftrightarrow \exists x : S \text{ holds}(s, F(x)) \)
   e. \( \text{holds}(s, (\forall x^* : S_{\text{VARS}}, F(x^*))) \Leftrightarrow \forall x : S \text{ holds}(s, F(x)) \)
(6) Partial order axioms for the sort \( \text{TIME} \):
   a. \( \forall t t \leq t \)
   b. \( \forall t_1, t_2 \ [t_1 \leq t_2 \land t_2 \leq t_1 \] \Rightarrow t_1 = t_2 \)
2. FORMAL JUSTIFICATION FOR THE TRANSFORMATION PROCEDURE

Lemma 1 (Normalization lemma)
Let t be a given time point. If a formula $\delta(\gamma, t)$ only contains temporal relations such as $t' < t''$ and $t' \leq t''$, and atoms of the form $\text{state}(\gamma, t) \models p$ for some state formula $p$, then some state formula $q(t)$ can be constructed such that $\delta(\gamma, t)$ is equivalent to the formula $\delta^*(\gamma, t)$ of the form $\text{state}(\gamma, t) \models q(t)$.

Proof sketch for Lemma 1.
First in the formula $\delta(\gamma, t)$ replace all temporal relations such as $t' < t''$ and $t' \leq t''$ by $\text{state}(\gamma, t) \models t' < t''$ and $\text{state}(\gamma, t) \models t' \leq t''$ respectively. Then proceed by induction on the composition of the formula $\delta(\gamma, t)$.

1) conjunction: $\delta(\gamma, t)$ is $\delta_1(\gamma, t) \& \delta_2(\gamma, t)$
By induction hypothesis
$\delta_1(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p_1$ (which is $\delta_1^*(\gamma, t)$)
$\delta_2(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p_2$ (which is $\delta_2^*(\gamma, t)$)
Then
$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p_1 \& \text{state}(\gamma, t) \models p_2$ (which becomes $\delta^*(\gamma, t)$)

2) disjunction: $\delta(\gamma, t)$ is $\delta_1(\gamma, t) \lor \delta_2(\gamma, t)$
Again by induction hypothesis
$\delta_1(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p_1$ (which is $\delta_1^*(\gamma, t)$)
$\delta_2(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p_2$ (which is $\delta_2^*(\gamma, t)$)
Then
$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p_1 \lor \text{state}(\gamma, t) \models p_2$ (which becomes $\delta^*(\gamma, t)$)

3) negation: $\delta(\gamma, t)$ is $\neg \delta_1(\gamma, t)$
$\delta_1(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p_1$
$\delta(\gamma, t) \Leftrightarrow \neg \text{state}(\gamma, t) \models p_1$
$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models \neg p_1$ (which is $\delta^*(\gamma, t)$)

4) implication: $\delta(\gamma, t)$ is $\delta_1(\gamma, t) \Rightarrow \delta_2(\gamma, t)$
Again by induction hypothesis
$\delta_1(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p_1$ (which is $\delta_1^*(\gamma, t)$)
$\delta_2(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p_2$ (which is $\delta_2^*(\gamma, t)$)
Then
$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models p_1 \Rightarrow \text{state}(\gamma, t) \models p_2$ (which becomes $\delta^*(\gamma, t)$)

5) universal quantifier:
$\delta(\gamma, t) \Leftrightarrow \forall t' \text{state}(\gamma, t) \models p_1(t')$
$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models \forall t' p_1(t')$ (which is $\delta^*(\gamma, t)$)

6) existential quantifier:
$\delta(\gamma, t) \Leftrightarrow \exists t' \text{state}(\gamma, t) \models p_1(t')$
$\delta(\gamma, t) \Leftrightarrow \text{state}(\gamma, t) \models \exists t' p_1(t')$ (which becomes $\delta^*(\gamma, t)$)

Definition 2.1 (Uniqueness and correctness of time)
To relate time within a state property to time external to states a functional symbol present_time is used. Here time is assumed to have the properties of correctness and uniqueness:

Uniqueness of time
This expresses that present_time(t) is true for at most one time point t.
Let Proposition 1

The last two rules are assumed to be included into two following theories Th_{a\rightarrow m} and Th_{p\rightarrow a} as well.

Proposition 1

Let \( \varphi_p(\gamma, t) \) be a past statement for a given \( t \), \( \varphi_{\text{mem}}(\gamma, t) \) the memory formula for \( \varphi_p(\gamma, t) \), \( q_{\text{mem}}(l) \) the normalized memory state formula for \( \varphi_{\text{mem}}(\gamma, t) \), and \( \text{Th}_{a\rightarrow m} \) the executable theory from the interaction states for \( \varphi_p(\gamma, t) \) to the memory states. Then,

\[
\text{Th}_{a\rightarrow m} \models [ \varphi_p(\gamma, t) \iff \varphi_{\text{mem}}(\gamma, t) ]
\]

and

\[
\text{Th}_{a\rightarrow m} \models [ \varphi_p(\gamma, t) \iff \text{state}(\gamma, t) \iff \text{state}(\gamma, t) \iff q_{\text{mem}} ].
\]

Proof.

