Specification and Dynamic Verification of Agent Properties

Stefania Costantini    Pierangelo Dell’Acqua    Luís Moniz Pereira
Panagiota Tsintza

Univ. of L’Aquila, Italy & Linköping Univers., Sweden & Univers. Nova de Lisboa, Portugal & Univ. of L’Aquila, Italy

Agent Verification
A priori or dynamic?

As agent systems are more widely used in real-world applications, the issue of verification is becoming increasingly important. Which approach?

- A priori verification: Model Checking & Theorem Proving.
- Dynamic verification: properties verified at runtime.

The two approaches are complementary rather than in competition.
Verification of Properties in Agents
A General Example

In agents, how to verify, e.g., that property $\xi = N\varphi$ holds?
(i.e., it is required to verify that $\varphi$ will never happen to be true)

1. One may try to prove that $\xi$ cannot possibly hold in any future state;

2. one can try to verify this by explicitly examining all possible future states of given agent;

3. one may perform a run-time verification with a certain frequency, where suitable counter-measures will be undertaken in case of violation.
Verification of Properties in Agents
A More Specific Example

Suppose that, for agent $a$, one wishes to verify $\xi = N \text{ received}(b, msg)$ (message with content $msg$ coming from sender agent $b$ will never be accepted by $a$). How to do that?

1. One could prove, e.g., that by the definition of $a$ and of its semantics, $b$ is in a list of “un-trusted” agents, and that no element can possibly be removed from this list and that no message coming from un-trusted agents is possibly accepted.

2. One might examine all possible states of $a$ and establish for instance that no message is ever exchanged with $b$.

3. If the above is not possible, one must verify the property at run-time and undertake the necessary repairs, e.g., cancel $msg$ from $a$’s beliefs set and possibly undo its unwanted effects, if any.
Interval Temporal Logics

Existing Work

Oriented to systems that can be represented as finite state systems and analyzed using model-checking:

- **Linear Interval Temporal Logic** for automated reasoning about actions and change, provides binary operators such as *Before*, *After*, *Started_By* and similar.

- **Future Interval Logic** (graphical counterpart *Graphical Interval Logic*) has a notion of properties that should either be true or become true during an interval.

In both cases, an interval represents a sub-context for formulas to hold and can be characterized by an *initial* property, an *invariant* property and an *eventually* property.
**A-IMETATEM: Extending METATEM to Intervals**

Syntax in Logic and Syntax in (Logic) Programming: Future

<table>
<thead>
<tr>
<th>A-IMETATEM Op&lt;sup&gt;k&lt;/sup&gt;</th>
<th>OP(m,n;k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau(t) )</td>
<td>NOW( (t) )</td>
</tr>
<tr>
<td>( \bigcirc^k )</td>
<td>NEXT( (l; k) )</td>
</tr>
<tr>
<td>( \bigcirc^k_j )</td>
<td>NEXT( (j; k) )</td>
</tr>
<tr>
<td>( \Diamond^k )</td>
<td>FINALLY( (l; k) )</td>
</tr>
<tr>
<td>( \Diamond^k_m )</td>
<td>FINALLY( (m; k) )</td>
</tr>
<tr>
<td>( \Box^k )</td>
<td>ALWAYS( (l; k) )</td>
</tr>
<tr>
<td>( \Box^k_{m,n} )</td>
<td>ALWAYS( (m, n; k) )</td>
</tr>
<tr>
<td>( \Box^k_{(m,n)} )</td>
<td>ALWAYS_2( (m, n; k) )</td>
</tr>
</tbody>
</table>
## A-IMETATEM: Extending METATEM to Intervals

Syntax in Logic and syntax in (Logic) Programming: Never & Past

<table>
<thead>
<tr>
<th>A-IMETATEM Op$^k$</th>
<th>OP(m,n;k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N^k$</td>
<td>NEVER(1; k)</td>
</tr>
<tr>
<td>$N_{m,n}^k$</td>
<td>NEVER(m, n; k)</td>
</tr>
<tr>
<td>$\bullet_k$</td>
<td>LAST(1; k)</td>
</tr>
<tr>
<td>$\bullet_k^m$</td>
<td>LAST(j; k)</td>
</tr>
<tr>
<td>$\blacksquare_k^m$</td>
<td>P_ALWAYS(1; k)</td>
</tr>
<tr>
<td>$\blacksquare_k_{m,n}$</td>
<td>P_ALWAYS(m, n; k)</td>
</tr>
<tr>
<td>$\blacksquare_k^{\langle m,n \rangle}$</td>
<td>P_ALWAYS_2(m, n; k)</td>
</tr>
</tbody>
</table>
Semantics of A-IMETATEM operators is defined in an similar way to original METATEM operators.

