Automatic Data-Abstraction in Model Checking Multi-Agent Systems

Alessio Lomuscio[1], Hongyang Qu[2], and Francesco Russo[1]

[1] Department of Computing, Imperial College London, London, UK
[2] Computing Laboratory, Oxford University, Oxford, UK

Abstract. We present an automatic data-abstraction technique for the verification of the universal fragment of the temporal-epistemic logic CTLK. We show the correctness of the methodology and present an implementation operating on ISPL models, the input files for MCMAS, a model checker for multi-agent systems. The experimental results point to the attractiveness of the technique in a number of examples in the multi-agent systems domain.

1 Introduction

Over the past few years model checking techniques [3] have been extended to temporal-epistemic logics [11]. Several model checkers, including MCMAS [15], McK [12] and Verics [14], are now available supporting this and other extended functionalities. Expressive specification languages find a natural application in the area of multi-agent systems [18]. Because of their autonomous nature multi-agent systems (MAS) naturally generate very large state spaces. Therefore, being able to tackle the state-space explosion remains of fundamental importance if we are to develop model checking techniques for the verification of MAS. While a number of abstraction-based techniques have been put forward for plain temporal logic, e.g., [2,6], little attention has gone so far towards developing efficient state-reduction methodologies preserving the validity of temporal-epistemic specifications. Crucially, there is no automatic implementation enabling the user to perform automatic abstraction directly on the program. This paper aims to make a first attempt at filling this gap.

In line with much of the literature we represent MAS as interpreted systems. In this paradigm we model agents by programming the local evolution of the agents' data in their local states. We use ISPL, the input language of MCMAS, to program interpreted systems. In this paper we show that data-abstraction notions can be defined on interpreted systems semantics and automatic reduction can be performed directly on ISPL programs. We illustrate the technique on two scenarios inspired by popular examples in the MAS literature: card games [9], and the bit transmission problem [11]. Both the scenarios we consider have over 10^{10} reachable states so are too large to be checked by MCMAS directly, but can be verified effectively by model checking the reduced program.

R. van der Meyden and J.-G. Smaus (Eds.): MoChArt 2010, LNAI 6572, pp. 52–68, 2011.

Related work. Abstraction for epistemic specifications was defined on Kripke models in [7,10]. However, given these structures are computationally ungrounded [17], it is difficult to apply these results to MAS descriptions. Previous work by some of the authors of this paper [5] established the basic theoretical framework for abstraction on interpreted systems, but the results were not applied to data-abstraction specifically, nor to concrete ISPL programs, the basis for this investigation.

More broadly, three main approaches are prominent in abstraction techniques for temporal logic. The first focuses on studying partial symmetries of the system under investigation [4]. In the second, predicate abstraction, introduced in [13], the system is described by a set of logical formulas; a finite set of local state predicates are selected, and any two local states satisfying exactly the same predicates are collapsed. Finally, the third technique, introduced by Cousot [6] and further developed by Clarke at al. [2,1] involves automatically reducing local states by collapsing the data values they are built from. This is the basis for the technique presented here.

Outline of the paper. The rest of the paper is organised as follows. Section 2 describes the interpreted system framework, the temporal epistemic specification logic ACTLK, the model checker MCMAS, and abstraction of interpreted systems. Section 3 reports the theoretical basis of this investigation, namely that ACTLK properties are preserved under abstraction, as well as details of the implementation. Section 4 presents experimental results. We conclude in Section 5.

2 Interpreted Systems, ACTLK and Abstraction

We use interpreted systems [11] as a semantic model for multi-agent systems. In this formalism a system is composed of n agents and an environment. Each agent and the environment are associated with a set of *local states* and a set of *actions*. The local states are private to the agent and the environment. *Local protocols* define the actions that may be executed at a given local state. The *local evolution function* of the agent defines the transition relation among the local states. The environment has the same structure as the agents. The formal definition of an interpreted system is given as follows.

Definition 1 (Interpreted system). *An interpreted system over a set* $Ag = \{1, \ldots, n\}$ *of agents and an environment* e *is a tuple*

$$\mathcal{I} = \langle \{L_i\}_{i \in Ag} \cup \{L_e\}, \{ACT_i\}_{i \in Ag} \cup \{ACT_e\}, \{P_i\}_{i \in Ag} \cup \{P_e\}, \{t_i\}_{i \in Ag} \cup \{t_e\}, I_0, V \rangle$$

where:

- L_i (L_e, *respectively*) *is a non-empty set of possible local states for agent* $i \in Ag$ *(the environment, respectively). The set of possible global states is denoted by* $S = L_1 \times \cdots \times L_n \times L_e$. *For any global state* $g \in S$, *we write* g_i *for the* i*-th component in* g, *i.e., the local state of agent* i *in* g. *Similarly,* g_e *represents the local state of* e *in* g.

- ACT_i *(ACT_e, respectively) is a non-empty set of possible actions for agent* $i \in Ag$ *(the environment, respectively). The set of possible joint actions is denoted by* $ACT = ACT_1 \times \cdots \times ACT_n \times ACT_e$.
- $P_i : L_i \longrightarrow 2^{ACT_i}$ *is the local protocol for agent* i *and* $P_e : L_e \longrightarrow 2^{ACT_e}$ *is the local protocol for the environment.*
- $t_i : L_i \times ACT \longrightarrow 2^{L_i}$ *is the local evolution function for agent* i*, and* $t_e : L_e \times ACT \longrightarrow 2^{L_e}$ *for the environment.*
- $I_0 \subseteq S$ *is a non-empty set of initial states.*
- $V : S \longrightarrow 2^{AP}$ *is the evaluation function for the set* AP *of atomic propositions.*

The Cartesian product of the local evolution functions denotes the global evolution function that describes how the system evolves from a global state to the next one. Let $\overline{a} \in ACT$ be a joint action and a_i the action of agent i ($i \in Ag$) in $\overline{a}$, as well as a_e for the environment.