From the definitions of \( q_{\text{mem}}(l) \) and of \( \text{Th}_{a\rightarrow m} \) follows

\[
\text{Th}_{a\rightarrow m} \models [ \varphi_p(\gamma, t) \iff \varphi_{\text{mem}}(\gamma, t) ]
\]

Further by Lemma 2

\[
\text{Th}_{a\rightarrow m} \models [ \varphi_p(\gamma, t) \iff \text{state}(\gamma, t) \iff q_{\text{mem}}(l) ]
\]

Definition 2.5 (Normalized condition state formula)

The state formula constructed by Lemma 1 for \( \varphi_{\text{mem}}(\gamma, t, t_1) \) is called the normalized condition state formula for \( \varphi_{\text{mem}}(\gamma, t, t_1) \) and denoted by \( q_{\text{cond}}(l, t_1) \). Moreover, \( q_{\text{cond}}(l) \) is the state formula \( \forall t \' [ \text{present}_\text{time}(t') \rightarrow q_{\text{cond}}(l, t') ] \)

Lemma 3

If time has properties of correctness and uniqueness, then

\[
\varphi_{\text{mem}}(\gamma, t, t_1) \iff \text{state}(\gamma, t_1) \iff q_{\text{cond}}(l, t_1) \iff \text{state}(\gamma, t_1) \iff q_{\text{cond}}(l)
\]

Proof.

The lemma can be proven in the same manner as Lemma 2.
Definition 2.6 (Normalized preparation state formula)

The state formula constructed by Lemma 1 for $\varphi_{\text{prop}}(\gamma, t_1)$ is called the normalized preparation state formula for $\varphi_{\text{prop}}(\gamma, t_1)$ and denoted by $q_{\text{prep}}(t_1)$. Moreover, $q_{\text{prep}}$ is the state formula $\forall t' \{ \text{present_time}(t') \rightarrow q_{\text{prep}}(t') \}$

Lemma 4

If time has properties of correctness and uniqueness, then

$\varphi_{\text{prop}}(\gamma, t_1) \iff \text{state(}\gamma, t_1) = q_{\text{prep}}(t_1) \iff \text{state(}\gamma, t_1) = q_{\text{prep}}$ \hspace{1cm} (3)

Proof.

The lemma can be proven in the same manner as Lemma 2.

Definition 2.7 (Conditional preparation formula and normalized conditional preparation state formula)

Let $q_{\text{cond}}(t_1)$ be the normalized condition state formula for $\varphi_{\text{cond}}(\gamma, t_1)$ and $q_{\text{prep}}(t_1)$ the normalized preparation state formula for $\varphi_{\text{prop}}(\gamma, t_1)$. The formula $q_{\text{prep}}(\gamma, t)$ of the form $\text{state}(\gamma, t) = \forall t_1 > t [q_{\text{cond}}(t_1) \rightarrow q_{\text{prep}}(t_1)]$ is called the conditional preparation formula for $\varphi(\gamma, t)$.

The state formula $\forall t_1 > t [q_{\text{cond}}(t_1) \rightarrow q_{\text{prep}}(t_1)]$ is called the normalized conditional preparation state formula for $\varphi_{\text{prop}}(\gamma, t)$ and denoted by $q_{\text{prep}}(t)$. Moreover, $q_{\text{prep}}$ is the formula $\forall t' \{ \text{present_time}(t') \rightarrow q_{\text{prep}}(t') \}$.

Lemma 5

If time has properties of correctness and uniqueness, then

$\varphi_{\text{prop}}(\gamma, t_1) \iff \text{state}(\gamma, t_1) = q_{\text{prep}}(t) \iff \text{state}(\gamma, t_1) = q_{\text{prep}}$ \hspace{1cm} (4)

Proof.

The lemma can be proven in the same manner as Lemma 2.

Definition 2.8 (Executable theory from memory to preparation)

For any state atom $p$ occurring in $\varphi_{\text{cond}}(\gamma, t_1)$, expressed in the InteractionOnt(A) for the agent $A$:

$\forall t' \text{state}(\gamma, t') = p \Rightarrow \text{state}(\gamma, t') = [\text{memory}(t', p) \land \text{stimulus\_reaction}(p)]$

$\forall t', t' \text{state}(\gamma, t') = \text{memory}(t', p) \Rightarrow \text{state}(\gamma, t' + 1) = \text{memory}(t', p)$

$\forall t' \text{state}(\gamma, t') = q_{\text{mem}} \Rightarrow \text{state}(\gamma, t') = q_{\text{prep}}$

$\forall t', t \text{state}(\gamma, t') = [q_{\text{prep}} \land q_{\text{cond}}(t_1) \land \bigcap p \text{stimulus\_reaction}(p)] \Rightarrow \text{state}(\gamma, t') = q_{\text{prep}}$

$\forall t' \text{state}(\gamma, t') = [\text{stimulus\_reaction}(p) \land \neg \text{preparation\_for}(t' + c, a)] \Rightarrow \text{state}(\gamma, t' + 1) = \text{stimulus\_reaction}(p)$

$\forall t' \text{state}(\gamma, t') = [\text{preparation\_for}(t' + c, a) \land \neg \text{performing\_action}(a)] \Rightarrow \text{state}(\gamma, t' + c) = \text{preparation\_for}(t' + c, a)$,

where an action for which $\text{state}(\gamma, t' + c) = \text{performing\_action}(a)$ occurs in $\varphi(\gamma, t)$.

