Verifying Interlevel Relations within Multi-Agent Systems

Sharpanskykh, A.; Treur, J.

2006

Verifying Interlevel Relations within Multi-Agent Systems: formal theoretical basis

Technical Report: TR-1701AI

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

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

In the general case, at any aggregation level a behavioral specification for a multi-agent system component consists of dynamic properties expressed by complex temporal relations in TTL, which therefore does not allow direct application of automatic verification procedures, more specifically, model checking techniques, used in this paper. In order to apply model checking techniques it is needed to transform an original behavioral specification of a certain aggregation level into a model based on a finite state transition system. In order to obtain this, as a first step a behavioral description for the lower aggregation level is replaced by one in executable (temporal) format. As a solution, an automated substitution of the behavioral specification for the component by an executable specification (expressed in an executable temporal language) is put forward. The justification 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 is described in Section 4.

In Sections 5 the application of the proposed approach is illustrated by a paradigmatic example. The SMV specification for the example is given in the Appendix A.

1. TTL Syntax and Semantics

The language TTL, short for Temporal Trace Language, is a variant of order-sorted predicate logic. Whereas the 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 State Language

For expressing state properties of a system 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. In order to represent different ontological entities of a system a number of different syntactical sorts are used. Thus, the state properties are specified using a multi-sorted first-order predicate language L_{STATE} with the vocabulary, specified by a signature:

Definition 1.1 (State Language Signature)

A signature ℓ_{SL} is a tuple, $\ell_{SL}=<\boldsymbol{S}; \boldsymbol{C}; \boldsymbol{P}; \boldsymbol{f}>$, where $\boldsymbol{S}$ is a set of sort names; $\boldsymbol{C}$ is a set of constants of each sort; $\boldsymbol{P}$ is a set of predicate symbols; $\boldsymbol{f}$ is a set of functional symbols.

Let $\ell_{SL}=<\boldsymbol{S}; \boldsymbol{C}; \boldsymbol{P}; \boldsymbol{f}>$ be a signature for L_{STATE}. We assume that the set of variables is given and that with each variable x from this set a sort S is associated, written as x:S. Then the terms and formulae of the language L_{STATE} are defined as follows.

Definition 1.2 (Terms of the state language)

The terms of any sort S are inductively defined by:

1. If x:S is a variable of the state language, then x is a term of L_{STATE}.
2. If $c \in \boldsymbol{C}$ is a constant symbol, then c is a term of L_{STATE}.
3. If $f \in \boldsymbol{f}$ is an n-place function symbol $S_1 \times \ldots \times S_n \rightarrow S$ and $\tau_1,\ldots, \tau_n$ are terms of L_{STATE}, such that $\tau_i \in S_i^{TERMS}$ (a set of all terms constructed using the sort S_i), then $f(\tau_1,\ldots, \tau_n) \in S^{TERMS}$ is a term of $L_{STATE.}$

Definition 1.3 (Formulae of the state language)

The formulae of L_{STATE} are inductively defined as follows:

1. If P: S_1 x S_2 x … x S_n is a predicate symbol from ***P*** and τ_1, τ_2…, τ_n are terms of L_{STATE}, such that $\tau_i \in S_i^{TERMS}$, then P(τ_1, τ_2,…, τ_n) is a formula of $L_{STATE.}$
2. If τ_1 and τ_2 are terms of the same sort S, then $\tau_1 = \tau_2$ is a formula of $L_{STATE.}$
3. If φ and ψ are the formulae of L_S, then ~φ, (φ|ψ), (φ&ψ), (φ⇒ψ) and (φ⇔ψ) are formulae of L_{STATE}.
4. If φ a formula of L_S containing x as a free variable, then (∀x φ) and (∃x φ) are formulae of L_{STATE}.

1.2. The Language of TTL

In the language TTL formulae of the state language L_{STATE} are used as objects. To provide names of object language formulae φ in the TTL the operator (*) is used (written as φ*). Then, by means of the defined in the TTL *holds* predicate these objects are evaluated in states defined within TTL. The language TTL (L_{TTL}) has a signature ℓ_{TTL} that facilitates the specification of and reasoning about the dynamics of systems:

Definition 1.4 (TTL Signature)

A signature consists of the symbols of the following classes:

(1) a number of 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 with for each time point a state), STATPROP (a set of all names for state properties expressed using the state language); and VALUE (an ordered set of numbers). Furthermore, for every sort S from L_{STATE} three TTL sorts exist: the sort S^{VARS}, which contains all variable names of sort S, and the sort S^{GTERMS}, which contains names of all ground terms, constructed using sort S. Sorts S^{GTERMS} and S^{VARS} are subsorts of a sort S^{TERMS}.

(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; v with subscripts and superscripts for variables of the sort VALUE.

(3) a set of function symbols **Φ**, among which:
   a) a function symbol *state* of type TRACE x TIME → STATE.
   b) functional symbols ∧, ∨, →, ↔: STATPROP x STATPROP → STATPROP; not: STATPROP → STATPROP.
   c) functional symbols ∀: S^{VARS} x STATPROP → STATPROP, and ∃: S^{VARS} x STATPROP → STATPROP for every sort S.
   d) functional symbols –, +, /, •: TIME x VALUE → TIME.
   e) functional symbols –, +, /, •: VALUE x VALUE → VALUE.

(4) a set of predicate symbols **P,** among which:
   a) a predicate symbol *holds* (|=) of type STATE x STATPROP.
   b) =: an identity relation on arbitrary sorts
   c) <: TIME x TIME is the earlier than relation on time
   d) <: VALUE x VALUE is the less than relation on the sort VALUE

When it is necessary to indicate an aspect state of a system component (i.e., input, output or internal), the sorts ASPECT_COMPONENT (a set of the component aspects of a system); COMPONENT (a set of all component names of a system); and COMPONENT_STATE_ASPECT (a set of all names of aspects of all component states) are included in the TTL. Using these sorts a functional symbol *comp_aspect* can be defined as: ASPECT_COMPONENT x COMPONENT → COMPONENT_STATE_ASPECT. Then, a function *state* is specified as: TRACE x TIME x COMPONENT_STATE_ASPECT → STATE.

Notice that also within states statements about time can be made, for this purposes the sort LTIME is used in the state language. Further we shall use u with subscripts and superscripts to denote constants of sort $LTIME^{VARS}$.

State language formulae are incorporated into the TTL by mappings of variable sets, terms sets and formulae sets into the names of sorts S^{GTERMS}, S^{TERMS}, S^{VARS} and STATPROP using the operator (*).

Definition 1.5 (Operator *)

The operator (*) is defined inductively on the structure of formulae from L_{STATE} by the following mappings:

1. Each constant symbol c∈S in ***C*** is mapped to the constant name c' of sort S^{GTERMS}.
2. Each variable x:S of the state language is mapped to the constant name x'∈ S^{VARS}
3. Each function symbol f: S_1 x S_2 x … x S_n → S_{n+1} in ℓ_{SL} is mapped to the function name f': S_1^{TERMS} x S_2^{TERMS} x … x S_n^{TERMS} → S_{n+1}^{TERMS}
4. Each predicate symbol P: S_1 x S_2 x … x S_n is mapped to the function name P': S_1^{TERMS} x S_2^{TERMS} x … x S_n^{TERMS} → STATPROP
5. The mappings for state formulae are defined as follows:
   a. (~φ)* = not(φ*)
   b. (φ & ψ)* = φ* ∧ ψ*
   c. (φ | ψ)* = φ* ∨ ψ*
   d. (φ ⇒ ψ)* = φ* → ψ*
   e. (φ ⇔ ψ)* = φ* ↔ ψ*
   f. (∀x φ(x))*= ∀x' φ*(x'), where x' is any constant of S^{VARS}

Make notice that the sorts S^{GTERMS} and S^{VARS} contain only the elements, corresponding to mappings in Definition 1.5.