Definition 2 (Global transition relation). *Given an interpreted system* $\mathcal{I}$*, the global transition relation* $\mathcal{T} \subseteq S \times ACT \times 2^S$ *is such that* $\langle g, \overline{a}, S' \rangle \in \mathcal{T}$ *(where* $S' \subseteq S$*) if and only if:*

$$\big(\forall i \in Ag : \langle g_i, \overline{a}, S'_i \rangle \in t_i \ \wedge \ \langle g_i, a_i \rangle \in P_i\big) \wedge \langle g_e, \overline{a}, S'_e \rangle \in t_e \ \wedge \ \langle g_e, a_e \rangle \in P_e$$

where $S'_i \subseteq L_i$ and $S'_e \subseteq L_e$. In the following we assume that the global transition relation $\mathcal{T}$ is total, i.e., for every $g \in S$, there is $S' \subseteq S$ such that $g \mathcal{T} S'$ and $S' \neq \emptyset$.

Definition 3 (Path). *A path* π *in* $\mathcal{I}$ *is an infinite sequence* $g^0 g^1 \ldots$ *of global states in* S *such that every pair of adjacent states forms a transition, i.e.,* $g^k \mathcal{T} g^{k+1}$ *for all* $k \geq 0$*. Let* $\pi(k)$ *be the* k*th global state in* π*, i.e.,* g^k.

As standard [11] the knowledge of an agent is defined by means of relations over global states defined as follows.

Definition 4 (Epistemic indistinguishability relation). *The epistemic indistinguishability relation for agent* i *in system* $\mathcal{I}$ *is:*

$$\sim_i = \{\langle g, g' \rangle \in S \times S \mid g_i = g'_i\}$$

ACTLK logic. We consider specifications expressed in the logic ACTLK [16], which adds epistemic modalities to the temporal logic ACTL, the universal fragment of Computation Tree Logic [8].

Definition 5 (ACTLK). *ACTLK formulae over a set* Ag *of agents and a set* AP *of propositions are defined by:*

$$\phi ::= \alpha \mid \neg\alpha \mid \phi \wedge \phi \mid K_i\phi \mid AX\phi \mid A(\phi U \phi) \mid A(\phi R \phi)$$

where $\alpha \in AP$ *and* $i \in Ag$.

As customary, a formulas $K_i\phi$ is read as "Agent i knows ϕ". The formula $AX\phi$ specifies that "for all paths ϕ holds in the next state of the path"; the formula $A(\phi U\psi)$ specifies that "along all paths ϕ holds until ψ holds"; the formula $A(\phi R\phi')$ specifies that "along all paths, ϕ releases ϕ'". Other universal temporal operators AF and AG can be equivalently expressed as $AF\phi = A(trueU\phi)$ and $AG\phi = A(falseR\phi)$.

The combination of temporal and epistemic modalities allows us to specify how agents' knowledge evolves over time. For example, $AG(\alpha \rightarrow AF(K_iK_j\psi))$ expresses that whenever α holds, eventually agent i will know that agent j knows ψ.

Given an interpreted system $\mathcal{I}$, the ACTL modalities are interpreted via the global transition relation $\mathcal{T}$, while the epistemic modality K_i is interpreted by the epistemic relation $\sim_i$ for agent i:

Definition 6 (Satisfaction). *Let $\mathcal{I}$ be an interpreted system over the set Ag of agents and the set AP of propositions, let ϕ be an ACTLK formula over Ag and AP, and let $g \in G$ be a reachable state. Truth of ϕ at g in $\mathcal{I}$, written $(\mathcal{I}, g) \models \phi$, is defined inductively by the following conditions:*

- $(\mathcal{I}, g) \models \alpha$ *iff* $\alpha \in V(g)$, *for* $\alpha \in AP$;
- $(\mathcal{I}, g) \models \neg\phi$ *iff* $(\mathcal{I}, g) \not\models \phi$;
- $(\mathcal{I}, g) \models \phi \wedge \psi$ *iff* $(\mathcal{I}, g) \models \phi$ *and* $(\mathcal{I}, g) \models \psi$;
- $(\mathcal{I}, g) \models K_i\phi$ *iff* $(\mathcal{I}, g') \models \phi$ *for all* $g' \in G$ *such that* $g \sim_i g'$;
- $(\mathcal{I}, g) \models AX\phi$ *iff for every path* $\pi = g^0g^1 \ldots$ *in* $\mathcal{I}$ *such that* $g = g^0$, *we have* $(\mathcal{I}, \pi(1)) \models \phi$;
- $(\mathcal{I}, g) \models A(\phi U\psi)$ *iff for every path* $\pi = g^0g^1 \ldots$ *in* $\mathcal{I}$ *such that* $g = g^0$, *there exists* $k \geq 0$ *such that* $(\mathcal{I}, \pi(k)) \models \psi$ *and* $(\mathcal{I}, \pi(j)) \models \phi$ *for all* $0 \leq j < k$;
- $(\mathcal{I}, g) \models A(\phi R\phi')$ *iff for every* i *and every path* $g^0, g^1, \ldots$ *in* $\mathcal{I}$ *such that* $g = g^0$, *if for all* $0 \leq j < i$, $(\mathcal{I}, \pi(j)) \not\models \phi$ *then* $(\mathcal{I}, \pi(i)) \models \phi'$.

Formula ϕ is true in $\mathcal{I}$, denoted by $\mathcal{I} \models \phi$, iff $(\mathcal{I}, g) \models \phi$ for all $g \in I_0$.

The abstraction technique [2] involves converting a ground, or *concrete* system, into an *abstract* system, typically smaller than the original. The abstract system is obtained by partitioning the system states into equivalence classes. Each equivalence class is represented by an *abstract state* in the abstract system. Every transition in the concrete system has a corresponding one in the abstract system; so every behavior of the concrete system is also a behavior of the abstract system. In [5] a quotient construction was defined.