Proposition 2

Let $\varphi(\gamma, t)$ be a future statement for $t$ of the form $\forall t_1 > t [q_{\text{cond}}(\gamma, t_1) \Rightarrow q_{\text{act}}(\gamma, t_1)]$, where $q_{\text{cond}}(\gamma, t_1)$ is an interval statement, which describes a condition for one or more actions and $q_{\text{act}}(\gamma, t_1)$ is a (conjunction of) future statement(s) for $t_1$ which describes action(s) that are to be performed; let $q_{\text{prep}}(\gamma, t)$ be the conditional preparation formula for $\varphi(\gamma, t)$, $q_{\text{prep}}(t)$ be the normalized conditional preparation state formula for $\varphi_{\text{prop}}(\gamma, t)$, and $Th_{\text{m-up}}$ the executable theory for $\varphi(\gamma, t)$ from memory states to preparation states. Then,

$Th_{\text{m-up}} = [\varphi(\gamma, t) \iff q_{\text{prep}}(\gamma, t)]$

and

$Th_{\text{m-up}} = [\varphi(\gamma, t) \iff \text{state}(\gamma, t) = q_{\text{prep}}(t) \iff \text{state}(\gamma, t) = q_{\text{prep}}].$

Proof.

From the definition of $Th_{\text{m-up}}$, Lemmas 3 and 4 follows that

$Th_{\text{m-up}} = [\varphi_{\text{cond}}(\gamma, t_1) \iff q_{\text{cond}}(t_1)]$ \hspace{1cm} (5)

and

$Th_{\text{m-up}} = [\varphi_{\text{act}}(\gamma, t_1) \iff q_{\text{prep}}(\gamma, t_1)]$ \hspace{1cm} (6)

1 If a future formula does not contain a condition, then stimulus\_reaction atoms are generated from a corresponding past formula.
From (5), (6), definitions of the conditional preparation formula and the normalized conditional preparation state formulae, and the conditions of the proposition it follows

\[ \text{Th}_{\text{m-p}} \models [ \forall \gamma, t \exists q_{\text{cond}}(t, t_\gamma) \rightarrow q_{\text{prop}}(\gamma, t_\gamma) ] \models \forall t_\gamma, t [ q_{\text{prop}}(\gamma, t_\gamma) ] \models q_{\text{prop}}(\gamma, t) \]

And from Lemma 5 follows

\[ \text{Th}_{\text{m-p}} \models [ \forall \gamma, t \exists state(\gamma, t) \models q_{\text{prop}}(t, t_\gamma) \models state(\gamma, t) = q_{\text{prop}}(t) \models state(\gamma, t) = q_{\text{prop}} ] \]

Proposition 3

Let \( \varphi(\gamma, t) \) be a past statement for \( t \) and \( \varphi(\gamma, t) \) be a future statement for \( t \). Let \( \varphi_{\text{mem}}(\gamma, t) \) be the memory formula for \( \varphi(\gamma, t) \) and \( \varphi_{\text{prop}}(\gamma, t) \) the conditional preparation formula for \( \varphi(\gamma, t) \). Then

\[ [ \varphi_{\text{prop}}(\gamma, t) \Rightarrow \varphi(\gamma, t) ] \iff [ \varphi_{\text{mem}}(\gamma, t) \Rightarrow \varphi_{\text{prop}}(\gamma, t) ] \]

Proof.

From the Proposition 1 and the Proposition 2 follows

\[ \varphi(\gamma, t) \Rightarrow \varphi_{\text{mem}}(\gamma, t) \text{ and } \varphi(\gamma, t) \Rightarrow \forall t_\gamma, t [ q_{\text{prop}}(\gamma, t, t_\gamma) ] \]

Then,

\[ [ \varphi(\gamma, t) \Rightarrow \varphi(\gamma, t) ] \iff [ \varphi_{\text{mem}}(\gamma, t) \Rightarrow \forall t_\gamma, t [ q_{\text{prop}}(\gamma, t, t_\gamma) ] ] \]

So it has been proven that \([ \varphi_{\text{prop}}(\gamma, t) \Rightarrow \varphi(\gamma, t) ] \iff [ \varphi_{\text{mem}}(\gamma, t) \Rightarrow \varphi_{\text{prop}}(\gamma, t) ] \]

Definition 2.9 (Executable theory from preparation to output)

For a given \( \varphi(\gamma, t) \) the executable theory from the preparation to the output state(s) \( \text{Th}_{\text{p-o}} \) consists of the formula

\[ \forall t' \models state(\gamma, t') = \text{preparation_for}(t'+c, a) \Rightarrow state(\gamma, t'+c) = \text{performing_action}(a), \]

where \( c \) is a number and \( a \) an action for which \( state(\gamma, t'+c) = \text{performing_action}(a) \) occurs in \( \varphi(\gamma, t) \).

Definition 2.10 (Executable specification)

An executable specification \( \pi(\gamma, t) \) for the agent \( A \) is defined by a union of the dynamic properties from the executable theories \( \text{Th}_{\text{p-m}} \), \( \text{Th}_{\text{m-p}} \) and \( \text{Th}_{\text{p-o}} \).

Definition 2.11 (Coinciding traces)

Two traces \( \gamma_1, \gamma_2 \) coincide on ontology Ont (denoted by coinciding_on(\( \gamma_1, \gamma_2, \text{Ont} \))) iff

\[ \forall t_\gamma \models t(\gamma) \models a \models state(\gamma_1, t) = a \models state(\gamma_2, t) = a \]

where \( t(\gamma) \) is the set of ground atoms expressed in terms of Ont.

