A new proof system to verify GDT agents

Bruno Mermet, Gaële Simon

A new proof system to verify GDT agents

Bruno Mermet, Gaele Simon

Laboratoire GREYC, UMR 6072 & Université du Havre
Campus Côte de Nacre
Boulevard du Maréchal Juin
BP 5186 - 14032 CAEN cedex
Bruno.Mermet@univ-lehavre.fr
Gaele.Simon@univ-lehavre.fr

Abstract The GDT4MAS model is dedicated to the formal specification of agents and multiagents systems. It has been presented in previous articles [7, 6]. The proof mechanism relies on *proof schemas* that generate *proof obligations*, that is to say first-order formulae that can be proven (in most cases) by an automatic prover (such as PVS). In this article, we present a new version of proof schemas that increase the number of proofs that can be performed.

1 Introduction

The GDT model has been first presented five years ago [7]. This model consists in a language to formally specify agents, a formal semantics, and a set of proof schemas to guarantee the correctness of the behaviour specified. It has been extended a few years later to specify and verify multiagent systems [6].

However, when applying the model and the proof system to concrete case studies, it appeared that true properties were unverifiable because necessary hypotheses were lacking. Thus, we propose here a new proof system that provides richer hypotheses.

In the next section, we briefly recall the main concepts of the GDT4MAS model. In section 3, we present the old proof schema principles, and we illustrate its weaknesses. Section 4 is dedicated to the presentation of the new proof schemas. Finally, section 6 concludes on this new proof mechanism.

2 The GDT4MAS model

2.1 Main concepts

In the GDT4MAS model, the MAS is described by an environment, mainly described by variables, and a population of agents evolving in this environment. Each agent is described as an instance of an agent type. As a consequence, in the rest of this section, after a short description of the notations we used, we begin by describing the notion of agent type, and of agent behaviour.

2.2 Notation

Notation 2.1 (primed and unprimed variable) *When the value of a variable v in two execution states is considered, the value of v in the first state, called the current state, is written v, and its value in the second state is written v'. For instance, the action consisting in increasing the value of v by 1 is specified by the postcondition $v' = v + 1$.*

2.3 Agent Type Specification

Simplified Definition 2.1 (Agent Type) *An agent type t is described by a name ($name_t$), a set of internal variables ($VarI_t$), a set of surface variables ($VarS_t$), an invariant (i_t), and a behabiour (b_t).*

In this definition, an *internal variable* is a variable that only the owner agent can see and modify (compare it to a private attribute in the object model); a *surface variable* is a variable that only the owner agent can modify, but that can be seen by the other agents (compare it to a private attribute with a public getter method); an *invariant* is a predicate defined on the internal and surface variables of the agent type and that must always be true for every agent of the given type; and the *behaviour* of an agent is specified by a Goal Decomposition Tree, defined later in this section.

Simplified Definition 2.2 (Action) *An action a is specified by a name ($name_a$), a precondition (pre_a), a postcondition ($post_a$), an ns flag (ns_a) and a gpf (gpf_a). The precondition is a predicate specifying when the action is enabled, the postcondition specifies what that action does ($x' = x - 1$ for instance expresses that the action decreases the value of x by 1), the ns flag has the value NS (necessarily Satifiable) if the action is guaranteed to always succeed, and NNS if the action may fail, and the gpf, the Guaranted Property in case of Failure, is a predicate specifying what is however guaranteed to be true if the action fails.*

Definition 1 (Goal Decomposition Tree (GDT)). A Goal Decomposition Tree describes the behaviour of the agents of a given type. Each node of this tree is a *GDT goal*. The tree structure is defined thanks to the decomposition of each GDT goal into subgoals.

Definition 2 (GDT goal). A GDT goal g is described by a name ($name_g$), a satisfaction condition (sc_g), a gpf (gpf_g), a decomposition, an ns flag (ns_g) and a laziness flag (l_g). The satisfaction condition is a predicate specifying what the goal must establish, the gpf is a predicate specifying what is guaranted to be established if the execution of the goal fails, the ns flag specifies whether the goal always succeed or not, and the laziness flag specifies whether the goal decomposition is executed when the satisfaction condition of the goal is already true when the goal is considered.