Definition 7 (Quotient of interpreted system [5]). *Assume an interpreted system $\mathcal{I}$ over the set Ag of agents, the environment e, and the set AP of atomic propositions. For each $i \in Ag$, assume an equivalence $\equiv_i \subseteq L_i \times L_i$ and an equivalence $\equiv_i^a \subseteq ACT_i \times ACT_i$. For $l \in L_i$, we write $[l]$ for the equivalence class of l with respect to $\equiv_i$. Similarly, we write $[a_i]$ for the equivalence class of $a_i \in ACT_i$ with respect to $\equiv_i^a$. Likewise we define $\equiv_e$, $\equiv_e^a$, and equivalence classes on L_e and ACT_e. We write $[g]$ for $\langle [g_1], \ldots, [g_n], [g_e]\rangle$ and write $[\overline{a}]$ for $\langle [a_1], \ldots, [a_n], [a_e]\rangle$. Let $AP' \subseteq AP$ consist of all propositions of AP that do*

not distinguish between equivalent local states, i.e., all $\alpha \in AP$ *such that for all* $g, g' \in S$*: if* $\alpha \in V(g)$ *and* $g_i \equiv g'_i$ *for all* $i \in Ag$*, as well as* $g_e \equiv g'_e$*, then* $\alpha \in V(g')$*. The quotient system of* $\mathcal{I}$ *is the interpreted system* $\mathcal{I}'$ *over the set* Ag *of agents, the environment* e *and the set* AP' *of proposition such that:*

- $L'_i = \{[l] \mid l \in L_i\}$ *for all* $i \in Ag$*, and* $L'_e = \{[l] \mid l \in L_e\}$.
- $ACT'_i = \{[a] \mid a \in ACT_i\}$ *for all* $i \in Ag$*, and* $ACT'_e = \{[a] \mid a \in ACT_e\}$.
- $P'_i = \{\langle [l], [a] \rangle \mid \langle l, a \rangle \in P_i\}$ *for all* $i \in Ag$*, and* $P'_e = \{\langle [l], [a] \rangle \mid \langle l, a \rangle \in P_e\}$.
- $t'_i = \{\langle [l], [\overline{a}], [l'] \rangle \mid \langle l, \overline{a}, l' \rangle \in t_i\}$ *for all* $i \in Ag$*, and* $t'_e = \{\langle [l], [\overline{a}], [l'] \rangle \mid \langle l, \overline{a}, l' \rangle \in t_e\}$.
- $I'_0 = \{[g] \mid g \in I_0\}$.
- $V'([g]) = V(g) \cap AP'$.

It has been proved in [5] that the construction above preserves satisfaction from abstract to concrete models.

Theorem 1 (Preservation [5]). *Let* $\mathcal{I}'$ *be a quotient of interpreted system* $\mathcal{I}$*. For any ACTLK formula* ϕ *over* AP'*, if* $\mathcal{I}' \models \phi$*, then* $\mathcal{I} \models \phi$*.*

3 Implementation and Data Abstraction Theorem

Definition 7 and Theorem 1 do not give a constructive way for building the abstract model. For any implementation purposes we need to give an algorithm for defining appropriate equivalence relations. In the following we give such procedure in the case of ISPL files, the input to the model checker MCMAS [15]. We operate on ISPL files as they provide a natural operational correspondence to interpreted systems.

In a nutshell an ISPL program P defines local states, actions, protocols, and local transition for agents and environment corresponding to a given interpreted system. Local states for the agents are defined by means of a finite set $V = \{v_1, \ldots, v_m\}$ of variables. Each variable $v_k \in V$ has an associated finite domain D_k. We consider the set $\{+, -, \div, \cdot\}$ denoting standard arithmetic operations. We also use *binary relation symbols* from the set $\{<, >, =, \leq, \geq\}$. An *arithmetic expression* is built from variables in V, constants in D_k and arithmetic operations; for instance, $v_2 - 5$ is an arithmetic expression. A *logic expression* p is built from arithmetic expressions and relation symbols as natural; for instance, $v_2 - 5 > 4$ is a logic expression. A *Boolean expression* ψ is composed from logic expressions p using negation $\neg$, conjunction $\wedge$ and disjunction $\vee$. Any global state g can be seen as an evaluation over V, i.e., $g = (d_1, \ldots, d_m) \in D = D_1 \times \cdots \times D_m$. Similarly, the local states of an agent can be seen as evaluations over a subset of V, named *local variables* of the agent. We proceed by giving an example to explain the details of the abstraction procedure on the data of the program.

Card Game Example [9,5]. *The system has two agents Player1 and Player2 and an environment* e*. There is a deck of* $2N$ *cards. Each player receives* $N-1$ *cards. Two cards are put aside. Higher index cards beat lower index cards. In each round of the game, each player plays a card from his or her hand. The player playing*

the stronger card wins the round. The game continues until all cards have been played. The player who won the most number of rounds wins the game.

An ISPL program for this example is described as follows. Let $\mathcal{C} = \{1, \ldots, 2N\}$ represent the set of $2N$ cards. We call *red cards* the subset $\{N+1, \ldots, 2N\}$ and the remaining cards *black cards*. A player $i \in \{1, 2\}$ can either play a card or do nothing: $ACT_i = \{playcard\, c_i^k \mid c_i^k \in \mathcal{C}\} \cup \{nothing\}$. The environment either calculates who wins the current round or does nothing: $ACT_e = \{eval, nothing\}$. The local state of an agent describes what cards he or she holds and how many rounds he has played so far, as well as the outcome of the game: $L_i = \{(\mathcal{H}_i, k, a, b) \mid |\mathcal{H}_i| + 1 = |\mathcal{N}|\}$, where $\mathcal{H}_i \subset \mathcal{C}$ represents the cards held by the agent i and $\mathcal{N} = \{1, \ldots, N\}$ represents the game rounds. The environment records the current score in its local state, whose domain is $L_e = \{(\mathcal{H}_1, \mathcal{H}_2, a, b) \mid a + b \leq N - 1\}$ where a and b encode the number of deals won by player 1 and 2, respectively. The local protocols are defined as follows.

$$
\begin{aligned}
P_i(\mathcal{H}_i, k, a, b) &= \{playcard\, c_i^k \mid c_i^k \in \mathcal{H}_i \text{ and } k \in \mathcal{N}\}, \text{ if } k < N;\\
P_i(\mathcal{H}_i, k, a, b) &= \{nothing\}, \text{ if } k = N;\\
P_e(\mathcal{H}_1, \mathcal{H}_2, a, b) &= \{eval\}, \text{ if } a + b < N - 1;\\
P_e(\mathcal{H}_1, \mathcal{H}_2, a, b) &= \{nothing\}, \text{ if } a + b = N - 1.
\end{aligned}
$$