Furthermore, it is assumed that the state language and the TTL signature define disjoint sets of expressions. Therefore, further in TTL formulae we will use the same notations for the elements of the object language (i.e, constants, variables, functions, predicates) and for their names in the TTL without introducing any ambiguity.

Definition 1.6 (TTL Terms)

a. Any variable x of TTL sort S is a term of L_{TTL} of sort S.
b. If f is an n-place function symbol of the TTL language $S_1 \times \ldots \times S_n \rightarrow S$ and $\tau_1,\ldots, \tau_n$ are terms of TTL sorts $S_1,\ldots., S_n$, then $f(\tau_1,\ldots, \tau_n)$ is a term of L_{TTL} of sort S.

Definition 1.7 (TTL Formulae)

TTL- formulae are defined inductively as follows:

A. The set of **atomic TTL-formulae** is defined as:

(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 (sometimes is used in infix notation like $v_1 \models u_1$).

(2) if τ_1, τ_2 are terms of any sort, then = (τ_1, τ_2) is an atomic TTL formula. (further we shall use this predicate in infix form $\tau_1 = \tau_2$)

(3) if t_1, t_2 are terms of sort TIME, then < (t_1, t_2) is an atomic TTL formula. (further we shall use this predicate in form $t_1 < t_2$, furthermore we shall use $t_1 \leq t_2$ for $t_1<t_2 \wedge t_1=t_2$)

(4) If v1, v2 are terms of sort VALUE, then v1 < v2 is an atomic TTL formula.

B. The set of **well-formed TTL-formulae** is defined as

(1) Any atomic TTL-formula is a well-formed TTL-formula

(2) If φ and ψ are well-formed TTL-formulae, then so are ~φ, (φ|ψ), (φ&ψ), (φ⇒ψ) and (φ⇔ψ).

(3) If φ is a well-formed TTL-formula containing TTL variable x of sort S, where S is one of TTL sorts, then (∀x φ) and (∃x φ) are well-formed TTL-formulae.

1.2. The Semantics of TTL

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

Definition 1.8 (Interpretation)

An **interpretation** of a TTL formula is defined by a mapping *I* that:

(1) associates each sort symbol S to a certain set (subdomain) D_S, and if $S \subseteq S'$ then $D_S \subseteq Ds'$;

(2) associates each constant c of sort S to some element of D_S

(3) associates each function symbol f of sort $<X_1, \ldots, X_i> \rightarrow X_{i+1}$ to a mapping $I(X_1) \times \ldots \times I(X_i) \rightarrow I(X_{i+1})$

(4) associates each predicate symbol P of sort $<X_1, \ldots, X_i>$ to a relation on $I(X_1) \times \ldots \times I(X_i)$

Definition 1.9 (TTL Model)

A **model M** for the language TTL is a pair M=<I, V>, where:

- *I* is an interpretation function, and

- *V* is a variable assignment function, mapping each variable x: S to an element of D_S.

We write *V* [x/v] for the assignment function that maps variables y other than x to *V*(y) and maps x to v. Analogously, we write M[x/v] = <I, V [x/v]>.

Definition 1.10 (Interpretation of TTL terms)

Let M=<I,V> be a model for TTL. Then the meaning of a term $\tau \in$ TTL, denoted by τ^M, is inductively defined by:

1. $(x)^M$=V(x), where x is a variable over one of the TTL sorts.
2. $(c)^M$= I(c), where c is a constant of one of the TTL sorts.

3. $f(\tau_1,\ldots,\tau_k)^M = I(f)(\tau_1^M,\ldots,\tau_k^M)$, where f is a TTL function of type $S_1 \times \ldots \times S_n \rightarrow S$ and $\tau_1,\ldots,\tau_n$ are terms of TTL sorts $S_1,\ldots,S_n$

Definition 1.11 (Truth definition for TTL)

Let M=<I, V> 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 $I(P)(\tau_1^M,\ldots,\tau_k^M) = true$
2. $\models_M \neg\varphi$ iff $\not\models_M \varphi$
3. $\models_M \varphi \wedge \psi$ iff $\models_M \varphi$ and iff $\models_M \psi$
4. $\models_M \forall x\,(\varphi(x))$ iff $\models_{M[x/v]} \varphi(x)$ for all $v \in D_S$, where x is a variable of sort S.

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

1.3 Axioms of TTL

(1) Equality of traces:

$\forall\gamma_1, \gamma_2\,[\forall t\,[state(\gamma_1, t) = state(\gamma_2, t)\,] \Rightarrow \gamma_1 = \gamma_2]$

(2) Equality of states:

$\forall s_1, s_2\,[\forall a{:}STATPROP\,[truth\_value(s_1, a) = truth\_value(s_2, a)] \Rightarrow s_1 = s_2]$

(3) Truth value in a state:

$holds(s, p) \Leftrightarrow truth\_value(s, p) = true$

(4) State consistency axiom:

$\forall\,\gamma, t, p\,(holds(state(\gamma, t), p) \Rightarrow \neg holds(state(\gamma, t), not(p)))$

(5) State property semantics:

a. $holds(s, (p_1 \wedge p_2)) \Leftrightarrow holds(s, p_1)\ \&\ holds(s, p_2)$
b. $holds(s, (p_1 \vee p_2)) \Leftrightarrow holds(s, p_1)\ |\ holds(s, p_2)$
c. $holds(s, not(p_1)) \Leftrightarrow \neg holds(s, p_1)$

For any constant variable name x from the sort S^{VARS}:

d. $holds(s, (\exists(x, F))) \Leftrightarrow \exists x'{:}S^{GTERMS}\ holds(s, G)$, with G, F terms of sort STATPROP, where G is obtained from F by substituting all occurrences of x by x'
e. $holds(s, (\forall(x, F))) \Leftrightarrow \forall x'{:}\ S^{GTERMS}\ holds(s, G)$, with G, F terms of sort STATPROP, where G is obtained from F by substituting all occurrences of x by x'

(6) Partial order axioms for the sort TIME:

a. $\forall t\ t \leq t$ (Reflexivity)
b. $\forall t_1,t_2\,[\,t_1 \leq t_2 \wedge t_2 \leq t_1\,] \Rightarrow t_1 = t_2$ (Anti-Symmetry)
c. $\forall t_1, t_2, t_3\,[\,t_1 \leq t_2 \wedge t_2 \leq t_3\,] \Rightarrow t_1 \leq t_3$ (Transitivity)

(7) Axioms for the sort VALUE:

a. $\forall v\ v \leq v$ (Reflexivity)
b. $\forall v_1,v_2\,[\,v_1 \leq v_2 \wedge v_2 \leq v_1\,] \Rightarrow v_1 = v_2$ (Anti-Symmetry)
c. $\forall v_1, v_2, v_3\,[\,v_1 \leq v_2 \wedge v_2 \leq v_3\,] \Rightarrow v_1 \leq v_3$ (Transitivity)
d. Standard arithmetic axioms

(8) Axioms, which relate the sorts TIME and VALUE:

a. $(t + v_1) + v_2 = t + (v_1 + v_2)$
b. $(t \bullet v_1) \bullet v_2 = t \bullet (v_1 \bullet v_2)$

(9) Finite variability property (optional):

$\forall\gamma\ \forall t\ \exists\delta{>}0\ \exists t_1, t_2\ \ t_1 \leq t \leq t_2\ \&\ t_2 - t_1 \geq \delta\ \&\ \forall t'\ t_1 \leq t' \leq t_2\ state(\gamma, t') = state(\gamma, t)$

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 $state(\gamma, t) \models p$ for some name of a 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 $state(\gamma, t) \models q(t)$.