Definition 2.12 (Refinement of an externally observable property)

Let \( \varphi(\gamma, t) \) be an externally observable dynamic property for agent \( A \). An executable specification \( \pi(\gamma, t) \) for \( A \) refines \( \varphi(\gamma, t) \) iff

1. \( \forall t_\gamma, t \models \pi(\gamma, t) \Rightarrow \varphi(\gamma, t) \)
2. \( \forall t_\gamma, t [ \varphi(\gamma, t) \Rightarrow [ \exists t_2 \text{ coincide_on}(\gamma_1, \gamma_2, \text{InteractionOnt}(A)) \land t(\gamma, t) ] ] \]

Note that for any past interaction statement \( \varphi_{\text{p}}(\gamma, t) \) and future interaction statement \( \varphi_{\text{f}}(\gamma, t) \) the following holds:

\[ \forall t_\gamma, t_2 [ \text{coinciding_on}(\gamma_1, \gamma_2, \text{InteractionOnt}) \Rightarrow [ \varphi_{\text{p}}(\gamma_1, t) \Rightarrow \varphi_{\text{f}}(\gamma_2, t) \land \varphi_{\text{f}}(\gamma_1, t) \Rightarrow \varphi(\gamma_2, t) ] ] \]

Lemma 6

Let \( \varphi(\gamma, t) \) be a dynamic property expressed using the state ontology Ont. Then the following holds:

1. \( \text{coinciding_on}(\gamma_1, \gamma_2, \text{Ont}) \land \text{coinciding_on}(\gamma_2, \gamma_1, \text{Ont}) \Rightarrow \text{coinciding_on}(\gamma_1, \gamma_1, \text{Ont}) \)
2. \( \text{coinciding_on}(\gamma_1, \gamma_2, \text{Ont}) \Rightarrow [ \varphi(\gamma_1, t) \Rightarrow \varphi(\gamma_2, t) ] \)

Proof sketch.

The transitivity property (1) follows directly from the definition of coinciding traces for \( \text{coinciding_on}(\gamma_1, \gamma_2, \text{Ont}) \) and \( \text{coinciding_on}(\gamma_2, \gamma_3, \text{Ont}) \):

\[ \forall t_\gamma \models t(\gamma, t) = a \models state(\gamma_3, t') = a \iff \text{coinciding_on}(\gamma_1, \gamma_3, \text{Ont}) \]

From
∀ γ₁, γ₂ [ coincide_on(γ₁, γ₂, Ont) ] ⇒ \( ∀ γ \) [ ϕ₁(γ₁, t) ⇔ ϕ₂(γ₂, t) & ϕ₁(γ₁, t) ⇔ ϕ₂(γ₂, t) ]

follows that ϕ(γ₁, t) ⇔ ϕ(γ₂, t).

Note that for any past interaction statement ϕ₁(γ, t) and future interaction statement ϕ₂(γ, t) the following holds:

∀ γ₁, γ₂ [ coincide_on(γ₁, γ₂, InteractionOnt) ] ⇒ [ ϕ₁(γ₁, t) ⇔ ϕ₂(γ₂, t) & ϕ₁(γ₁, t) ⇔ ϕ₂(γ₂, t) ]

**Theorem**

If the executable specification πᵦ(γ, t) refines the external behavioral specification φᵦ(γ, t) of agent A, and ψ(γ, t) is a dynamic interaction property of agent A in its environment, expressed using the interaction ontology InteractionOnt(𝔸), then

\[ [ ∀ γ \ [ π \瑞典(γ, t) \Rightarrow ψ(γ, t) ] ] ⇔ [ ∀ γ \ [ φ \瑞典(γ, t) \Rightarrow ψ(γ, t) ] ] \]

**Proof sketch for Theorem.**

\( \equiv \) is direct:

from π(γ, t) ⇒ ϕ(γ, t) and \( ∨ ϕ(γ, t) \Rightarrow ψ(γ, t) \) it follows \( ∨ π(γ, t) \Rightarrow ψ(γ, t) \).

\( \Rightarrow \) runs as follows:

Suppose ϕ(γ, t) holds for all i, then since π(γ) refines ϕ(γ, t), then according to the definition of refinement of an externally observable property exists such a γ that π(γ) and coincide_on(γ, γ₁, InteractionOnt(𝔸)).

Due to Lemma 6, this γ still satisfies all ϕ(γ₁, t) (i.e., ϕ(γ₁, t) holds for all i).

Proceed with γ to obtain a γ₂ and further for all i to reach a state γₙ, for which π(γₙ) holds for all i,

and coincide_on(γ, γₙ, InteractionOnt(𝔸)),

and ϕ(γₙ) holds for all i.

From

\( ∀ γ \forall i [ π(i) \Rightarrow ϕ(i) ] \),

and

\( ∀ γ [ ∨ π(γ) \Rightarrow ψ(γ, t) ] \)

it follows that \( ∀ γ ∨ ϕ(γ) \Rightarrow ψ(γ) \).

So it has been proven that \( ∀ γ ∨ ϕ(γ) \Rightarrow ψ(γ) \). ■

3. **TRANSFORMATION INTO THE FINITE STATE TRANSITION SYSTEM FORMAT**

According to the Definition 2.10 the executable specification of an agent’s behavior consists of the union of three theories \( Th_{on-om} \), \( Th_{on-op} \) and \( Th_{op-op} \), which in turn contain a number of executable dynamic properties. These dynamic properties can be translated into transition rules for a finite state transition system, based on which the same traces are generated as by executing the dynamic properties. For this purpose we use the predicate present_time(t) introduced earlier, which is only true in a state for the current time point t. Further the executable properties from the executable specification, translated into the transition rules are given.