The local evolution functions have the form:

$$
\begin{aligned}
t_i((\mathcal{H}_i, k, a, b), \langle nothing\rangle) &= (\mathcal{H}_i, k, a, b); && (1)\\
t_i((\mathcal{H}_i, k, a, b), \langle playcard\, c_i^k\rangle) &= (\mathcal{H}_i, k+1, a', b'), && (2)\\
t_e((\mathcal{H}_1, \mathcal{H}_2, a, b), \langle playcard\, c_1^k, playcard\, c_2^k, eval\rangle) &= (a+1, b), \text{ if } c_1^k > c_2^k; && (3)\\
t_e((\mathcal{H}_1, \mathcal{H}_2, a, b), \langle playcard\, c_1^k, playcard\, c_2^k, eval\rangle) &= (a, b+1), \text{ if } c_1^k < c_2^k; && (4)\\
t_e((\mathcal{H}_1, \mathcal{H}_2, a, b), \langle nothing\rangle) &= (a, b). && (5)
\end{aligned}
$$

where a' and b' in (2) are the new updated values of a and b, according to (3) and (4).

The set of initial states is: $I_0 = \{(\mathcal{H}_1, 0, 0, 0), (\mathcal{H}_2, 0, 0, 0), (\mathcal{H}_1, \mathcal{H}_2, 0, 0)\}$. The atomic propositions we consider are $allred_i$ ("Player i holds only red cards."), win_i ("Player i has won the game."), $topred_i$ and $lowred_i$ for $i \in \{1, 2\}$.

$$
\begin{aligned}
allred_i &\text{ holds where } \mathcal{H}_i \subset \{N+1, \ldots, 2N\};\\
win_1 &\text{ holds where } a > b \text{ and } a + b = N - 1;\\
win_2 &\text{ holds where } b > a \text{ and } a + b = N - 1;\\
topred_i &\text{ holds where } \mathcal{H}_i = \{N+2, ..., 2N\};\\
lowred_i &\text{ holds where } \mathcal{H}_i \subset \{N, ..., 2N\}.
\end{aligned}
$$

Fig 1 shows an ISPL program encoding the example above in the case of $N = 3$, where $\mathcal{H}_1$ is described by the variables c11 and c12, and $\mathcal{H}_2$ by c21 and c22. In this case, $L_e = \{(\mathcal{H}_1, \mathcal{H}_2, a, b) \mid a + b \leq 2\}$, i.e., there are just two rounds. In the first round, the players play the cards $c11$, $c21$; in the second round, they play

```
Agent Environment
 Obsvars:
  a: 0 .. 2;
  b: 0 .. 2;
 end Obsvars
 Vars:
  c11: 1 .. 6;
  c12: 1 .. 6;
  c21: 1 .. 6;
  c22: 1 .. 6;
 end Vars
 --Actions and Protocol are omitted
 Evolution
  a=a+1 if c11>c21 and ...
  b=b+1 if c11<c21 and ...
 --the rest of the Evolution is omitted
end Agent

Agent Player1
 Lobsvars={c11,c12};
 Vars:
  n: 1 .. 3;
 end Vars
 Actions = {null,playcard1,playcard2};
 Protocol:
  n=1: { playcard1 };
  n=2: { playcard2 };
  n=3: { null };
 end Protocol
 Evolution:
  n = 2  if n=1;
  n = 3  if n=2;
 end Evolution
end Agent

-- Agent Player2 omitted

Evaluation
 lowred1  if (c11>2 and c12>2);
 topred1  if (c11>4 and c12>4);
 allred1  if (c11>3 and c12>3);
 win1 if (a>b and a+b=2);
 --The same properties
 --for Player2 are omitted
end Evaluation

--InitStates omitted

Formulae
 (AG(allred1->K(Player1,(AF win1))));
end Formulae
```

Fig. 1. Sketch of an ISPL program for the *Card Game* with 6 cards ($N = 3$)

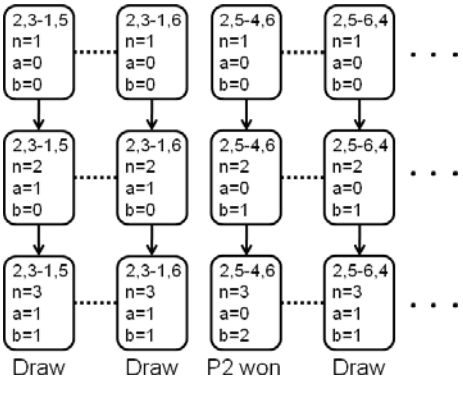

Fig. 2. Sketch of the global transition relation for the concrete *Card Game Example* with 6 cards. The dashed line represents the epistemic relation $\sim_1$.

$c21$, $c22$. Note from Fig 1 that *Player*1 has a special set called *Lobsvars*. This represents the environment variables accessible, i.e., visible, to the agent. This makes the program more succinct.

Fig 2 illustrates four possible paths in the model representing a run of the game. The notation "$i, j - k, t$" in Fig 2 means: $c11 = i$, $c12 = j$, $c21 = k$, $c22 = t$ for $i, j, k, y \in \{1, 2, 3, 4, 5, 6\}$.

Implementation. We now describe the procedure of the tool performing data abstraction on ISPL programs by partitioning the domain of variables. This extension takes an ISPL program for an interpreted system $\mathcal{I}$ as input and returns an ISPL program denoting a quotient interpreted system $\mathcal{I}'$ for $\mathcal{I}$ according to Definition 7. Briefly, the tool builds a new set of abstract reachable states G' from G by constructing new domains for the variables in $\mathcal{I}$. The procedure consists of four key steps. All steps are executed automatically.

Algorithm 1. The generation of the Boolean four-dimension Q.

for all $i \in Ag$ **do**
  for all $\alpha \in AP$ **do**
    for all p in precondition of α **do**
      if $var(p) \in i$ **then** $LE_i \Leftarrow LE_i \cup p$; **end if**
    end for
  end for
end for
$Q(i, v, p, d_t) \Leftarrow 0$;
for all $i \in Ag$ **do**
  for all local variable v of agent i **do**
    for all $p \in LE_i$ **do**
      for all $d_t \in D_v$ **do**
        if $d_t \models p$ **then** $Q(i, v, p, d_t) \Leftarrow True$; **else** $Q(i, v, p, d_t) \Leftarrow False$; **end if**
      end for
    end for
  end for
end for

1: Building the set LE$_i$. In an ISPL program P each atomic proposition $\alpha \in AP$ is defined by a boolean expression over variables defining on which global states α holds, thereby implementing the evaluation V. For each agent i, the tool builds a set LE_i of logic expressions containing local variables for agent i that appear in the definition of any $\alpha \in AP$. In the end, the tool builds LE by the union of all LE_i for all $i \in Ag$.