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 $state(\gamma, t) \models t' < t''$ and $state(\gamma, t) \models t' \leq t''$ respectively. Then proceed by induction on the composition of the formula $\delta(\gamma, t)$. Treat the logical connectives &, |, $\neg$, $\Rightarrow$, $\forall$s, $\exists$s.

1) conjunction: $\delta(\gamma, t)$ is $\delta1(\gamma, t)$ & $\delta2(\gamma, t)$

By induction hypothesis

$\delta1(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1$ (which is $\delta1^*(\gamma, t)$)

$\delta2(\gamma, t) \Leftrightarrow state(\gamma, t) \models p2$ (which is $\delta2^*(\gamma, t)$)

Then

$\delta(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1 \ \& \ state(\gamma, t) \models p2 \Leftrightarrow state(\gamma, t) \models [\, p1 \wedge p2 \,]$ (which becomes $\delta^*(\gamma, t)$)

2) disjunction: $\delta(\gamma, t)$ is $\delta1(\gamma, t) \mid \delta2(\gamma, t)$

Again by induction hypothesis

$\delta1(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1$ (which is $\delta1^*(\gamma, t)$)

$\delta2(\gamma, t) \Leftrightarrow state(\gamma, t) \models p2$ (which is $\delta2^*(\gamma, t)$)

Then

$\delta(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1 \mid state(\gamma, t) \models p2 \Leftrightarrow state(\gamma, t) \models [\, p1 \vee p2 \,]$ (which becomes $\delta^*(\gamma, t)$)

3) negation: $\delta(\gamma, t)$ is $\neg\delta1(\gamma, t)$

$\delta1(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1$

$\delta(\gamma, t) \Leftrightarrow \neg state(\gamma, t) \models p1$

$\delta(\gamma, t) \Leftrightarrow state(\gamma, t) \models not(p1)$ (which is $\delta^*(\gamma, t)$)

4) implication: $\delta(\gamma, t)$ is $\delta1(\gamma, t) \Rightarrow \delta2(\gamma, t)$

Again by induction hypothesis

$\delta1(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1$ (which is $\delta1^*(\gamma, t)$)

$\delta2(\gamma, t) \Leftrightarrow state(\gamma, t) \models p2$ (which is $\delta2^*(\gamma, t)$)

Then

$\delta(\gamma, t) \Leftrightarrow [state(\gamma, t) \models p1 \Rightarrow state(\gamma, t) \models p2] \Leftrightarrow state(\gamma, t) \models [p1 \rightarrow p2]$ (which becomes $\delta^*(\gamma, t)$)

5) universal quantifier:

$\delta(\gamma, t) \Leftrightarrow \forall t'\ state(\gamma, t) \models p1(t')$

$\delta(\gamma, t) \Leftrightarrow state(\gamma, t) \models \forall u'\ p1(u')$ (which is $\delta^*(\gamma, t)$)

6) existential quantifier:

$\delta(\gamma, t) \Leftrightarrow \exists t'\ state(\gamma, t) \models p1(t')$

$\delta(\gamma, t) \Leftrightarrow state(\gamma, t) \models \exists u'\ p1(u')$ (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: $LTIME^{TERMS} \rightarrow STATPROP$ 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:

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

Correctness of time

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

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

Definition 2.2 (Memory formula)
The formula $\varphi_{mem}(\gamma, t)$ obtained by replacing all occurrences in $\varphi_p(\gamma, t)$ of subformulae of the form $state(\gamma, t') \models p$ by $state(\gamma, t) \models memory(t', p)$ is called the *memory formula for* $\varphi_p(\gamma, t)$.

Definition 2.3 (Normalized memory state formula)
The state formula constructed by Lemma 1 for a memory formula $\varphi_{mem}(\gamma, t)$ is called *the normalized memory state formula for* $\varphi_{mem}(\gamma, t)$ and denoted by $q_{mem}(t)$. Moreover, q_{mem} is the state formula $\forall u' [present\_time(u') \rightarrow q_{mem}(u')]$.

Lemma 2
If time has properties of correctness and uniqueness, then

$$\varphi_{mem}(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t) \Leftrightarrow state(\gamma, t) \models q_{mem} \quad (1)$$

Proof.

The proof for Lemma 2 follows directly from the Lemma 1, definitions of correctness and uniqueness of time and the definition of the formula q_{mem}. Lemmas 3, 4 and 5 can be proven in the same manner.

Definition 2.4 (Executable theory from interaction to memory)
For a given $\varphi(\gamma, t)$ the executable theory from observation states to memory states $Th_{o\rightarrow m}$ consists of the formulae:
For any atom p occurring in $\varphi_p(\gamma, t)$, expressed in the InteractionOnt(A) for a component A:

$\forall t'\ state(\gamma, t') \models p \Rightarrow state(\gamma, t') \models memory(t', p)$,

$\forall t''\ state(\gamma, t'') \models memory(t', p) \Rightarrow state(\gamma, t''+1) \models memory(t', p)$,

$state(\gamma, 0) \models present\_time(0)$,

$\forall t\ state(\gamma, t) \models present\_time(t) \Rightarrow state(\gamma, t+1) \models present\_time(t+1)$,

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

Proposition 1
Let $\varphi_p(\gamma, t)$ be a past statement for a given t, $\varphi_{mem}(\gamma, t)$ the memory formula for $\varphi_p(\gamma, t)$, $q_{mem}(t)$ the normalized memory state formula for $\varphi_{mem}(\gamma, t)$, and $Th_{o\rightarrow m}$ the executable theory from the interaction states for $\varphi_p(\gamma, t)$ to the memory states. Then,

$$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t)]$$

and

$$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t) \ \&\ state(\gamma, t) \models q_{mem}(t) \Leftrightarrow state(\gamma, t) \models q_{mem}].$$

Proof.