**Time increment rules:**

- present_time(0) ∧ ¬p → present_time(1)
- present_time(t) ∧ ¬q$_{mem}$ ∧ ¬p → present_time(t+1)
- present_time(t) ∧ q$_{prop}$ ∧ ¬q$_{cond}(t) ∧ ¬p → present_time(t+1)
- present_time(t) ∧ q$_{prop}$ → present_time(t+1)

**Memory state creation rule:**

For any state atom p occurring in φ$_{cond}(γ, t, t_1)$, expressed in the InteractionOnt(𝔸) for agent A:

present_time(t) ∧ p → [ memory(t, p) ∧ stimulus_reaction(p) ]

For all other state atoms p
present_time(t) \land p \rightarrow \text{memory}(t, p)

\textit{Memory persistence rule:}
\text{memory}(t, p) \rightarrow \text{memory}(t, p)

\textit{Conditional preparation generation rule:}
q_{\text{mem}} \rightarrow \text{conditional}\_\text{preparation}\_\text{for}(a),
where a an action for which state(γ, t'+c) |= \text{performing\_action}(a) occurs in φ(γ, t).

\textit{Preparation state creation rule:}
present\_time(t') \land \text{conditional}\_\text{preparation}\_\text{for}(a) \land q_{\text{cond}}(t) \land \bigcap p \rightarrow \text{preparation}\_\text{for}(t'+c, a)

\text{for every subformula of the form}
\text{present}\_\text{time}(t') \rightarrow \text{preparation}\_\text{for}(t'+c, a)

\text{that occurs in q}_{\text{prop}}.

\textit{Preparation state persistence rule:}
\text{preparation}\_\text{for}(t+c, a) \land \neg\text{performing\_action}(a) \rightarrow \text{preparation}\_\text{for}(t+c, a)

\textit{Stimulus reaction state persistence rule:}
present\_time(t') \land \text{stimulus}\_\text{reaction}(p) \land \neg\text{preparation}\_\text{for}(t'+c, a) \rightarrow \text{stimulus}\_\text{reaction}(p)

\textit{Output state creation rule:}
\text{preparation}\_\text{for}(t+c, a) \land \text{present}\_\text{time}(t+c-1) \rightarrow \text{performing}\_\text{action}(a), \text{where a is an action.}

\section{4. TRANSFORMATION INTO THE SMV MODEL SPECIFICATION FORMAT}

For automatic verification of general properties of an agent in its environment by means of model checking techniques, a corresponding to the external agent behavioral specification representation of a finite state transition system should be translated into the input format of one of the existing model checkers. The model checker SMV has been chosen as a verification tool for two reasons. First, the input language of SMV is syntactically and semantically similar to the general description of a finite state transition system, which facilitates automatic translation into the SMV input format. Second, SMV uses efficient symbolic algorithms to traverse a model and the expressive temporal logic CTL for specifying properties to check.

A procedure for translating from a general representation for a finite state transition system into the model specification format for the SMV model checker has been developed. In Section 4.1 the main steps of the procedure are explained. Then, in Section 4.2 the complexity issues of a specification, generated by application of this procedure, are considered.

\subsection{4.1 Transformation procedure}
Let us describe the transformation procedure, which is automatically performed by dedicated software that has been developed.

First, using the standard rules \cite{1} \(q_{\text{mem}}(t)\) and \(q_{\text{cond}}(t)\) expressions for each dynamic property \(DP_n\) are transformed into the prenex normal form. Then, for each dynamic property the described below steps 1-3 are applied first to \(q_{\text{mem}}(t)\) and then to \(q_{\text{cond}}(t)\). After that conditional preparation generation rules are added by performing the step 4. Finally, the preparation and action state creation rules are generated for each dynamic property by performing the step 5.

\textbf{Step 1.} For each occurrence of an existential quantifier of the form \(\exists t1\ P(t1)\), where \(t1\) is a time variable name and \(P(t1)\) is some predicate of the form \(\text{memory}(\text{observed}(t1, \text{obs\_event}))\) or \(\neg\text{memory}(\text{observed}(t1, \text{obs\_event}))\), \text{obs\_event} is some atom, and for each occurrence of a universal quantifier of the form \(\forall t1\ P(t1)\), create an atom (a label) \(t1\) and add to the specification the following:

\begin{verbatim}
t1: boolean;
init(t1):=0;
obs\_event: boolean;
init(obs\_event):=0;
\end{verbatim}

\textbf{Step 2.} For each existentially quantified time variable and universally quantified time variable that is not in the scope of any existential quantifier with a time variable:
(a) For each occurrence of the expression $Q \ t_1, t_2 \ R \ t_1 \ memory(\text{observed}(t_1, \text{obs}_event))$, where $Q$ is either an existential or a universal quantifier, $R$ is the comparison relation for the linear ordered time line: $R=\{<, \leq\}$; $t_1$ and $t_2$ are time variables, add to the specification the following rules:

```plaintext
next(t_1):= case
    t_2 & obs_event: 1; //memory state creation
    !t_2: 0;
    1: t_1;       //persistence of memory
esac;
```