For example, for the card game reported in Fig 1, we have: $D_{c11} = D_{c12} = D_{c21} = D_{c22} = \{1, 2, 3, 4, 5, 6\}$, $V_e = \{c11, c12, c21, c22, a, b\}$, $V_{P_1} = \{n\}$, $V_{P_2} = \{n\}$. Notice that in V_{P_1} and V_{P_2} the other four variables are not present: $c11$, $c12$, a, b for P_1 and $c21$, $c22$, a, b for P_2. Those variable are not inserted in V_{P_1} and V_{P_2} by the procedure since all cij are local observable variables (*Lobsvars*) and a, b are global observable variables (*Obsvars*). Local observable variables of an agent i are those variables belonging to the environment agent that can be "seen" by the agent i. Therefore, the agent i knows the values of those variables at every moment and those variables contribute to form the local state of the agent i. However, the agent i cannot change the value of a local observable variable. Observable variables have the same characteristics of the local observable ones, but they can be seen by all agents indifferently.

Now, the abstraction tool automatically builds the sets $LE_e = \{c11 > 2, c12 > 2, c21 > 2, c22 > 2, c11 > 3, c12 > 3, c21 > 3, c22 > 3, c11 > 4, c12 > 4, c21 > 4, c22 > 4\}$, $LE_{P_1} = \emptyset$ and $LE_{P_2} = \emptyset$ from the logic expressions found in the *Evaluation* section of the ISPL-file. Note the following.

- Local variables from different agents cannot appear in the same logic expression.
- If a logic expression contains more than one variable, we rewrite it as a Boolean expression where each logic expression contains exactly one variable.
- If a logic expression contains the "not" connective, we rewrite it as an equivalent Boolean expression where the "not" connective does not appear.
- We cannot collapse values for variables that are updated by an arithmetic expression. This is because we may have transitions that are present in the original model but not present in the abstract one. Therefore, for the card game we cannot collapse a and b as they are updated in the Evolution by the arithmetic expressions $a = a + 1$ and $b = b + 1$ respectively (see Fig.1).

2: Generating the four-dimension vector Q. The tool automatically builds a four-dimension vector Q of Boolean values. The first dimension, i, of Q represents the agents; the second dimension, j, encodes the variables of a given agent; the third one, k, represents all logic expressions in which the current variable appears; the last one, t, represents the values d_t of the current variables. The vector Q encodes whether a logic expression p is evaluated to true when all free occurrences of the current variables are replaced by d_t, denoted by $d_t \models p$. From the vector Q we build new domains of abstract variables by collapsing values of every concrete variable that satisfies the same set of logic expressions. Algorithm 1 presents the generation of Q, where *var* is a function that returns the variable contained in the given logic expression.

3: Generating the abstraction functions by defining value clusters. The tool automatically builds a set of abstraction functions $\rho_1, \ldots, \rho_n$, where each ρ_i is defined on D_i, the domain of variable v_i. Given two local states $\bar{d} = (d_1, \ldots, d_m)$ and $\bar{e} = (e_1, \ldots, e_m)$ of agent i, we define the component abstraction functions ρ_i in the same way as defined in [2], i.e.:

$$\rho_i(d_1, \ldots, d_m) = \rho_i(e_1, \ldots, e_m) \quad \text{iff} \bigwedge_{p \in LE_i} (d_1, \ldots, d_m) \models p \Leftrightarrow (e_1, \ldots, e_m) \models p \qquad (6)$$

In other words, two tuples of values $\bar{e}, \bar{d}$ are in the same equivalence class if they cannot be distinguished by the same subset of logic expressions. Therefore, the particular $\equiv_i$ is automatically chosen according to formula (6).

The new values for the card game example (shown in bold fonts) are reported in Fig 3. Some instances of variables $c11$, $c12$, $c21$, $c22$ are collapsed into new values in the following way: $\mathbf{1} = \{1, 2\}$, $\mathbf{2} = \{3\}$, $\mathbf{3} = \{4\}$, $\mathbf{4} = \{5, 6\}$, because of $1, 2 \models \{\emptyset\}$, $3 \models \{cij > 2\}$, $4 \models \{cij > 2, cij > 3\}$, $5, 6 \models \{cij > 2, cij > 3, cij > 4\}$, where $i, j \in \{1, 2\}$. The partitioning of variable domains is done automatically.

4: Generating new domains for the abstract ispl-file P'. New values of the variables for the ISPL-file P' representing the abstract model are calculated from

Algorithm 2. The generation of new Domains. Line 8 shows how the partitioning of variable domains is calculated.

```
 1: for all i ∈ Ag do
 2:   for all local variable v of agent i do
 3:     for all d_t ∈ D_v do
 4:       indx_p ⇐ 0; newval ⇐ 0;
 5:       for all p ∈ LE_i do
 6:         if var(p) ∈ i then
 7:           if Q(i, v, d_t, p) = true then newval ⇐ newval + 2^indx_p; end if
 8:         end if
 9:         indx_p ⇐ indx_p + 1;
10:       end for
11:       D'_v ⇐ D'_v ∪ {newval};
12:     end for
13:   end for
14: end for
```