From the definitions of $q_{mem}(t)$ and of $Th_{o\rightarrow m}$ follows

$$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t)]$$

Further by Lemma 2

$$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t)] \blacksquare$$

Definition 2.5 (Normalized condition state formula)
The state formula constructed by Lemma 1 for $\varphi_{cmem}(\gamma, t, t_1)$ is called *the normalized condition state formula for* $\varphi_{cmem}(\gamma, t, t_1)$ and denoted by $q_{cond}(t, t_1)$. Moreover, $q_{cond}(t)$ is the state formula $\forall u' [present\_time(u') \rightarrow q_{cond}(t, u')]$

Lemma 3
If time has properties of correctness and uniqueness, then

$$\varphi_{cmem}(\gamma, t, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{cond}(t, t_1) \ \&\ state(\gamma, t_1) \models q_{cond}(t, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{cond}(t) \quad (2)$$

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 $\varphi_{prep}(\gamma, t_1)$ is called *the normalized preparation state formula for* $\varphi_{prep}(\gamma, t_1)$ and denoted by $q_{prep}(t_1)$. Moreover, q_{prep} is the state formula $\forall u' [present\_time(u')] \rightarrow q_{prep}(u')]$

Lemma 4
If time has properties of correctness and uniqueness, then

$$\varphi_{prep}(\gamma, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{prep}(t_1) \ \&\ state(\gamma, t_1) \models q_{prep}(t_1) \Leftrightarrow state(\gamma, t_1) \models q_{prep} \quad (3)$$

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 $q_{cond}(t, t_1)$ be the normalized condition state formula for $\varphi_{cmem}(\gamma, t, t1)$ and $q_{prep}(t_1)$ the normalized preparation state formula for $\varphi_{prep}(\gamma, t_1)$. The formula $\varphi_{cprep}(\gamma, t)$ of the form $state(\gamma, t) \models \forall u_1 > t\ [q_{cond}(t, u_1) \rightarrow q_{prep}(u_1)]$ is called the *conditional preparation formula* for $\varphi_f(\gamma, t)$.

The state formula $\forall u_1 > t\ [\ q_{cond}(t, u_1) \rightarrow q_{prep}(u_1)\ ]$ is called *the normalized conditional preparation state formula for* $\varphi_{cprep}(\gamma, t)$ and denoted by $q_{cprep}(t)$. Moreover, q_{cprep} is the formula $\forall u'\ [\ present\_time(u') \rightarrow q_{cprep}(u')\ ]$.

Lemma 5
If time has properties of correctness and uniqueness, then

$$\varphi_{cprep}(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t)\ \&\ state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep} \qquad (4)$$

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 $\varphi_{cond}(\gamma, t, t_1)$, expressed in the InteractionOnt(A) for component A[1]:

$\forall t'\ state(\gamma, t') \models p \Rightarrow state(\gamma, t') \models [\ memory(t', p) \wedge stimulus\_reaction(p)\ ]$

$\forall t'', t'\ state(\gamma, t'') \models memory(t', p) \Rightarrow state(\gamma, t''+1) \models memory(t', p)$

$\forall t'\ state(\gamma, t') \models q_{mem} \Rightarrow state(\gamma, t') \models q_{cprep}$,

$\forall t', t\ state(\gamma, t') \models [q_{cprep} \wedge q_{cond}(t) \wedge \bigcap_{p} stimulus\_reaction(p)\ ] \Rightarrow state(\gamma, t') \models q_{prep}$,

$\forall t'\ state(\gamma, t') \models [\ stimulus\_reaction(p) \wedge \neg\ preparation\_for(output(t'+c, a))\ ] \Rightarrow state(\gamma, t'+1) \models stimulus\_reaction(p)$,

$\forall t'\ state(\gamma, t') \models [\ preparation\_for(output(t'+c, a)) \wedge \neg\ output(a)\ ] \Rightarrow state(\gamma, t'+c) \models preparation\_for(output(t'+c, a))$,

where a is an action or a communication for which $state(\gamma, t'+c) \models output(a)$ occurs in $\varphi_f(\gamma, t)$.

Proposition 2
Let $\varphi_f(\gamma, t)$ be a future statement for t of the form $\forall t_1 > t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$, where $\varphi_{cond}(\gamma, t, t_1)$ is an interval statement, which describes a condition for one or more actions and/or communications and $\varphi_{bh}(\gamma, t_1)$ is a (conjunction of) future statement(s) for t_1, which describes action(s) and/or communications that are to be performed; let $\varphi_{cprep}(\gamma, t)$ be the conditional preparation formula for $\varphi_f(\gamma, t)$, $q_{cprep}(t)$ be the normalized conditional preparation state formula for $\varphi_{cprep}(\gamma, t)$, and $Th_{m \rightarrow p}$ the executable theory for $\varphi(\gamma, t)$ from memory states to preparation states. Then,

$$Th_{m \rightarrow p} \models [\varphi_f(\gamma, t) \Leftrightarrow \varphi_{cprep}(\gamma, t)]$$

and

$$Th_{m \rightarrow p} \models [\ \varphi_f(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t)\ \&\ state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}].$$

Proof.

From the definition of $Th_{m \rightarrow p}$, Lemmas 3 and 4 follows that

$$Th_{m \rightarrow p} \models [\ \varphi_{cond}(\gamma, t, t_1) \Leftrightarrow q_{cond}(t, t_1)\ ] \qquad (5)$$

and

$$Th_{m \rightarrow p} \models [\ \varphi_{bh}(\gamma, t_1) \Leftrightarrow q_{prep}(\gamma, t_1)] \qquad (6)$$

From (5), (6), definitions of the conditional preparation formula and the normalized conditional preparation state formulae, and the conditions of the proposition it follows

$$Th_{m \rightarrow p} \models [\ \varphi_f(\gamma, t) \Leftrightarrow \forall u_1 > t\ [q_{cond}(t, u_1) \rightarrow q_{prep}(\gamma, u_1)]\ \&\ \forall u_1 > t\ [q_{cond}(t, u_1) \rightarrow q_{prep}(\gamma, u_1)] \Leftrightarrow \forall t_1 > t\ \varphi_{cprep}(\gamma, t, t_1) \Leftrightarrow \varphi_{cprep}(\gamma, t)\ ]$$

And from Lemma 5 follows

$$Th_{m \rightarrow p} \models [\ \varphi_f(\gamma, t) \Leftrightarrow \forall t_1 > t\ state(\gamma, t) \models q_{cprep}(t, t_1) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t) \ \&\ state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}\ ]\ \blacksquare$$

Proposition 3

Let $\varphi_p(\gamma, t)$ be a past statement for t and $\varphi_f(\gamma, t)$ be a future statement for t. Let $\varphi_{mem}(\gamma, t)$ be the memory formula for $\varphi_p(\gamma, t)$ and $\varphi_{cprep}(\gamma, t)$ the conditional preparation formula for $\varphi_f(\gamma, t)$. Then

$$[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)] \Leftrightarrow [\varphi_{mem}(\gamma, t) \Rightarrow \varphi_{cprep}(\gamma, t)]$$

Proof.

[1] If a future formula does not contain a condition, then stimulus_reaction atoms are generated from the corresponding past formula

From the Proposition 1 and the Proposition 2 follows

$$\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t) \text{ and } \varphi_f(\gamma, t) \Leftrightarrow \forall t_1 > t\ \varphi_{cprep}(\gamma, t, t_1)$$

Then,

$$[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)] \Leftrightarrow [\varphi_{mem}(\gamma, t) \Rightarrow \forall t_1 > t\ \varphi_{cprep}(\gamma, t, t_1)]$$

So it has been proven that $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)] \Leftrightarrow [\varphi_{mem}(\gamma, t) \Rightarrow \varphi_{cprep}(\gamma, t)]$ ■

Definition 2.9 (Executable theory from preparation to output)
For a given $\varphi_f(\gamma, t)$ the executable theory from the preparation to the output state(s) $Th_{p\to o}$ consists of the formula

$\forall t'$ state(γ, t') |= preparation_for(output(t'+c, a)) $\Rightarrow$ state(γ, t'+c) |= output(a),

where c is a number and a an action or a communication for which state(γ, t'+c) |= output(a) occurs in $\varphi_f(\gamma, t)$.