Please notice that in this article, in order to simplify the formulae and their understanding by the reader, we only consider non-lazy goals.

Definition 3 (Goal decomposition). A GDT goal is either a leaf goal or an intermediate goal. In the latter case, the goal is decomposed into one or several subgoals, thanks to a *decomposition operator*. A list of decomposition operators can be found in [7].

Among others, we can introduce the following decomposition operators:

- **SeqOr**: Sequential Or. It decomposes the parent goal into several subgoals N_i. Subgoals are considered from the the left to the right. If the considered subgoal succeeds, the parent goal is achieved and the execution of the decomposition is ended. But if it fails, the next subgoal is considered. If the last subgoal is reached and fails, the satisfaction condition of the parent goal must be evaluated to know if it is achieved or not.
- **SeqAnd**: Sequential And. It decomposes the parent goal into several subgoals N_i. Subgoals are considered from the left to the right. If the considered subgoal succeeds, the next one is considered. If the last subgoal is considered and succeeds, the parent goal is achieved. But if it fails, the satisfaction condition of the parent goal must be evaluated to determine whether the parent goal is achieved or not.
- **SyncSeqOr** and **SyncSeqAnd**: These operators are similar to the SeqOr and SeqAnd operator, but environment variables can be locked during the whole execution of the parent goal decomposition.
- **case**: The *case* operator is similar to the *switch* statement of most imperative languages: it decomposes a goal into several subgoals, with a choice condition associated to each subgoal. When the parent goal must be achieved, one of the subgoals whose choice condition is true is executed.
- **iter**: This operator decomposes a goal into one subgoal. The achievement of this subgoal will be attempted as long as the parent goal is not achieved.

2.4 Properties proven by the method

The GDT4MAS method allows to prove several kinds of properties. We first prove invariant and liveness properties, at the agent-type level and at the system-level. We recall here that invariant properties are properties that must be always true, and that liveness properties are properties that must eventually be true. Moreover, the proof-system of the method verifies that goal decompositions are valid. In this article, we focus on the proof of decompositions and of invariant properties. This is the topic of the next section.

3 Previous proof system

3.1 Principles

The main principles of the previous proof system consists in the following steps:

- A *context* is inferred for each node, in a top-down manner, and from left to right when oriented operators are used (like SeqAnd and SeqOr);

- A *gpf*, Guaranted property in case of failure, is inferred for each NNS goal, in a bottom-up manner;
- A *proof schema* is used for each decomposition operator, in order to generate a *proof obligation*, that is to say a first-order formula, whose proof can be attempted by any first-order logic theorem prover.

Consider the GDT presented in figure 1.

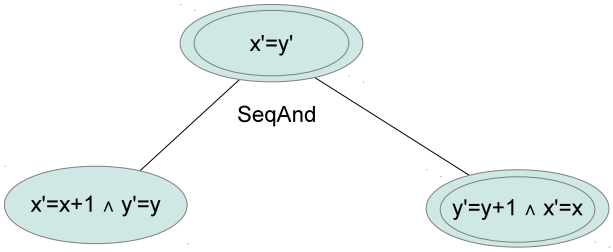

Fig. 1 Simple GDT

Several states can be determined during the execution of this tree. These states are detailed in figure 2.

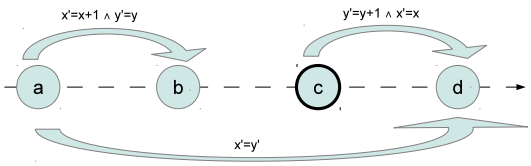

Fig. 2 Execution trace

Between states b and c, other agents may act on the environment. So, the solution we have chosen is a projection of formulae on internal variables. This lead us to the following context propagation rules and proof schemas for the SeqAnd Operator, when a goal n is decomposed into n_1 *SeqAnd* n_2.