the old ones by taking into account what formulas the old values satisfy. Each formula is identified by an index ($indx_p$). These indexes are used to calculate an integer number (see line 8 of Algorithm 2) that will be the corresponding new value. Line 8 of Algorithm 2 shows how the partitioning of values is calculated. Values that satisfy the same set of formulas will get the same integer. The abstract ISPL-file P' is generated by substituting in every logic expression $p \in P$ the corresponding new value in D'. Notice from Fig 3 that the new domains of the abstract model in the example become $D'_{cij} = \{\mathbf{0}, \mathbf{1}, \mathbf{3}, \mathbf{7}\}$. The concrete values 1 and 2 are collapsed into the abstract value **0** as they do not satisfy any atomic proposition in LE_e. The concrete value 3 becames the abstract value $\mathbf{1} = 2^0$ since it satisfies formula $cij > 2$ and this formula gets the index 0. This index is used in the power 2^0 to calculates the new value **1**. The concrete value 4 becomes **3** since it satisfies formulas $cij > 2$ and $cij > 3$. Those formulas have the index 0 and 1 respectively. Therefore, the abstract value **3** is the result of the following calculation: $\mathbf{3} = 2^0 + 2^1$. Finally, the abstract value **7** represents the concrete values 5 and 6 that satify formulas $cij > 2$, $cij > 3$ and $cij > 4$. Those formulas have index 0, 1 and 2 respectively, therefore: $2^0 + 2^1 + 2^2 = \mathbf{7}$. The new abstract domain $D'_{cij} = \{\mathbf{0}, \mathbf{1}, \mathbf{3}, \mathbf{7}\}$ is "flattened" by an extra procedure that transforms D'_{cij} into $D''_{cij} = \{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}$, where **0**, **1**, **3**, **7** of D'_{cij} correspond to **1**, **2**, **3**, **4** of D''_{cij} respectively.

In this case, the abstraction process has reduced the number of reachable states from $\frac{6!}{2!} = 360$ different initial card combinations to $4! = 24$ in the new program.

As expected, the abstraction process might synthesise behaviours not present in the original model. For instance, let us analyse the following program lines describing part of the environment's evolution.

```
a=a+1 if c11>c21
b=b+1 if c11<c21
```

```
Agent Environment
 --Obsvars are not changed
 Vars:
  c11: 1 .. 4;
  c12: 1 .. 4;
  c21: 1 .. 4;
  c22: 1 .. 4;
 end Vars
 --Actions,Protocol and Evolution omitted
end Agent

Agent Player1
 Lobsvars={c11,c12};
 Vars:
  n: 1 .. 3;
 end Vars
 Actions = {null,playcard1,playcard2};
 Protocol:
  n=1: { playcard1 };
  n=2: { playcard2 };
  n=3: { null };
 end Protocol
 Evolution:
  n = 2  if n=1;
  n = 3  if n=2;
 end Evolution
end Agent

-- Agent Player2 omitted

Evaluation
 lowred1  if (c11>1 and c12>1);
 topred1  if (c11>3 and c12>3);
 allred1 if (c11>2 and c12>2);
 win1 if (a>b and a+b=2);
 --The same properties
 --for Player2 are omitted
end Evaluation

--InitStates omitted

Formulae
 (AG(allred1->K(Player1,(AF win1))));
end Formulae
```

Fig. 3. Sketch of an ISPL-file for the abstract *Card Game* with 6 cards

Those lines are transformed in the following:

```
a=a+1  if (c11=6  and c21=5)  or  (c11=6  and c21=4)  or ...
b=b+1  if (c11=5  and c21=6)  or  (c11=4  and c21=6)  or ...
```

Following the abstraction process, the program for the abstract model includes the following non-determinism:

```
a=a+1 if (c11=4 and c11=4)  or ...
b=b+1 if (c11=4 and c21=4)  or ...
```

Fig 4 illustrates execution branches not existing in the original model. This phenomenon can cause *false negatives*, i.e., the property checked results to be false in the abstract model while it is true in the original one.

Still, it can be checked that validity is preserved under the construction above.

Theorem 2 (Data Abstraction Theorem). *Given an ISPL-program P describing an interpreted system $\mathcal{I}$ and given a specification ϕ of the logic ACTLK, let P' be the ISPL program, generated from P by the procedure presented above. P' describes an interpreted system $\mathcal{I}'$. We have that specification ϕ holds in $\mathcal{I}$ if ϕ holds in $\mathcal{I}'$, i.e.,*

$$\mathcal{I}' \models \phi \quad \Longrightarrow \quad \mathcal{I} \models \phi.$$

Proof. We have to show that abstraction algorithm generates an ISPL-code P' that describes an interpreted system $\mathcal{I}'$ that is a quotient *one of the interpreted system $\mathcal{I}$ described by the original ISPL-code P*

By Theorem 1, we only need to prove that $\mathcal{I}'$ is a quotient of $\mathcal{I}$ according to Definition 7.

1. *L'_i is generated by the abstraction function ρ_i such that $L'_i = \{[l] \mid l \in L_i\}$, which is a partition of the set L_i. So, we have proved point 1 of the definition 7.*

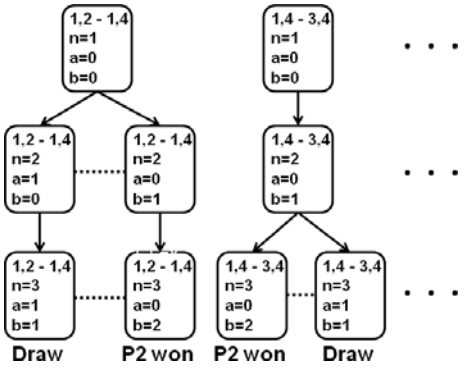

Fig. 4. Sketch of the global transition relation for the abstract *Card Game Example* with 6 cards. Notice how some paths split. This splitting causes new behaviours in the abstract model.

2. $ACT'_i = ACT_i$ *(i.e.,* $[a] \equiv a$*) because* ρ_i *does not partition the set of actions.*
3. $P' = \{\langle [l], a\rangle \mid \langle l, a\rangle \in P\}$ *because the abstraction technique simply replaces the old values of a variable with the new ones.*
4. *Note that* $[l] \neq [l']$ *(*$[l], [l'] \in L'_i$*) if and only if there is at least one* $l \in [l]$ *and one* $l' \in [l']$ *in* L_i *such that* $l \neq l'$*.*
   Now, we have to show that if they are connected in the concrete system, then they are also connected in the abstract one. Formally, we have to show the following condition holds:

$$\forall l, l' (\ \rho(l) = [l] \wedge \rho(l') = [l'] \wedge lRl' \ \Rightarrow\ [l]\hat{R}[l']\ )$$