Definition 2.10 (Executable specification)
An *executable specification* $\pi(\gamma, t)$ for the component A is defined by a union of the dynamic properties from the executable theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to o}$.

Definition 2.11 (Coinciding traces)
Two traces γ_1, γ_2 *coincide* on ontology Ont (denoted by a predicate symbol coincide_on: TRACE x TRACE x ONTOLOGY (ONTOLOGY is a sort that contains all names of ontologies)) iff

$\forall t\ \forall a \in STATATOM_{Ont}$ state(γ_1, t) |= a $\Leftrightarrow$ state(γ_2, t) |= a,

where $STATATOM_{Ont} \subseteq STATPROP_{Ont}$ is the sort, which contains all names 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 component A. An executable specification $\pi(\gamma, t)$ for A *refines* $\varphi(\gamma, t)$ iff

(1) $\forall \gamma, t\ \ \pi(\gamma, t) \Rightarrow \varphi(\gamma, t)$

(2) $\forall \gamma_1, t\ [\ \varphi(\gamma_1, t) \Rightarrow [\ \exists \gamma_2$ coincide_on(γ_1, γ_2, InteractionOnt(A)) & $\pi(\gamma_2, t)$]]

Note that for any past interaction statement $\varphi_p(\gamma, t)$ and future interaction statement $\varphi_f(\gamma, t)$ the following holds:

$\forall \gamma 1, \gamma 2$ [coincide_on($\gamma 1, \gamma 2$, InteractionOnt) $\Rightarrow$ $[\varphi_p(\gamma 1, t) \Leftrightarrow \varphi_p(\gamma 2, t)$ & $\varphi_f(\gamma 1, t) \Leftrightarrow \varphi_f(\gamma 2, t)$]]

Lemma 6

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

(1) coincide_on(γ_1, γ_2, Ont) & coincide_on(γ_2, γ_3, Ont) $\Rightarrow$ coincide_on(γ_1, γ_3, Ont)

(2) coincide_on(γ_1, γ_2, Ont) $\Rightarrow$ [$\varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)$].

Proof sketch.

The transitivity property (1) follows directly from the definition of coinciding traces for coincide_on(γ_1, γ_2, Ont) and coincide_on(γ_2, γ_3, Ont):

$\forall a \in STATATOM_{Ont}\ \ \forall t'$ [state(γ_1, t') |= a $\Leftrightarrow$ state(γ_3, t') |= a] $\Rightarrow$ coincide_on(γ_1, γ_3, Ont)

From

$\forall\ \gamma_1, \gamma_2$ [coincide_on(γ_1, γ_2, Ont) $\Rightarrow$ $\forall t'$ $[\varphi_p(\gamma_1, t') \Leftrightarrow \varphi_p(\gamma_2, t')$ & $\varphi_f(\gamma_1, t') \Leftrightarrow \varphi_f(\gamma_2, t')$]]

follows that $\varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)$.

Theorem

If the executable specification $\pi_A(\gamma, t)$ refines the external behavioral specification $\varphi_A(\gamma, t)$ of component A, and $\psi(\gamma, t)$ is a dynamic interaction property of component A in its environment, expressed using the interaction ontology InteractionOnt(A), then

$$[\forall \gamma\ [\ \pi_A(\gamma, t) \Rightarrow \psi(\gamma, t)\ ]\ ] \Leftrightarrow [\forall \gamma\ [\ \varphi_A(\gamma, t) \Rightarrow \psi(\gamma, t)\ ]\ ]$$

Proof sketch for Theorem.

$\Leftarrow$ is direct:

from $\pi_i(\gamma, t) \Rightarrow \varphi_i(\gamma, t)$ and $\wedge\varphi_i(\gamma, t) \Rightarrow \psi(\gamma, t)$ it follows $\wedge\pi_i(\gamma, t) \Rightarrow \psi(\gamma, t)$.

$\Rightarrow$ runs as follows:

Suppose $\varphi_i(\gamma, t)$ holds for all i, then since $\pi_1(\gamma)$ refines $\varphi_1(\gamma, t)$, then according to the definition of refinement of an externally observable property exists such a γ_1 that $\pi_1(\gamma_1)$ and coincide_on(γ, γ_1, InteractionOnt (A)).

Due to Lemma 6, this γ_1 still satisfies all $\varphi_i(\gamma_1, t)$ (i.e., $\varphi_i(\gamma_1, t)$ holds for all i).

Proceed with γ_1 to obtain a γ_2 and further for all i to reach a trace γ_n, for which

$\pi_i(\gamma_n)$ holds for all i,

and

coincide_on(γ, γ_n, InteractionOnt(A)),

and

$\varphi_i(\gamma_n)$ holds for all i.

From

$$\forall\gamma\ \forall i\ [\pi_i(\gamma) \Rightarrow \varphi_i(\gamma)],$$

and

$$\forall\gamma\ [\wedge\pi_i(\gamma) \Rightarrow \psi(\gamma, t)]$$

it follows that $\forall\gamma\ \wedge\varphi_i(\gamma) \Rightarrow \psi(\gamma)$.

So it has been proven that $\forall\gamma\ \wedge\varphi_i(\gamma) \Rightarrow \psi(\gamma)$. ■

3. TRANSFORMATION INTO THE FINITE STATE TRANSITION SYSTEM FORMAT

According to the Definition 2.10 the executable specification of a component's behavior consists of the union of three theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to o}$, 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) $\wedge\ \neg$p $\rightarrow\!\!\!\!\rightarrow$ present_time(1)

present_time(t) $\wedge\ \neg q_{mem} \wedge\ \neg$p $\rightarrow\!\!\!\!\rightarrow$ present_time(t+1)

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

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

Memory state creation rule:

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

present_time(t) $\wedge$ p $\rightarrow$ [memory(t, p) $\wedge$ stimulus_reaction(p)]

For all other state atoms p

present_time(t) $\wedge$ p $\rightarrow\!\!\!\!\rightarrow$ memory(t, p)

Memory persistence rule:

memory(t, p) $\rightarrow\!\!\!\!\rightarrow$ memory(t, p)

Conditional preparation generation rule:

$q_{mem} \rightarrow\!\!\!\!\rightarrow$ conditional_preparation_for(output(a)),

where a an action or a communication for which state(γ, t'+c) |= output(a) occurs in $\varphi_f(\gamma, t)$.

Preparation state creation rule:

present_time(t') $\wedge$ conditional_preparation_for(output(a)) $\wedge\ q_{cond}$(t) $\wedge\ \bigcap_{p}$ stimulus_reaction(p) $\rightarrow\!\!\!\!\rightarrow$ preparation_for(output(t'+c, a))

for every subformula of the form

present_time(t') $\rightarrow$ preparation_for(output(t'+c, a))

that occurs in q_{cprep}.

Preparation state persistence rule:

preparation_for(output(t+c, a)) $\wedge\ \neg$output(a) $\rightarrow\!\!\!\!\rightarrow$ preparation_for(output(t+c, a))

Stimulus reaction state persistence rule:

present_time(t') $\wedge$ stimulus_reaction(p) $\wedge \neg$ preparation_for(output(t'+c, a)) $\Rightarrow$ stimulus_reaction(p)

Output state creation rule:

preparation_for(output(t+c, a)) $\wedge$ present_time(t+c-1) $\Rightarrow$ output(a), where a is an action or a communication.