(b) For each occurrence of the expression $Q \ t_1, t_2 \ R \ t_1 \neg memory(\text{observed}(t_1, \text{obs}_event))$, add to the specification the following rules:

```plaintext
next(t_1):= case
    t_2 & !obs_event: 1;
    !t_2: 0;
    1: t_1;
esac;
```

**Step 3.** For each expression of the form $\exists t_1, t_2 \forall t_3 \ [ t_3 R t_2 \land t_1 R t_3 \land memory(\text{observed}(t_1, \text{obs}_event1)) \land memory(\text{observed}(t_2, \text{obs}_event2)) \land P_3(t_3) ]$:

(a) if $P_3(t)$ is of the form $memory(\text{observed}(t_3, \text{obs}_event))$

i. For $t_3 < t_2$ and $t_1 < t_3$ add to the specification the following rules:

```plaintext
t_3 t_1_eq: boolean ;
init(t_3 t_1_eq):=0;
next(t_3 t_1_eq):= case
    t_1: 1;
    1: 0;
esac;
next(t_1):= case
    t_2 & !obs_event2 & !t_2 & t_3 t_1_eq & !obs_event3: 0;
    !t_2: 0;
    1: t_1;
esac;
next(t_3):= case
    !t_1: 0;
    !t_2 & !obs_event3: 0;
    !t_2 & obs_event3: 1;
    1: t_3;
esac;
```

ii. For $t_3 < t_2$ and $t_1 \leq t_3$ add to the specification the following rules:

```plaintext
t_3 t_1_eq: boolean ;
init(t_3 t_1_eq):=0;
next(t_3 t_1_eq):= case
    t_1: 1;
    1: 0;
esac;
next(t_1):= case
    t_2 & t_3 t_1_eq & !obs_event3: 0;
    !t_2: 0;
    1: t_1;
esac;
next(t_3):= case
    !t_1: 0;
    !t_2 & !obs_event3: 0;
    !t_2 & obs_event3: 1;
    1: t_3;
esac;
```

iii. For $t_3 \leq t_2$ and $t_1 < t_3$ add to the specification the following rules:

```plaintext
next(t_1):= case
    !obs_event2 & !t_2 & !obs_event3: 0;
```
1: t1;
next(t3) := case
  t1: 0;
  !obs_event2 & !t2 & !obs_event3: 0;
  !obs_event2 & !t2 & obs_event3: 1;
1: t3;
esac;

iii. For $t_3 \leq t_2$ and $t_1 \leq t_3$ add to the specification the following rules:

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

(b) If $P_3(t)$ is of the form $\neg \text{memory}(\text{observed}(t_3, \text{obs_event}))$

i. For $t_3 < t_2$ and $t_1 < t_3$ add to the specification the following rules:

neg_t3t1_eq: boolean;
init(neg_t3t1_eq):=0;
next(neg_t3t1_eq) := case
  t1: 1;
1: 0;
esac;
next(t1) := case
  !obs_event2 & !t2 & neg_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;

ii. For $t_3 < t_2$ and $t_1 \leq t_3$ add to the specification the following rules:

neg_t3t1_eq: boolean;
init(neg_t3t1_eq):=0;
next(neg_t3t1_eq) := case
  t1: 1;
1: 0;
esac;
next(t1) := case
  !t2 & neg_t3t1_eq & obs_event3: 0;
1: t1;
esac;
next(t3) := case
  t1: 0;
  !t2 & obs_event3: 0;
  !t2 & !obs_event3: 1;
1: t3;
esac;

iii. For $t_3 \leq t_2$ and $t_1 < t_3$ add to the specification the following rules:
next(t1) := case
    !obs_event2 & !t2 & 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;

iii. For t3 ≤ t2 and t1 ≤ t3 add to the specification the following rules:

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

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 q_{mem}(γ, t)
    i: 1: 0;
esac;

Step 5. For each action predicate performing_action(act) in a formula q_{act}(t) add to the specification the following rules:

next(fprep_act) := case
    fmemN & ^tj: 1; //conjunction of all labels, created based on q_{prep}(γ, t)
    j: 1: 0;
esac;
next(act) := case
    fprep_act: 1;
    1: 0;
esac;

4.2 Complexity issues

The complexity of the representation of the obtained executable model is linear in size of the non-executable behavioral specification. More specifically, the non-executable specification is related to the 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 q_{mem}(γ, t) and q_{prep}(γ, t1) respectively) and corresponding update rules are introduced;
(5) for every action specified in q_{act}(γ, t1) 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 q_{prep}(γ, t1) are introduced.
For verifying an executable model in the SMV OBDD-based symbolic model checking algorithms are used; the study of complexity of such algorithms is given in [2].

5. EXAMPLE

We illustrate the proposed approach by an example from the studies of animal behavior. In this example we investigate two cases of a laboratory mouse behavior (i.e., delayed-response and motivation-based behavior) in two different experimental settings. By means of the approach introduced in this paper we are aiming at determining what type of behavior will bring the agent to the satisfaction state in both environmental settings.

The initial situation is identical for both experimental settings and defined as follows: the mouse is placed in front of a transparent screen that separates it from a piece of food that is put at the position p1 behind the screen. The other possible position for placing food is p2. The mouse is able to observe the position of food and of the screen. In the first experimental setting at some moment after food has been put, a cup is placed covering the food at position p1, 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.

According to the proposed approach, first for each experimental setting an external behavioral specification is formally defined. The specification consists of environmental properties and externally observable behavioral properties of the agent. The formal behavioral specifications for the experiments are given below.