To account for periodical verification at a certain frequency state subsequences are introduced:

- Consider infinite sequence $\sigma$ of states $s_0, s_1, \ldots$ of a system and $\sigma^k$ to be the subsequence $s_0, s_{k_1}, s_{k_2}, \ldots$ where for each $k_r$ ($r \geq 1$), $k_r \mod k = 0$, i.e., $k_r = g \times k$ for some $g$. Note that we have $\sigma^1 = \sigma$, $\sigma^2 = s_0, s_2, s_4, \ldots$ and so on.

- Let $O_p$ be any of the operators introduced in A-IMETATEM and $k \in \mathbb{N}$ with $k > 1$. Then $O_p^k$ is an operator whose semantics is a variation of the semantics of $O_p$ where the sequence $\sigma_s$ is replaced by the subsequence $\sigma_s^k$.
Basic A-IMETATEM rules

Syntax

An A-IMETATEM rule $\rho$ is a writing of the form $\alpha : \beta$ or simply $\beta$ where we have the following:

- $\beta$ is a conjunction including logic programming literals and/or (possibly negated) A-IMETATEM operators;
- $\alpha$ (if specified) is an atom of the form $p(t_1, \ldots, t_n)$ which is called the rule representative.
- Whenever $\beta$ is found to be true, its representative $\alpha$ is assumed to be recorded in the agent’s knowledge base.
Basic A-IMETATEM rules

Example

Goal $g$ if not achieved cannot be dropped:

\[ \neg \text{ever} (\text{not achieved}(g), \text{dropped}(g)) \]

Check when? Either upon action attempt or periodically.
Basic A-IMETATEM rules

Example

Goal $g$ if not achieved cannot be dropped:

$$NEVER(t_{init}, t_{end}, f) \ (not \ achieved(g), \ dropped(g))$$

$(t_{init}, t_{end})$: interval in which the property is checked, at frequency $f$. 
Basic A-IMETATEM rules

Example

Goal $g$ if not achieved cannot be dropped:

$$\textit{steadfast}(g, t_{\text{init}}, t_{\text{end}}) : \text{NEVER}(t_{\text{init}}, t_{\text{end}}; f) \ (\text{not achieved}(g), \text{dropped}(g))$$

Upon successful check representative will be recorded as $\textit{past event steadfastP}(g, t_{\text{init}}, t_{\text{end}})$. 
Contextual A-IMETATEM rules

Syntax

Let $\rho$ be an \textit{A-IMETATEM} rule.

The corresponding \textit{contextual A-IMETATEM rule} is a rule of the form $\rho :: \chi$ where:

- $\chi$ is called the \textit{evaluation context} of the rule, and consists of a conjunction of logic programming literals;
- every variable occurring in $\rho$ must occur in an atom (non-negated literal) of the context $\chi$. 
Contextual A-IMETATEM rules

Example

A generic goal $G$ with timeout $T$ cannot be dropped if the timeout has not expired ($NOW(T1)$ returns current time):

$$NEVER\ (not\ achieved(G),\ dropped(G)) \ ::\$$

$$\ (goal(G),\ deadline(G, T),\ NOW(T1),\ T1 \leq T)$$
Contextual A-IMETATEM rules

Example

A generic goal $G$ with timeout $T$ cannot be dropped if the timeout has not expired ($NOW(T1)$ returns current time):

$$NEVER (not\ achieved(G), \ dropped(G)) :: (goal(G), \ deadline(G, T), \ NOW(T1), T1 \leq T)$$
Contextual A-IMETATEM rules with improvement/repair

Syntax

An A-IMETATEM rule with improvement is a rule of the form:
\[ \rho ::= \chi \div \psi, \text{ or } \alpha P \div \psi \]
where:
- \[ \rho ::= \chi \] is a contextual A-IMETATEM rule;
- \[ \alpha P \] is the recorded representative of a contextual A-IMETATEM rule;
- \[ \psi \] is called the improvement action of the rule, and it consists of an atom \[ \psi \].