4. TRANSFORMATION INTO THE SMV MODEL SPECIFICATION FORMAT

For automatic verification of relationships between dynamic properties of components of different aggregation levels by means of model checking techniques, a corresponding to the behavioral specification of the lower aggregation level 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.

Let us describe the transformation procedure, which is automatically performed by the dedicated software that has been developed.

First, using the standard rules [1] $q_{mem}(t)$ and $q_{cond}(t, t_1)$ 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 $q_{mem}(t)$ and then to $q_{cond}(t, t_1)$. After that conditional preparation generation rules are added by performing the step 4. Finally, the preparation and output state creation rules are generated for each dynamic property by performing the step 5.

Step 1. Rewrite in the external behavioral specification all occurrences of the function memory(communicated(t1, a)) by memory(observed(t1, a)) and of the function $\neg$memory(communicated(t1, a)) by $\neg$memory(observed(t1, a)) for some given atom a. 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 function of the form memory(observed(t1, obs_event)), $\neg$memory(observed(t1, obs_event)), memory(output(t1, act_event)), and $\neg$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 $\forall$t1 P(t1), create an atom (a label) t1 and add to the specification the following:

For memory(observed(t1, obs_event)), $\neg$memory(observed(t1, obs_event)):

```
t1: boolean ;
init(t1):=0;
obs_event: boolean;
init(obs_event):=0;
```

For memory(output(t1, obs_event)) and $\neg$memory(output(t1, obs_event)):

```
t1: boolean ;
init(t1):=0;
act_event: boolean;
init(act_event):=0;
```

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 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 rules:

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

(b) For each occurrence of the expression Q t1, t2 R t1 memory(output(t1, act_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 rules:

```
next(t1):= case
             t2 & act_event: 1; //memory state creation
             !t2: 0;
```

```
            1: t1;              //persistence of memory
  esac;
```

(c) For each occurrence of the expression Q t1, t2 R t1 ¬memory(observed(t1, obs_event)), add to the specification the following rules:

```
next(t1):= case
           t2 & !obs_event: 1;
           !t2: 0;
           1: t1;
esac;
```

(d) For each occurrence of the expression Q t1, t2 R t1 ¬memory(output(t1, act_event)), add to the specification the following rules:

```
next(t1):= case
           t2 & !act_event: 1;
           !t2: 0;
           1: t1;
esac;
```

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)]:

(a) if P3(t) is of the form memory(observed(t3, obs_event))

i. 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;
```

ii. 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
            !t2 & 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 t3 ≤ t2 and t1< t3 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;
```

iiii. 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;
```

(b) if P3(t) is of the form memory(output(t3, act_event))

i. 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
            !act_event2 & !t2 & t3t1_eq & !act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !act_event2 & !t2 & !act_event3: 0;
           !act_event2 & !t2 & act_event3: 1;
           1: t3;
esac;
```

ii. 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
            !t2 & t3t1_eq & !act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & !act_event3: 0;
           !t2 & act_event3: 1;
           1: t3;
esac;
```

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

```
next(t1):= case
            !act_event2 & !t2 & !act_event3: 0;
            1: t1;
```

```
esac;
next(t3):= case
           !t1: 0;
           !act_event2 & !t2 & !act_event3: 0;
           !act_event2 & !t2 & act_event3: 1;
           1: t3;
esac;
```

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

```
next(t1):= case
            !t2 & !act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & !act_event3: 0;
           !t2 & act_event3: 1;
           1: t3;
esac;
```

(c) If $P3(t)$ is of the form ¬memory(observed(t3, obs_event))

i. For t3 < t2 and t1< t3 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 t3 < t2 and t1≤ t3 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 t3 ≤ t2 and t1< t3 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;
```

iiii. 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;
```

(d) If $P3(t)$ is of the form ¬memory(output(t3, act_event))

i. For t3 < t2 and t1< t3 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
         !act_event2 & !t2 & neg_t3t1_eq & act_event3: 0;
         1: t1;
esac;
next(t3):= case
           !t1: 0;
           !act_event2 & !t2 & act_event3: 0;
           !act_event2 & !t2 & !act_event3: 1;
           1: t3;
esac;
```

ii. For t3 < t2 and t1≤ t3 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 & act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & act_event3: 0;
           !t2 &!act_event3: 1;
           1: t3;
esac;
```

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

```
next(t1):= case
            !act_event2 & !t2 & act_event3: 0;
            1: t1;
```

```
esac;
next(t3):= case
           !t1: 0;
           !act_event2 & !t2 & act_event3: 0;
           !act_event2 & !t2 & !act_event3: 1;
           1: t3;
esac;
```

iiii. For $t3 \leq t2$ and $t1 \leq t3$ add to the specification the following rules:

```
next(t1):= case
            !t2 & act_event3: 0;
            1: t1;
esac;
next(t3):= case
           !t1: 0;
           !t2 & act_event3: 0;
           !t2 &!act_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 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; //conjunction of all labels, created based on φcond(γ, t, t1)
                    j
                     1: 0;
esac;

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

5. Case study

To illustrate the proposed approach to verify interlevel relations, a multi-agent system for co-operative information gathering is considered at two aggregation levels. At the higher level the multi-agent system as a whole is considered. At the lower level the four components and their interactions are considered: two information gathering agents A and B, agent C, and environment component E representing the conceptualized part of the external world. Each of the agents is able to acquire partial information from an external source (component E) by initiated observations. Each agent can be reactive or proactive with respect to the information acquisition process. An agent is proactive if it is able to start information acquisition independently of requests of any other agents, and an agent is reactive if it requires a request from some other agent to perform information acquisition.

Observations of any agent taken separately are insufficient to draw conclusions of a desired type, but the combined information of both agents is sufficient. Therefore, the agents need to co-operate to be able to draw conclusions. Each agent can be proactive with respect to the conclusion generation, i.e., after receiving both observation results an agent is capable to generate and communicate a conclusion to agent C. Moreover, an agent can be request pro-active to ask information from another agent, and an agent can be pro-active or reactive in provision of (already acquired) information to the other agent.

For the lower-level components of this example multi-agent system, a number of dynamic properties were identified and formalized. The variables A1 and A2 are defined over the sort $\text{AGENT}^{\text{TERMS}}$, the constant E belongs to the sort $\text{ENVIRONMENTAL\_COMPONENT}^{\text{GTERMS}}$, the variable IC is defined over the sort $\text{INFORMATION\_CHUNK}^{\text{TERMS}}$, the constants IC1, IC2 and IC3 belong to the sort $\text{INFORMATION\_CHUNK}^{\text{GTERMS}}$ and the constant C belongs to the sort $\text{AGENT}^{\text{TERMS}}$.

Notice that some properties in the behavioral specification below (e.g., DP1, DP2) are already specified in executable format and expressed using ontologies different than interaction ontologies of agents, which these properties concern. Since there is no need to translate such properties into executable format, they will be added to the executable specification of an agent behavior at the step 5 of the translation procedure.