   Suppose $\rho(l) = [l] \wedge \rho(l') = [l'] \wedge lRl'$*, we show* $[l]\hat{R}[l']$*. By assuming* lRl'*, we know* $\exists \bar{a} \in Act\ \forall i \in Ag : \langle l, \bar{a}, l'\rangle \in t_i$ *and* $\langle l, a\rangle \in P_i$ *by definition. Since the abstraction technique performs a partition of states, we have, for all agents* $i \in Ag$ *and for all local states,* $[l] \neq [l']$*. As actions are not modified, we have* $\langle [l], [a]\rangle \in P'_i$*, where* $l \in [l]$ *and* $a \equiv [a]$*. Moreover, Since the local evolution function is rebuilt by substituting* l_i*, such that* $l \in [l]$*, with* $[l]$*, we have* $\langle [l], \bar{a}, [l']\rangle \in t'_i$ *iff* $l, \bar{a}, l'\rangle \in t_i$*. Therefore, we have* $\langle [l], \bar{a}, [l']\rangle \in t'_i$ *and* $\langle [l], [a]\rangle \in P'_i$*, but that means* $[l]\hat{R}[l']$*.*
5. *Point 5 and 6 in Definition 7 can be proved trivially.* □

4 Experimental Results

We now report on the experimental results obtained by the implementation described in the previous section. Specifically, we comment on our findings for the card game example and a variant of the bit transmission problem. These scenarios are to be considered as a sanity check of the technique and not as real applications. We leave these for further work.

Number Transmission Protocol. This scenario is an extension of the well known *bit transmission problem* described in [11]. Let $Ag = \{S, R\}$ be the set of agents where S, R represent the *sender* and the *receiver* respectively. Moreover, there is the *environment* agent labelled with the letter E.

In this scenario, a number is sent from the sender to the receiver via a an unreliable channel modelled by the environment. Note that in the known bit transmission problem only one bit is sent. In this case the sender sends a number ranging from 1 to N. In the experiments below we used N in the set $D = \{0.5{\cdot}10^4, 10^4, 2 \cdot 10^4, 2.5 \cdot 10^4, 3 \cdot 10^4\}$.

We describe below the interpreted system for the protocol above. The Environment is described by the set of local states $L_E = \{S, R, RS, none\}$. The local state S represents the channel reliably sending messages from sender to receiver and dropping messages from receiver to sender. Conversely, R represents a situation where messages only travel from receiver to sender. RS encodes a situation where the channel is transmitting both directions, whereas when in *none* the channel loses all messages. The set of local states for the sender is $L_S = D \times \{true, false\}$ where $D = \{1, \ldots, N\}$ represents the domain of the integer that can be sent and $\{true, false\}$ is the domain of the variable *ack* keeping track of whether an acknowledgement has been received from the sender. We model the receiver by considering the set $L_R = \{received, notrec\}$; *notrec* represents a situation where the receiver has not yet received any message, while *received* encodes the fact that the receiver got a message. The environment can perform four actions: $ACT_E = \{S, SR, R, none\}$ representing which direction, if any, it is letting messages flow. The sender can either send the intended message or do nothing (*null*) $ACT_S = \{send\,N \mid N \in D\} \cup \{nothing\}$. Similarly, the receiver can either send an acknowledgement or remain silent: $ACT_R = \{sendack, nothing\}$. As in the original bit transmission problem we assume the sender keeps sending the same message until he or she receives an acknowledgement; the receiver remains silent before receiving the message from the sender; after that he repeatedly sends acknowledgements back to the sender. Consequently the protocols can be defined as follows.

$$\begin{aligned} P_E(S) &= \{S\}; & P_E(R) &= \{R\}; \\ P_E(SR) &= \{SR\}; & P_E(none) &= \{none\}; \\ P_S(N, false) &= \{send\,N\}; & P_S(N, true) &= \{nothing\}; \\ P_R(notrec) &= \{nothing\}; & P_R(received) &= \{sendack\}. \end{aligned}$$

Local evolution functions are defined as follow:

$$t_E(state, \langle a_E, a_S, a_R\rangle) = state'; \tag{7}$$

$$t_S((N, false), \langle SR, a_S, sendack\rangle) = (N, true); \tag{8}$$

$$t_S((N, false), \langle R, a_S, sendack\rangle) = (N, true); \tag{9}$$

$$t_R(notrec, \langle SR, send\,N, a_R\rangle) = received; \tag{10}$$

$$t_R(notrec, \langle S, send\,N, a_R\rangle) = received. \tag{11}$$

Where $state, state' \in L_E$ and $a_k \in ACT_k$, for $k \in Ag$. Note that by (7) the channel moves non-deterministically. In (8) and (9) the sender receives an acknowledgement from the receiver as the channel transmits messages from the receiver to the sender in both cases. Similarly, in (10) and (11) the receiver receives the number.

The set of initial states I_0 is as follows: $I_0 = \{(E.state, (N, false), notrec) \mid E.state \in L_E\}$. The evaluation for the proposition $numberN, recack, recNumber$ is the obvious one. Fig 5 represents a sketch of an ISPL-file corresponding to the interpreted system defined above for $N = 10000$. From Fig 5, in the *Evaluation* section, we want to know if the number sent was either exactly 1 or it was greater than 2500, 5000 or 7500. By running the abstraction toolkit to this file we obtain an abstract Number Transmission ISPL-file in which the Sender can only send 5 possible digits. This is of course the result we would expect. The system in Fig 5 corresponds to the second one listed in Table 2.