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

\[ \exists t1, t2, t3 \ (t2 > t1) \land \text{state}(t2) = \text{cup_at}(p1) \land \text{state}(t1) = \text{food_at}(p1) \land \text{state}(t3) = \text{not_screen} \]

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

\[ \forall t4 \ (t4 > t1) \implies [\text{food_at}(X) \land \text{not(mouse_sat)} \land \text{not(food_taken_away_from}(X))] \implies \text{state}(t4) = \text{food_at}(X), \text{where } X \in \{p1, p2\} \]

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

\[ \forall t5 \ (t5 > t4) \implies \text{not_screen} \implies \text{state}(t5) = \text{not_screen} \]

EP4: After placing the cup it will not be removed

\[ \forall t6 \ (t6 > t5) \implies \text{cup_at}(t6) \implies \text{cup_at}(t6), \text{where } X \in \{p1, p2\} \]

Properties that define the externally observable behavior of the mouse:

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

\[ \forall t7 \ (t7 > t6) \implies [\text{obsered}(X), \text{where } X \in \{\text{not_screen}, \text{screen}\}] \]

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

\[ \forall t9 \ (t9 > t8) \implies [\text{observed}(X), \text{where } X \in \{\text{food_at}(Y), \text{not(food_at}(Y))\} \land Y \in \{p1, p2\}] \]

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

\[ \forall t11 \ (t11 > t10) \implies [\text{food_taken_away_from}(X) \land \text{not(cup_at}(t10) \implies \text{observed}(X), \text{where } X \in \{\text{food_at}(Y), \text{not(food_at}(Y))\} \land Y \in \{p1, p2\}] \]

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

\[ \forall t13 \ (t13 > t12) \implies [\text{performing_action(goto}(X)) \implies \text{mouse_at}(t13), \text{where } X \in \{p1, p2\}] \]

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

\[ \forall t15 \ (t15 > t14) \implies [\text{mouse_at}(X) \land \text{food_at}(X) \implies \text{mouse_sat}(t15), \text{where } X \in \{p1, p2\}] \]

BP6: The mouse consumes food completely.

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 t_{32}, t_{33}, t_{34}, t_{33}\lt t_{32} & t_{33}\lt t_{34} & t_{33}=t_{34} \text{ state}(\gamma, t_{33}) |\Rightarrow [\text{cup}_1(p1) & \text{cup}_2(p2)] & \text{state}(\gamma, t_{32}) |\Rightarrow \text{food}_1(p1) & \text{state}(\gamma, t_{34}) |\Rightarrow [\text{food}_1(p1)] \}


Consider the case when the mouse has the delayed-response type of behavior. It is described by a property expressing that 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. By the example of this property let us demonstrate how the external behavioral specification of
the agent is transformed into the executable internal behavioral specification. The complete description of the finite state transition system for the first experiment is given below:

\[
\begin{align*}
\text{present\_time}(t) & \land \neg(\exists t_{13}, t_{13} < t \land \text{memory}(t_{13}, \text{observed}(\text{food\_at}(p_1)))) \land (\forall t_{14}, t_{13} < t_{14} \leq t \Rightarrow \\
& \neg(\text{memory}(t_{14}, \text{observed}(\neg(\text{food\_at}(p_1)))))) \Rightarrow \text{present\_time}(t+1) \\
\text{present\_time}(t) & \land \neg(\exists t_{131}, t_{131} < t \land \text{memory}(t_{131}, \text{observed}(\text{food\_at}(p_2)))) \land (\forall t_{141}, t_{131} < t_{141} \leq t \Rightarrow \\
& \neg(\text{memory}(t_{141}, \text{observed}(\neg(\text{food\_at}(p_2)))))) \Rightarrow \text{present\_time}(t+1) \\
\text{present\_time}(t) & \land \text{qcprep8} \land \neg(\exists t_{12}, t_{12} \prec t \land \text{memory}(t_{12}, \text{observed}(\neg(\text{screen})))) \Rightarrow \text{present\_time}(t+1) \\
\text{present\_time}(t) & \land \text{qcprep9} \land \neg(\exists t_{121}, t_{121} \prec t \land \text{memory}(t_{121}, \text{observed}(\neg(\text{screen})))) \Rightarrow \text{present\_time}(t+1) \\
\text{present\_time}(t) & \land \text{preparation}(\text{goto}(p_1)) \Rightarrow \text{present\_time}(t+1) \\
\text{present\_time}(t) & \land \text{preparation}(\text{goto}(p_2)) \Rightarrow \text{present\_time}(t+1) \\
\text{present\_time}(t) & \land \text{observed}(\text{food\_at}(p_1)) \land \neg(\text{cup\_at}(p_1)) \Rightarrow \text{memory}(t, \text{observed}(\text{food\_at}(p_1))) \\
\text{present\_time}(t) & \land \text{observed}(\text{food\_at}(p_2)) \land \neg(\text{cup\_at}(p_2)) \Rightarrow \text{memory}(t, \text{observed}(\text{food\_at}(p_2))) \\
\text{present\_time}(t) & \land \text{observed}(\neg(\text{food\_at}(p_1))) \land \neg(\text{cup\_at}(p_1)) \Rightarrow \text{memory}(t, \text{observed}(\neg(\text{food\_at}(p_1)))) \\
\text{present\_time}(t) & \land \text{observed}(\neg(\text{food\_at}(p_2))) \land \neg(\text{cup\_at}(p_2)) \Rightarrow \text{memory}(t, \text{observed}(\neg(\text{food\_at}(p_2)))) \\
\text{present\_time}(t) & \land \text{observed}(\text{food\_taken\_away\_from}(p_1)) \land \neg(\text{cup\_at}(p_1)) \Rightarrow \text{memory}(t, \text{observed}(\text{food\_taken\_away\_from}(p_1))) \\
\text{present\_time}(t) & \land \text{observed}(\text{food\_taken\_away\_from}(p_2)) \land \neg(\text{cup\_at}(p_2)) \Rightarrow \text{memory}(t, \text{observed}(\text{food\_taken\_away\_from}(p_2))) \\
\text{present\_time}(t) & \land \text{observed}(\text{screen}) \Rightarrow \text{memory}(t, \text{observed}(\text{screen})) \\
\text{present\_time}(t) & \land \text{observed}(\neg(\text{screen})) \Rightarrow \text{memory}(t, \text{observed}(\neg(\text{screen}))) \\
\text{memory}(t, \text{observed}(\text{food\_at}(p_1))) & \Rightarrow \text{memory}(t, \text{observed}(\text{food\_at}(p_1))) \\
\text{memory}(t, \text{observed}(\text{food\_at}(p_2))) & \Rightarrow \text{memory}(t, \text{observed}(\text{food\_at}(p_2))) \\
\text{memory}(t, \text{observed}(\neg(\text{food\_at}(p_1)))) & \Rightarrow \text{memory}(t, \text{observed}(\neg(\text{food\_at}(p_1)))) \\
\text{memory}(t, \text{observed}(\neg(\text{food\_at}(p_2)))) & \Rightarrow \text{memory}(t, \text{observed}(\neg(\text{food\_at}(p_2)))) \\
\text{memory}(t, \text{observed}(\text{food\_taken\_away\_from}(p_1))) & \Rightarrow \text{memory}(t, \text{observed}(\text{food\_taken\_away\_from}(p_1))) \\
\text{memory}(t, \text{observed}(\text{food\_taken\_away\_from}(p_2))) & \Rightarrow \text{memory}(t, \text{observed}(\text{food\_taken\_away\_from}(p_2))) \\
\text{memory}(t, \text{observed}(\text{screen})) & \Rightarrow \text{memory}(t, \text{observed}(\text{screen})) \\
\text{memory}(t, \text{observed}(\neg(\text{screen}))) & \Rightarrow \text{memory}(t, \text{observed}(\neg(\text{screen}))) \\
\text{food\_at}(p_1) & \land \neg(\text{mouse\_sat}) \land \neg(\text{food\_taken\_away\_from}(p_1)) \Rightarrow \text{food\_at}(p_1) \\
\text{food\_at}(p_2) & \land \neg(\text{mouse\_sat}) \land \neg(\text{food\_taken\_away\_from}(p_2)) \Rightarrow \text{food\_at}(p_2) \\
\neg(\text{screen}) & \Rightarrow \neg(\text{screen}) \\
\text{cup\_at}(p_1) & \Rightarrow \text{cup\_at}(p_1)
\end{align*}
\]
Using the state transition system representation simulations of the agent internal dynamics and verification of the agent model with respect to a general system property can be performed. As has been mentioned before, the general property for the both experiments represents a satisfactory condition, namely for all traces if the screen is removed and food is hidden under the cup at the position p1, then the mouse will be eventually satisfied. Or in CTL,

\[ AG(\neg \text{screen} \land \text{food(at(p1))} \land \text{cup(at(p1))} \rightarrow AF \text{mouse_sat}) \]  

where \( \Delta \) is a path quantifier defined in CTL, meaning “for all computational paths”. \( \mathcal{G} \) and \( \mathcal{F} \) are temporal quantifiers that correspond to “globally” and “eventually” respectively.

In order to perform verification by means of SMV model checker, the general description of the finite state transition system has been automatically translated into the SMV model specification format. The automatic verification showed that the property (7) satisfies the model of the agent delayed-response behavior.

Now, consider the case of the motivation-based behavior of the animal. It is described by a property which expresses that if the screen is removed and the mouse is not satisfied (has hunger) and it observed a position of food before, then it will start searching for food first at the position of last observation and then, if it can not find food there, it will continue searching at the other position. Such specification of behavior can be attributed for example to an animal that feels strong hunger. The model of agent behavior for this case also satisfies the general property (7).

In the second experimental setting 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. 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 global property for verifying expresses that for all traces if the screen is removed and food is hidden behind the cup at the position p2, then the mouse will be eventually satisfied.

\[ AG(\neg \text{screen} \land \text{food(at(p2))} \land \text{cup(at(p2))} \rightarrow AF \text{mouse_sat}) \]  

The automatic verification in SMV showed that the model of the agent behavior for the delayed-response case does not satisfy the property (8). 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, which caused the failure of the satisfaction property. As has been confirmed by automatic verification, the external behavioral specification for the case of motivation-based behavior satisfies the property (8).

From the results of verification of external behavioral specifications for both cases of behavior in both experimental settings with respect to the satisfaction properties (7) and (8) we draw the conclusion that the mouse that manifests motivation-based behavior fits more for surviving in the world, described by the experimental conditions than the mouse that has the delayed-response behavior.

REFERENCES