The left-hand-side is called the monitoring condition of the rule, its negation is called the check condition of the rule.
Contextual A-IMETATEM rules with improvement

Example

\[ N - NEVER (not\ achieved(G), dropped(G)) :: (goal(G), deadline(G, T), NOW(T1), T1 \leq T) \div inc\_comt(T1) \]

\[ incr\_comt(T) \leftarrow level(commitment, L), \]
\[ increase\_level(L, L1), \]
\[ assert(neg(commitment\_mod(L))), \]
\[ assert(commitment\_mod(L1)) \]

Whenever the check condition is violated for some specific goal \( g \), i.e., its negation \( N - NEVER \) holds, upon detection of the violation the system will attempt the improvement (in this case a repair) action consisting in executing \( inc\_comt(t) \).
Contextual A-IMETATEM rules with improvement/repair

Operational Behavior

- Each A-IMETATEM rule $\rho$ is attempted at the frequency stated for it in $CI$ (or at a default frequency).
- If the monitoring condition of $\rho$ holds when the rule is checked (or, symmetrically, the check condition is violated), then the improvement action $\psi$ is performed.
- The improvement action is specified via an atom that is executed as an ordinary logic programming goal.
Use of A-IMETATEM rules with improvement
Example with rule representative

A-IMETATEM axioms can be used to check the past behavior and knowledge of the agent but also to influence its future behavior.

The agent evolution in fact entails also an evolution of recorded information.

Recorded information may affect the evaluation of social factors such as trust, confidence, etc.
Contextual A-IMETATEM rules with improvement

Example with rule representative

Example below: level of trust increased for agents that have proved themselves reliable in communication during a test interval.

\[ \text{Rel}_\text{Ag}(Ag) : \text{ALWAYS}(m, n; k) \text{ reliable}(Ag) \]
\[ \text{Rel}_\text{Ag}(A)P \div \text{increase\_trust\_level}(A) \]

Increase of the level of trust as an improvement, in effect as soon as the A-IMETATEM rule is checked. The improvement is determined based on recorded representatives.
Agent Model
Evolutionary Semantics

- An agent starts from an initial program $P_0$ obtained by agent program $\mathcal{P}_M$ by knowledge compilation.

- Changes inside an agent which are determined either by changes in the environment as well as agent’s own self-modifications are modeled as *Program Transformation Steps*.

- Each program-transformation step makes $P_i$ evolve into $P_{i+1}$. 
Agent Model

Evolutionary Semantics

- We thus have a Program Evolution Sequence $PE = [P_0, \ldots, P_n]$.

- We have a corresponding Semantic Evolution Sequence $ME = [M_0, \ldots, M_n]$ where $M_i$ is the semantic account of $P_i$ according to the specific language and the chosen semantics.

- The couple $\langle PE; ME \rangle$ is called the *Evolutionary Semantics* of the agent program $P_{Ag}$. 
Agent Model
Multi-layered agent structure

Basic assumption: logic programming (in one of its variants).

- An agent is considered to be composed of two distinct interacting layers: the BA (or base layer, or ground layer) and (one or more) Meta-level(s);
- MA (that stands for Meta-Agent) along with the IEA (Information Exchange Agent), constitutes the Meta-Level.
Agent Model
Agent Program

An *agent program* $\mathcal{P}_M$ is a tuple $\langle BA, MA, C, CI \rangle$ of software components where: $BA$ and $MA$ are logic programs, $C$ is the control component and $CI$ (optional) contains some kind of control information.

- $BA$ and $MA$ are logic programs,
- $C$ is the control component (based upon an *underlying control mechanism* $U^M$),
- $CI$ contains control information,
A-IMETATEM
Operational interpretation

- Events that happen to the agent are recorded as *Past events* giving rise to program transformation steps;
- A-IMETATEM rules are invoked as stated in $C$ at a frequency stated in $CT$;
- If the property expressed by an A-IMETATEM rule holds this may lead to a program transformation step involving:
  - a *repair action* if the property is undesirable;
  - an *improvement action* if the property is desirable and the agent behavior can be modified accordingly.
Concluding Remarks

- Proposal: an approach to the definition and the run-time verification of properties based upon a temporal-logic-like axioms and meta-axioms defined on time intervals.
- Suitable improvement/repair actions that may imply modifications to the agent’s program components and knowledge base.
- Implementation is under way
- Experiments with DALI *Internal Events* (similar operational mechanism show that performance not only is not worse, but is even better than without.)
A-IMETATEM: Derived Operators
Syntax in Logic Programming

- Given a sentence $\varphi$ and a related frequency $f_\varphi \in \mathbb{N}$ we have:
  
  \[ USUALLY(m, n) \varphi \equiv ALWAYS(m, n; f_\varphi) \varphi. \]

  $USUALLY \varphi$ stands for $ALWAYS(start, now; f_\varphi) \varphi$.

- Given a sentence $\varphi$ we have:
  
  \[ SOMETIMES(m, n) \varphi \equiv \neg ALWAYS(m, n) \varphi \land \neg NEVER(m, n) \varphi. \]

- Given $OP(m, n; k)$, we define
  
  \[ N \rightarrow OP(m, n; k) \text{ standing for } \neg OP(m, n; k). \]