3.1.1 Context Propagation rule

$$C_{n_1} = C_n \tag{1}$$

$$C_{n_2} = ((sc_{n_1})_{ri})_r \tag{2}$$

In these formulae:

- sc_{n_1} is the satisfaction condition of goal n_1;
- $\mathcal{F}_{ri}$ is the projection of formula $\mathcal{F}$ on the primed internal variables, preserving all the properties entailed by $\mathcal{F}$ on the primed internal variables. For instance, if

$\mathcal{F} \equiv (x = 2 \wedge y = 3 \wedge x' > y \wedge y' = x + 1)$ with x and internal variable and y an environment variable, we have $\mathcal{F}_{ri} \equiv (x = 2 \wedge y = 3 \wedge x' > 3)$.

- $\mathcal{F}_r$ is the projection of a formula $\mathcal{F}$ onto the right, that is to say on primed variables, where primes have been removed. For instance, if $\mathcal{F} \equiv (x = 2 \wedge y = 3 \wedge x' > 3)$, we have $\mathcal{F}_r \equiv (x > 3)$.

3.1.2 Guaranted Property in case of Failure

In the GDT4MAS model, the achievement of an agent goal may fail (it can a consequence of the behaviour of another agent in the system for instance). When a goal execution fails, the situation reached is however not completely chaotic; certain properties are guaranted to be true. For instance, if an agent has to open a door and it fails, it is not magically teleported into another room. So, we associate to each goal that may fail (NNS goal: Non-Necessarily Satisfiable goal) a *Guaranted Property in case of Failure* (or GPF). This property is inferred from leaf goals thanks to GPF propagation schemas.

3.1.3 Proof schema

When we defined the proof schema for the SeqAnd operator, we considered that the proof obligation should contain three types of variables:

- *unprimed variables*, corresponding to the value of variables in state a on figure 2;
- *primed variables*, corresponding to the value of variables in state d on figure 2;
- *temporary variables*, corresponding to the value of variables in state c on figure 2. These variables are subscripted by tmp and are computed from the satisfaction condition of the first subgoal, after a projection on the internal variables in order to take into account the fact that other agents may have modified environment variables between states b and c.

Thus, we demonstrated that the following proof schema was valid:

$$i_\epsilon \wedge i_A \wedge C_n \wedge \left(T^{tmp}(sc_{n_1})\right)_{ri} \wedge T'_{tmp}(sc_{n_2}) \rightarrow sc_n \tag{3}$$

Where:

- i_ϵ is the invariant of the environment;
- i_A is the invariant of the agent;
- C_n is the context of node n;
- $T^{tmp}(\mathcal{F})$ is the transformation of formula $\mathcal{F}$ where each primed variable becomes unprimed and subscripted by tmp. For instance, $T^{tmp}(x = y' + 1) \equiv x = y_{tmp} + 1$;
- $T_{tmp}{}'(\mathcal{F})$ is the transformation of formula $\mathcal{F}$ where each unprimed variable becomes subscripted by tmp. For instance, $T'_{tmp}(x = y' + 1) \equiv x_{tmp} = y' + 1$;

3.2 Limits

3.2.1 Projections

The main limit of the approach defined above is that true projections, as defined in section 3.1.1, cannot be computed automatically. A simplified version can be com-

puted, by removing each term of the conjunctive normal form containing another variable that those concerning the projection. For instance, if a projection of the formula $\mathcal{F} \equiv (x = 2 \wedge y = 4 \wedge x = z + 2 \wedge z > 3)$ on x is required, we obtain $\mathcal{F}_x \equiv (x = 2)$. However, this formula is weaker that what we would like to obtain (In particular, the fact that x is greater than 5 is lost).

3.2.2 Weak contexts

As shown in equation 2, The context inferred for a goal is quite weak: it only depends on the execution of the previous goal: its satisfaction condition if the operator is a Seqand, or its guaranted property in case of failure if the operator is a SeqOr. But more information could be preserved from the context in which this previous goal is executed. For instance, if node n is decomposed into $n_1 SeqAnd n_2$, if the context of node n_1 is $x = y$, if x and y are internal variables and if the satisfaction condition of node n_1 is $sc_{n_1} \equiv x' = x + 1 \wedge y' = y + 1$, we know that $x = y$ when node n_2 is considered. But this cannot be automatically inferred by the contexte propagation rules.