DP1(A1, A2) (Effectiveness of information request transfer between agents)

∀IC ∀t1<t [state(γ, t1, output(A1)) |= output(communicated(request_from_to_for(A1, A2, IC)))]
⇒ ∃t2>t [state(γ, t2, input(A2)) |= communicated(request_from_to_for(A1, A2 , IC))]

DP2(A1, A2) (Effectiveness of information transfer between agents)

∀IC ∀t1<t [state(γ, t1, output(A1)) |= output(communicated(send_from_to(A1, A2, IC)))]
⇒ ∃t3>t [state(γ, t3, input(A2)) |= communicated(send_from_to(A1, A2, IC))]

DP3(A1, E) (Effectiveness of information transfer between an agent and environment)

∀IC ∀t1<t [state(γ, t1, output(A1)) |= output(obs_focus_from_to_for(A1, E, IC))]
⇒ ∃t3>t [state(γ, t3, input(E)) |= observed(obs_focus_from_to_for(A1, E, IC))]

DP4(A1, E) (Information provision effectiveness)
∀IC ∀t1<t [state(γ, t1, input(E)) |= observed(obs_focus_from_to_for(A1, E, IC))]
⇒ ∃t3>t [state(γ, t3, output(E)) |= observed(provide_result_from_to(E, A1, IC))]

DP5(E, A1) (Effectiveness of information transfer between environment and an agent)

∀IC ∀t1<t [state(γ, t1, output(E)) |= observed(provide_result_from_to(E, A1, IC))]
⇒ ∃t3>t [state(γ, t3, input(A1)) |= observed(provided_result_from_to(E, A1, IC))]

DP6(A1, A2) (Information acquisition reactiveness)
∀IC ∀t1<t [state(γ, t1, input(A2)) |= communicated(request_from_to_for(A1, A2, IC))]
⇒ ∃t3>t [state(γ, t3, output(A2)) |= output(obs_focus_from_to_for(A2, E, IC))]

DP7(A1, A2) (Information provision reactiveness)
∀IC ∀t1 [∃t2 [t1<t & t2<t1 & state(γ, t1, input(A2)) |= observed(provided_result_from_to(E, A2, IC) & state(γ, t2, input(A2)) |= communicated(request_from_to_for(A1, A2, IC))]]
⇒ ∃t4>t [state(γ, t4, output(A2)) |= output(communicated(send_from_to(A2, A1, IC)))]

DP8(A1, A2) (Conclusion proactiveness)
∃t1, t2 t1<t & t2<t & state(γ, t1, input(A1)) |= observed(provided_result_from_to(E, A1, IC1)) & state(γ, t2, input(A1)) |= communicated(send_from_to(A2, A1, IC2))
⇒ ∃t4>t [state(γ, t4, output(A1)) |=output(communicated(send_from_to(A1, C, IC3)))]

DP9(A1, E) (Information acquisition proactiveness)
∃t ∃IC state(γ, t, output(A1)) |= output(obs_focus_from_to_for(A1, E, IC))

DP10(A1, A2) (Information request proactiveness)
∃t ∃IC state(γ, t, output(A1)) |= output(communicated(request_from_to_for(A1, A2, IC)))

The dynamic properties of the higher aggregation level component (the whole multi-agent system considered as one component) can be formulated and formalized in the following way.

GP1 (Information acquisition initiation effectiveness): At some points in time A and B will start information acquisition to E.

∃t1, t2 [state(γ, t1, output(A)) |= output(obs_focus_from_to_for(A, E, IC1)) & state(γ, t2, output(B)) |= output(obs_focus_from_to_for(B, E, IC2))]

GP2(A1) (Information source effectiveness for agent A): If at some point in time A starts information acquisition to E, then E will generate all the correct relevant information for agent A.

∀t1<t ∀IC [state(γ, t1, output(A)) |= output(obs_focus_from_to_for(A, E, IC))]
⇒ ∃t2>t [state(γ, t2, output(E)) |=observed(provide_result_from_to(E, A1, IC))]

GP3 (Concluding effectiveness): If at some point in time E generates all the correct relevant information, then C will receive a correct conclusion.

∀t1, t2 [t1<t & t2<t [state(γ, t1, output(E)) |= observed(provide_result_from_to(E, A, IC1)) & state(γ, t2, output(E)) |= observed(provide_result_from_to(E, B, IC2))]]
⇒ ∃t3>t [state(γ, t3, input(C)) |= communicated(send_from_to(A, C, IC3))

Between the lower level and the higher level properties the following interlevel relations were manually identified:

DP9 & DP10 & DP1(A, B) & DP6(A, B) ⇒ GP1 (7)

DP3 (A, E) & DP4(A, E) ⇒ GP2(A) (8)

DP3 (B, E) & DP4(B, E) ⇒ GP2(B) (9)

DP5 (E, A) & DP5 (E, B) & DP10 & DP1(A, B) & DP7(A, B) & DP2(B, A) & DP8(A, B) & DP2(A, C) ⇒ GP3 (10)

From the higher level properties GP1, GP2(A), GP2(B) and GP3 the global system successfulness property can be inferred.

GP (Successfulness): For any trace there exists a point in time when agent C will receive a correct conclusion.

∃t ∃IC state(γ, t, input(C)) |= communicated(send_from_to(A, C, IC))

GP1 & GP2(A) & GP2(B) & GP3 ⇒ GP

Now, by means of the proposed approach the identified interlevel relations between dynamic properties of different aggregation levels (7), (8), (9) and (10) can be formally justified (or refuted). For this purpose first for every relationship using the developed software the external behavioral specification of the multi-agent system that consists of the lower level properties of the antecedent of the relationship is automatically transformed into an executable behavioral specification. Then, using the other automated translation procedure, every executable behavioral specification is converted into a description of a finite state transition system. In order to perform verification by means of SMV model checker, every general description of the finite state transition system has been automatically translated into the SMV model specification format. Using the state transition system representation, verification of the entailment relations, represented as CTL formulas, can be performed. For example, for the relation (10) (for the complete description of the finite state transition system we refer to the Appendix A the dynamic property GP3 is expressed in CTL as:

AG (E_output_observed_provide_result_from_to_E_A_info & E_output_observed_provide_result_from_to_E_B_info
→ AF input_C_communicated_send_from_to_A_C_info)

The automatic verification showed that all the relationships between properties (7), (8), (9) and (10) indeed hold with respect to the corresponding models of the multi-agent system.

At the same time, if one excludes property DP8 from the antecedent of the relation (10), model checking proves that the formulated in such way relationship fails. From the counter-example produced by the model checker, it is visible that although agent A has all necessary information to draw a conclusion, it will never send the conclusion to agent C. Thus, by performing such verification, it is possible to reveal mistakes in manually identified relations between properties and improve them.

REFERENCES

[1] Fitting, M. First-order Logic and Automated Theorem Proving. 2nd edition, Springer-Verlag, 1996.