We tested both examples above on the abstraction toolkit paired with MCMAS against known specifications for the protocols. In particular we verified $AG(allred_1 \rightarrow K_{Player1}(AFwin1))$ in the case of the card game example (specifying that if player one has only red cards he knows he will win the game), and $AG((numberN \wedge recack) \rightarrow (K_S K_R number = N))$ for the number transmission protocol (specifying that once an ack has been received the sender knows that the receiver knows the value of the number transmitted).

```
Agent Environment
 Vars:
  state : {S,R,SR,none};
 end Vars
 Actions = {S,SR,R,none};
 Protocol:
  state=S: {S,SR,R,none};
  state=R: {S,SR,R,none};
  state=SR: {S,SR,R,none};
  state=none: {S,SR,R,none};
 end Protocol
-- Evolution Omitted
 end Agent

Agent Sender
 Vars:
  number : 0..10000;
  ack : boolean;
 end Vars
 Actions = { send,nothing };
 Protocol:
  ack=false : {send};
  ack=true : {nothing};
 end Protocol
 Evolution:
  (ack=true) if (ack=false)
  and (Receiver.Action=sendack)
  and ((Environment.Action=SR)
  or (Environment.Action=R));
 end Evolution
end Agent
```

```
Agent Receiver
 Vars:
  state : boolean;
 end Vars
 Actions = {nothing,sendack};
 Protocol:
  state=false : {nothing};
  state=true : {sendack};
 end Protocol
 Evolution:
  state=true if (Sender.Action=send)
  and (state=false)
  and ((Environment.Action=SR)
  or (Environment.Action=S));
 end Evolution
end Agent

Evaluation
 recNumber if ( (Receiver.state=true);
 recack if ( ( Sender.ack = true ) );
 N1 if  ( (Sender.number=1));
 N2500 if ( (Sender.number>2500) );
 N5000 if ( (Sender.number>5000) );
 N7500 if ( (Sender.number>7500) );
end Evaluation
InitStates
 !Sender.Number=0 and Receiver.state=false
 and Sender.ack=false and (Environment.state=S
 or Environment.state=R or Environment.state=SR
 or Environment.state=none ;
end InitStates
```

Fig. 5. Sketch of an ISPL-file for the number transmission protocol for $N = 10000$

Table 1. Verification results for the *Card Game*

Number of cards	With reduction			Without reduction		
	States	Time (s)	BDD (MB)	States	Time (s)	BDD (MB)
6	138	0	4.70	11316	0	4.67
8	22528	2	6.67	80640	4	15.27
10	135866	4	9.59	$2,167 \times 10^9$	867	66.71
12	762812	26	31.87	?	> 86400	?
14	3.877×10^6	106	41.68	?	> 86400	?

Table 2. Verification results for the *Number Transmission Problem*

Maximum number N	With reduction			Without reduction		
	States	Time (s)	BDD (MB)	States	Time (s)	BDD (MB)
5000	48	0	4.55	98292	11	5.72
10000	60	0	4.59	196596	47	6.58
15000	84	1	4.67	196596	118	7.12
20000	108	1	4.64	393204	216	8.24
25000	132	1	4.82	393204	350	8.62
30000	156	1	5.64	393204	485	8.56

The experiments were executed on a machine running Ubuntu 9.10 on an Intel Core 2 1.86GHz with 1GB memory. The results are reported in Table 1 and Table 2.

From Table 1 we can notice that as expected the implementation drastically reduces both memory and time for the verification process. In the case of 12 and 14 cards, MCMAS could not verify the specification in over 24hrs, while the abstract systems could be verified in seconds. It is perhaps less obvious that, from Table 2, in the transmission problem for the case of $N = 10000$ and $N = 15000$ the two systems have the same number of reachable states. This is because MCMAS uses 14 BDD variables to encode both 10000 and 15000 states and MCMAS does not remove redundant states, using 2^{14} BDD states in both cases. The same phenomenon occurs for $N = 20000$, $N = 25000$ and $N = 30000$. In this case, MCMAS uses 15 BDD variables.

5 Conclusions

In this paper we began to explore fully automatic abstraction techniques for multi-agent systems. The technique abstracts a multi-agent system, described by an ISPL program, by collapsing the local states for the agents. We showed that our technique builds an abstract system that simulates the concrete thereby guaranteeing the methodology is sound. We evaluated the technique on a card game example for several numbers of cards and on the number transmission protocol. The results produced point to a considerable, although expected, reduction in the verification time and memory.

In the future we intend to test the methodology on more complex cases and to implement a refinement procedure that can be used to refine the model upon receiving false negatives from MCMAS.

Acknowledgements. This research was partially funded by EPSRC under grant EP/E035655 "Verification of security protocols: a multi-agent systems approach".

References

1. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
2. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. 16(5), 1512–1542 (1994)
3. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
4. Cohen, M., Dam, M., Lomuscio, A., Qu, H.: A data symmetry reduction technique for temporal-epistemic logic. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 69–83. Springer, Heidelberg (2009)
5. Cohen, M., Dam, M., Lomuscio, A., Russo, F.: Abstraction in model checking multi-agent systems. In: AAMAS 2009 (2009)
6. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, pp. 238–252 (1977)
7. Dechesne, F., Orzan, S., Wang, Y.: Refinement of kripke models for dynamics. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigun, H. (eds.) ICTAC 2008. LNCS, vol. 5160, pp. 111–125. Springer, Heidelberg (2008)
8. Emerson, E.A., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Science of Computer Programming 2(3), 241–266 (1982)
9. van Ditmarsch, H., van der Hoek, W., van der Meyden, R., Ruan, J.: Model Checking Russian Cards. Electr. Notes Theor. Comput. Sci. 149(2), 105–123 (2006)
10. Enea, C., Dima, C.: Abstractions of multi-agent systems. In: Burkhard, H., Lindemann, G., Verbrugge, R., Varga, L.Z. (eds.) CEEMAS 2007. LNCS (LNAI), vol. 4696, pp. 11–21. Springer, Heidelberg (2007)
11. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge (1995)
12. Gammie, P., van der Meyden, R.: MCK: Model checking the logic of knowledge. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 479–483. Springer, Heidelberg (2004)
13. Graf, S., Saïdi, H.: Construction of abstract state graphs with pvs. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
14. Kacprzak, M., Nabialek, W., Niewiadomski, A., Penczek, W., Pólrola, A., Szreter, M., Wozna, B., Zbrzezny, A.: Verics 2007 - a model checker for knowledge and real-time. Fundamenta Informaticae 85(1-4), 313–328 (2008)

15. Lomuscio, A., Qu, H., Raimondi, F.: Mcmas: A model checker for the verification of multi-agent systems. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 682–688. Springer, Heidelberg (2009)
16. Penczek, W., Lomuscio, A.: Verifying epistemic properties of multi-agent systems via bounded model checking. Fundamenta Informaticae 55(2), 167–185 (2003)
17. Wooldridge, M.: Computationally grounded theories of agency. In: Durfee, E. (ed.) ICMAS, pp. 13–22. IEEE Press, Los Alamitos (2000)
18. Wooldridge, M.: An introduction to MultiAgent systems, 2nd edn. Wiley, Chichester (2009)