3.2.3 Weak proof schemas

In the proof schema given in equation 3, some important hypotheses miss. For instance, even if environment variables can be modified between states b and c in figure 2, the invariant of the environment is preserved. But this is not specified in this formula.

3.2.4 General observations

Most of the weaknesses highlighted above can be bypassed by re-inforcing satisfaction conditions of goals. However, this is not a satisfying solution, because it requires more work to the developper, and it makes the specification more complicated. Thus, it appeared necessary to re-inforce the proof mechanism to solve these drawbacks.

4 The new proof system

To define the new proof system, we need new notations, that are introduced in the following section.

4.1 Predicate transformers

Notation 4.1 (At) *Let f a predicate. $f[i]$ is a predicate where each non-subscripted variable in f is subscripted by i.*

Example: $(x = y_0)[1] \equiv (x_1 = y_0)$.

Notation 4.2 (Between) *Let f a predicate. $f^{i \to j}$ is a predicate derived from f where each unprimed and unsubscripted variable is subscripted by i and each primed variable becomes unprimed and subscripted by j.*

Example: $(y' < x \wedge x' = x_0)^{1 \to 2} \equiv (y_2 < x_1 \wedge x_2 = x_0)$.

Notation 4.3 (Temporal switch) *Let f a predicate. $f^{\to i}$ is predicate derived from f where each subscript is increased by i.*

Example: $(x = x_1 \wedge y_2 = x_1)^{\to -2} \equiv (x = x_{-1} \wedge y_0 = x_{-1})$.

Notation 4.4 (Priming) *Let f a predicate. If f contains at least one primed variable, then $pr(f) = f$. Otherwise, $pr(f)$ is the predicate derived from f where each unsubscripted variable is primed.*

Examples: $pr((x = x_0)) \equiv (x' = x_0)$ and $pr((x = x')) \equiv (x = x')$.

Notation 4.5 (Stability) *Let t and agent type with two internal variables via, vib and one surface variable vs (internal and surface variables are described in the next section). Then, when one agent a of this type is considered $stab^{i \to j}$ is the predicate $via_i = via_j \wedge vib_i = vib_j \wedge vs_i = vs_j$.*

Notation 4.6 (Untemporalization) *Let f a predicate. $f^{\divideontimes}$ is the formula f in which all subscripts of value x are removed.*

Example: $(x_1 = x_2)^{\divideontimes} \equiv x = x_2$.

Notation 4.7 (Invariant) *Let A an agent situated in an environment $\mathcal{E}$. We write:*

- *i_A the invariant associated to the internal variables of the agent;*
- *$i_\mathcal{E}$ the invariant associated to the environment variables;*
- *$i_{\mathcal{E}A}$ the conjunction of i_A and $i_\mathcal{E}$.*

4.2 Context inference

In a context formula, it may be necessary to refer to the value of a variable in a previous state. In that case, the variable is subscripted by a negative integer. The value in the current state is represented by the variable name neither subscripted nor primed. For instance, consider the GDT presented in figure 1, and suppose the context of the root goal is $x = y$.

In the state in which the right subgoal is considered (with a bold outline in figure 2), we know that:

- in state a (numbered -2), from the parent goal, x and y are equals; So: $x_{-2} = y_{-2}$;
- between states a and b (numbered -1), the value of x is increased, whereas the value of y is preserved, and so : $x_{-1} = x_{-2} + 1 \wedge y_{-1} = y_{-2}$;
- between state b and the state in which the right subgoal is considered, if x and y are internal variables (and thus, cannot be modified by other agents), we have: $x = x_{-1} \wedge y = y_{-1}$.

So, the context of the right subgoal is:

$$ctx = \begin{cases} (x_{-2} = y_{-2}) \wedge (x_{-1} = x_{-2} + 1 \wedge y_{-1} = y_{-2}) \\ (x = x_{-1} \wedge y = y_{-1}) \end{cases}$$

Please notice that this allows to deduce that in the state in which the right subgoal is considered (state c), we have $x = y + 1$.