Appendix A: SMV model description for the example of the co-operative information gathering MAS

```
MODULE main
 VAR
fpast1: boolean;
t1: boolean;
E_output_observed_provide_result_from_to_E_A_info: boolean;
t2: boolean;
A_input_provided_result_from_to_E_A_info: boolean;
fprep_A_input_provided_result_from_to_E_A_info: boolean;
fpast2: boolean;
t4: boolean;
E_output_observed_provide_result_from_to_E_B_info: boolean;
t5: boolean;
B_input_provided_result_from_to_E_B_info: boolean;
fprep_B_input_provided_result_from_to_E_B_info: boolean;
fpast3: boolean;
t7: boolean;
A_output_request_from_to_for_A_B_info: boolean;
t8: boolean;
B_input_request_from_to_for_A_B_info: boolean;
fprep_B_input_request_from_to_for_A_B_info: boolean;
fpast4: boolean;
t10: boolean;
t11: boolean;
t12: boolean;
B_output_send_from_to_B_A_info: boolean;
fprep_B_output_send_from_to_B_A_info: boolean;
fpast5: boolean;
t14: boolean;
t15: boolean;
A_input_send_from_to_B_A_info: boolean;
fprep_A_input_send_from_to_B_A_info: boolean;
fpast6: boolean;
t17: boolean;
A_output_communicated_send_from_to_A_C_info: boolean;
t18: boolean;
C_input_communicated_send_from_to_A_C_info: boolean;
fprep_C_input_communicated_send_from_to_A_C_info: boolean;
fpast7: boolean;
t20: boolean;
t21: boolean;
t22: boolean;
fprep_A_output_communicated_send_from_to_A_C_info: boolean;
t24: boolean;
ASSIGN
init(fpast1):=0;
init(t1):=0;
init(E_output_observed_provide_result_from_to_E_A_info):=0;
init(t2):=0;
init(A_input_provided_result_from_to_E_A_info):=0;
init(fprep_A_input_provided_result_from_to_E_A_info):=0;
init(fpast2):=0;
init(t4):=0;
init(E_output_observed_provide_result_from_to_E_B_info):=0;
init(t5):=0;
init(B_input_provided_result_from_to_E_B_info):=0;
init(fprep_B_input_provided_result_from_to_E_B_info):=0;
init(fpast3):=0;
```

```
init(t7):=0;
init(t8):=0;
init(B_input_request_from_to_for_A_B_info):=0;
init(fprep_B_input_request_from_to_for_A_B_info):=0;
init(fpast4):=0;
init(t10):=0;
init(t11):=0;
init(t12):=0;
init(B_output_send_from_to_B_A_info):=0;
init(fprep_B_output_send_from_to_B_A_info):=0;
init(fpast5):=0;
init(t14):=0;
init(t15):=0;
init(A_input_send_from_to_B_A_info):=0;
init(fprep_A_input_send_from_to_B_A_info):=0;
init(fpast6):=0;
init(t17):=0;
init(A_output_communicated_send_from_to_A_C_info):=0;
init(t18):=0;
init(C_input_communicated_send_from_to_A_C_info):=0;
init(fprep_C_input_communicated_send_from_to_A_C_info):=0;
init(fpast7):=0;
init(t20):=0;
init(t21):=0;
init(t22):=0;
init(fprep_A_output_communicated_send_from_to_A_C_info):=0;
init(t24):=0;
init(A_output_request_from_to_for_A_B_info):=1;
next(t2):= case
 !A_input_provided_result_from_to_E_A_info: 1;
1:t2;
esac;
next(t1):= case
 E_output_observed_provide_result_from_to_E_A_info: 1;
1:t1;
esac;
next(t4):= case
 E_output_observed_provide_result_from_to_E_B_info: 1;
1:t4;
esac;
next(t5):= case
 !B_input_provided_result_from_to_E_B_info: 1;
1:t5;
esac;
next(t7):= case
 t24: 1;
!t24:0;
1:t7;
esac;
next(t8):= case
 !B_input_request_from_to_for_A_B_info: 1;
B_input_request_from_to_for_A_B_info:0;
1:t8;
esac;
next(t11):= case
 B_input_request_from_to_for_A_B_info: 1;
!B_input_request_from_to_for_A_B_info:0;
1:t11;
esac;
next(t12):= case
 !B_output_send_from_to_B_A_info: 1;
B_output_send_from_to_B_A_info:0;
1:t12;
esac;
```

```
next(t10):= case
 !t11: 0;
B_input_provided_result_from_to_E_B_info & t11: 1;
!t11: 0;
!B_input_provided_result_from_to_E_B_info & t11:0;
1:t10;
esac;
next(t14):= case
 B_output_send_from_to_B_A_info: 1;
!B_output_send_from_to_B_A_info:0;
1:t14;
esac;
next(t15):= case
 !A_input_send_from_to_B_A_info: 1;
A_input_send_from_to_B_A_info:0;
1:t15;
esac;
next(t17):= case
 A_output_communicated_send_from_to_A_C_info: 1;
!A_output_communicated_send_from_to_A_C_info:0;
1:t17;
esac;
next(t18):= case
 !C_input_communicated_send_from_to_A_C_info: 1;
C_input_communicated_send_from_to_A_C_info:0;
1:t18;
esac;
next(t22):= case
 !A_output_communicated_send_from_to_A_C_info: 1;
1:t22;
esac;
next(t21):= case
 A_input_send_from_to_B_A_info: 1;
1:t21;
esac;
next(t20):= case
 A_input_provided_result_from_to_E_A_info: 1;
1:t20;
esac;
next(t24):= case
 A_output_request_from_to_for_A_B_info: 1;
1:t24;
esac;
next(fpast1):= case
t2 & t1: 1;
1:0;
 esac;
next(fprep_A_input_provided_result_from_to_E_A_info):= case
 fpast1: 1;
1:0;
 esac;
next(A_input_provided_result_from_to_E_A_info):= case
 fprep_A_input_provided_result_from_to_E_A_info: 1;
1:0;
 esac;
next(fpast2):= case
t4 & t5: 1;
1:0;
 esac;
next(fprep_B_input_provided_result_from_to_E_B_info):= case
 fpast2: 1;
1:0;
 esac;
next(B_input_provided_result_from_to_E_B_info):= case
```

```
 fprep_B_input_provided_result_from_to_E_B_info: 1;
1:0;
 esac;
next(fpast3):= case
t7 & t8: 1;
1:0;
 esac;
next(fprep_B_input_request_from_to_for_A_B_info):= case
 fpast3: 1;
1:0;
 esac;
next(B_input_request_from_to_for_A_B_info):= case
 fprep_B_input_request_from_to_for_A_B_info: 1;
1:0;
 esac;
next(fpast4):= case
t11 & t12 & t10: 1;
1:0;
 esac;
next(fprep_B_output_send_from_to_B_A_info):= case
 fpast4: 1;
1:0;
 esac;
next(B_output_send_from_to_B_A_info):= case
 fprep_B_output_send_from_to_B_A_info: 1;
1:0;
 esac;
next(fpast5):= case
t14 & t15: 1;
1:0;
 esac;
next(fprep_A_input_send_from_to_B_A_info):= case
 fpast5: 1;
1:0;
 esac;
next(A_input_send_from_to_B_A_info):= case
 fprep_A_input_send_from_to_B_A_info: 1;
1:0;
 esac;
next(fpast6):= case
t17 & t18: 1;
1:0;
 esac;
next(fprep_C_input_communicated_send_from_to_A_C_info):= case
 fpast6: 1;
1:0;
 esac;
next(C_input_communicated_send_from_to_A_C_info):= case
 fprep_C_input_communicated_send_from_to_A_C_info: 1;
1:0;
 esac;
next(fpast7):= case
t22 & t21 & t20: 1;
1:0;
 esac;
next(fprep_A_output_communicated_send_from_to_A_C_info):= case
 fpast7: 1;
1:0;
 esac;
next(A_output_communicated_send_from_to_A_C_info):= case
 fprep_A_output_communicated_send_from_to_A_C_info: 1;
1:0;
 esac;
```