Finally, the new context inference rules are the following:

$$\begin{cases} C_{N_1} = C_N \\ C_{N_2} = \left(\left((C_{N_1} \wedge sc_{N_1})^{0 \to 1} \wedge stab^{1 \to 2} \wedge i_{\mathcal{E}A}[1] \wedge i_{\mathcal{E}A}[2] \right)^{\to -2} \right)^{\alpha} \end{cases} \quad (4)$$

Using such rules, contexts are guaranted to use neither primed variables nor variables with positive subscripts. Indeed, if its true for C_N:

- it is trivially true for C_{N_1};
- it is true for C_{N_2} because:
  - $(C_{N_1} \wedge sc_{N_1})^{0 \to 1}$ contains variables with negative subscripts from C_{N_1}, and variables subscripted by 0 and 1;
  - $stab^{1 \to 2}$ contains variables subscripted by 1 and 2;
  - $i_{\mathcal{E}A}[1]$ contains variables subscripted by 1;
  - $i_{\mathcal{E}A}[2]$ contains variables subscripted by 2;
  - From the previous facts, the formula

$$\left((C_{N_1} \wedge sc_{N_1})^{0 \to 1} \wedge stab^{1 \to 2} \wedge i_{\mathcal{E}A}[1] \wedge i_{\mathcal{E}A}[2] \right)^{\to -2}$$

    contains variables with negative or null subscripts;
  - and so, C_{N_2} contains variables with negative subscripts or unprimed variables without subscripts.

4.3 Proof schema

In this new proof mechanism, the context of the right subgoal, in the case of the SeqAnd operator, contains the essential properties that are guaranted to be true when the right subgoal is considered. So, as the semantics of the SeqAnd operator is that, when the right subgoal is executed and succeed, the parent goal must be achieved, the proof schema associated to the SeqAnd operator only consists in verifying that when the right subgoal succeeds in its context, then the parent goal also succeeds. This leads to the following proof schema:

$$\left(C_{N_2}[0] \wedge sc_{N_2}^{0 \to 1} \wedge i_{\mathcal{E}A}[1] \right) \to sc_N^{-2 \to 1} \quad (5)$$

In this formula, we can notice that the values of variables in state a of figure 2 are represented by the variables subscripted by -2, values in state b are represented by the variables subscripted by -1, values in state c are represented by variables subscripted by 0 and values in state d are represented by variables subscripted by 1.

In order to give to the reader an intuitive meaning for this proof schema, it can be explained as follows:

- We consider the case where the second subgoal, N_2, is executed, which occurs when its context C_{N_2} is verified. We assign to the state the number 0;
- We only consider the case where this subgoal succeed, reaching another state numbered 1, hence the formula $sc_{N_2}^{0 \to 1}$;

- We also know that in this new state, the environment variable and the agent variable are true. So we have the hypothesis $i_{\mathcal{E}A}[1]$;
- Finally, if all these hypotheses are true, the satisfaction condition of the parent goal must be established between the state in which the parent goal execution has begun (numbered -2) and the state in which the execution of the second subgoal ends (numbered 1).

4.4 Guaranted Property in case of Failure

In the previous proof system, GPF were inferred in a bottom-up manner from leaf goals. This characteristics presented at lease two drawbacks:

- The GDT must be completely specified to be proven. Indeed, leaf goals must be known to infer GPF of intermediate goals, and these GPFs are required to prove the correctness of the GDT, as they are used in some proof schemas;
- The GPF inferred may be very complicated and may hold many non-necessary hypotheses, that may reduce the success rate of automatic provers.

So, in the new proof system, it is asked to the developper to explicitely give GPF of each node, as he has to do for satisfaction condition. This is not a harder work than specifying satisfaction condition, and it makes the system quite more compositional. However, it must be checked that the given GPF are entailed by the behaviour specified. Thus, we must give GPF inference rules and we have toverify that the GPF given by the developper is entailed by the GPF inferred from the GDT.

For the SeqAnd operator, the parent goal may fail either when the first subgoal fails (in that case, its gpf is verified in the context of the execution of the first subgoal) or when the second subgoal fails (the gpf of this subgoal is then verified in the context of the execution of this seconde subgoal). So we have:

$$infgpf_N = \begin{array}{l} \big(C_{N_1} \wedge gpf_{N_1}\big) \\ \vee \\ \big(C_{N_2}[0] \wedge gpf_{N_2}[0]\big)^{\times 2} \end{array} \tag{6}$$

We can notice that in the formula corresponding to the inferred gpf of a goal, unprimed variables correspond to the values of the variables when the execution of the goal begins, and primed variables correspond to the values of the variables when the execution of the goal ends.

We have now to prove that the specified gpf is correct. Thus, for each NNS goal, we must establish that when the decomposition fails (that it to say, the inferred gpf is true), either the parent goal is achieved or its gpf is true. Hence the following proof schema:

$$infgpf_N \rightarrow (sc_N \vee gpf_N) \tag{7}$$

5 Related works

Several works deal with the formal specification of multi-agent systems, but just a few consider the formal verification of their specification. Moreover, most of these

systems use model-checking [1, 8, 5]. The most recent and promising work in this area is surely the Agent Infrastructure Layer (AIL) [3] and its connection with AJPF (Agent JPF), an adaptation of the Java Path Finder model-checker. AIL is a kind of intermediate language that can be used to give a formal semantics to most BDI agent languages (such as AgentSpeak or MetateM). JPF is however not a very efficient model-checker. Although there are best model-checkers such as SPIN [4], this technique is not well-suited to massive multi-agent systems, where the state-space is too huge, especially because of the great interleaving possibilities between agent actions, and as a consequence, only systems with few agents manipulating boolean concepts can be proven. On the other hand, the system we propose make possible the automatic verification of huge systems, manipulating complex data using an any existing first-order theroem prover (it has been tested with PVS [9] and krt [2]).

6 Conclusion

The previous proof system for GDT4MAS agents was of course valid, but it was difficult to perform proofs with intuitive goal specifications: the satisfaction conditions that the developper had to give had to be really proof-oriented, and using the model was not really feasible by a developper who did not know all the details of the proof process. With the new principles presented in this article, the specification task is more independant from the proof process. Of course, the developper must be well-heeled with the predicate logic, but it it not necessary for him to understand how the proof works. Using the method is thus easier. Moreover, the number of proofs that can be performed is increased.

References

1. Bordini, R., Fisher, M., Pardavila, C., Wooldridge, M.: Model-checking AgentSpeak. In: AAMAS-03. Melbourne, Australia (2003)
2. Clear-Sy: B for free. Http://www.b4free.com
3. Dennis, L., Fisher, M., Webster, M., Bordini, R.: Model Checking Agent Programming Languages. Automated Software Engeneering Journal **19**(1), 3–63 (2012)
4. Holzmann, G.J.: The Model Checker SPIN. IEEE Trans. Softw. Eng. **23**, 279–295 (1997). DOI 10.1109/32.588521. URL http://dl.acm.org/citation.cfm?id=260897.260902
5. Kacprzak, M., Lomuscio, A., Penczek, W.: Verification of multiagent systems via unbounded model checking. In: Autonomous Agents and Multi-Agent Systems (AAMAS'04) (2004)
6. Mermet, B., Simon, G.: GDT4MAS: an extension of the GDT model to specify and to verify MultiAgent Systems. In: D. *et al.* (ed.) Proc. of AAMAS 2009, pp. 505–512 (2009)
7. Mermet, B., Simon, G., Saval, A., Zanuttini, B.: Specifying, verifying and implementing a MAS: A case study. In: M. Dastani, A.E.F. Segrouchni, A. Ricci, M. Winikoff (eds.) Post-Proc. of ProMAS'07, no. 4908 in Lecture Notes in Artificial Intelligence, pp. 172–189. Springer (2007)
8. Raimondi, F., Lomuscio, A.: Verification of multiagent systems via orderd binary decision diagrams: an algorithm and its implementation. In: Autonomous Agents and Multi-Agent Systems (AAMAS'04) (2004)
9. SRI International: PVS. http://pvs.csl.sri.com