Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 195 (2008) 133–149
www.elsevier.com/locate/entcs

A Compositional Automata-based Approach for Model Checking Multi-Agent Systems

Mario Benevides[a,1,2], Carla Delgado[a,3], Carlos Pombo[b,4], Luis Lopes[a] and Ricardo Ribeiro[a]

[a] *COPPE-Sistemas*
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil

[b] *Universidad de Buenos Aires*
Buenos Aires, Argentina

Abstract

This paper addresses the issue of model checking knowledge in concurrent systems. The work benefits from many recent results on model checking and combined logics for time and knowledge, and focus on the way knowledge relations can be captured from automata-based system specifications. We present a formal language with compositional semantics and the corresponding Model Checking algorithms to model and verify Multi-Agent Systems (*MAS*) at the knowledge level, and a process for obtaining the global automaton for the concurrent system and the knowledge relations for each agent from a set of local automata that represents the behavior of each agent. Our aim is to describe a model suitable for model checking knowledge in a pre-defined way, but with the advantage that the knowledge relations for this would be extracted directly from the automata-based model.

Keywords: Epistemic Modal Logics, Model Check, Multi-Agent Systems.

1 Introduction

The growing relevance of interactive multi-agent based software drives attentions to formal modelling and verifying multi-agent rational interaction by means of computational methods. Model Checking is a powerful and mature technique for verifying finite state concurrent systems [4], and many efforts have been made to contemplate rational interaction aspects with model checking facilities.

[1] The authors acknowledge partial support from the Brazilian Research Agencies CNPq, FAPERJ and CAPES. The authors want to thank the University of Edinburgh

[2] Email: mario@cos.ufrj.br

[3] Email: cdelgado@microart.eu

[4] Email: clpombo@dc.uba.ar

doi:10.1016/j.entcs.2007.08.030

In particular, the work of Alur, Henzinger and Kupferman has generalized the basic model checking logic, branching-time temporal logic (CTL), to Alternating Time Temporal Logic (ATL) [1], replacing the path quantifiers of CTL by "cooperation modalities" that can be used to talk about the powers that groups of agents have to achieve certain results. ATL formulas are interpreted over game structures, which according to the authors allow to capture compositions of open systems [5].

Interested in capturing group interactive concepts like powers and strategies, the model for ATL is based on Game Theory concepts, specially in what concerns the path quantifiers used in the model checking process. The ATL formula $\langle\langle A\rangle\rangle\psi$ is satisfied at state q iff there is a winning strategy for the agents in set A to choose their actions in such a way that ψ holds in the successor state(s) of q that results from the execution of the actions chosen by agents in A no matter which are the actions executed by agents not in A.

Epistemic aspects of concurrent systems were studied in [9] and [7], where a multi-modal language and models incorporating time and knowledge for groups of agents were established. These models for knowledge are interpreted over Kripke Structures.

The interest to define and check knowledge properties of concurrent systems was explored in many recent works like [3], [6], and [5]. These initiatives, as many others, drive our attention to plausibility of automatic verification when epistemic aspects need to be considered.

In particular, [5] extends ATL with knowledge modalities. The extended language ATEL has the power to express properties about the relations of group powers and knowledge, an interesting feature for game-like multi-agent systems. But ATEL has not yet got a generally accepted semantics, due to the difficulties in coordinating knowledge modalities - originally established over stable systems - and the group action modalities from ATL - based on game-theory models and specially designed for open systems.

ATEL enriches ATL with knowledge modalities, and adds to ATL model epistemic accessibility relations $\sim_a$ for each agent a. The epistemic accessibility relations play an important role in the semantics of knowledge modalities, but are not used in the semantic definitions of path quantifiers and time modalities semantics. This is not reasonable because the existence of epistemic accessibility relations means uncertainty, and when uncertainty is involved on game theory models, the basic definition of strategies change, what would affect ATL semantics. To give ATEL a precise semantics, these changes on the model should be taken into account.

[12] presents an approach to model check knowledge and time in systems with perfect recall, with very interesting results. A combined logic of knowledge and linear time in synchronous systems with perfect recall is presented, and the semantics captures the notions of knowledge using observation functions that are defined in a way to enforce perfect recall.

This paper also addresses the issue of model checking knowledge in concurrent

[5] "An open system is a system that interacts with its environment and whose behavior depends on the state of the system as well as the behavior of the environment" [1].

systems. The work described here benefits from the results presented at recent papers on the area mentioned above, and focus on the way knowledge relations can be captured from automata-based system specifications. Our aim is to describe a model suitable for model checking knowledge in a (very basic) pre-defined way, but with the advantage that the knowledge relations for this would be extracted directly from the automata-based model. Doing so, we avoid the tedious task of identifying and formally stating the knowledge relations (or epistemic states) together with the system specification, what is also a way to keep a clean specification.

We present a formal language with compositional semantics and the corresponding Model Checking algorithms to model and verify Multi-Agent Systems (*MAS*) at the knowledge level. We extend branching time *CTL* logic with knowledge operators $\mathcal{K}_k$ for each agent k. The resulting language, which we call *KCTL*, provides the capability of observing the occurrence of an event from the point of view of one agent k.

[8] discusses unbounded model checking of a combined CTL and Knowledge logic and uses a SAT-based technique to improve efficiency of the model checking algorithms. The semantics is based on interpreted semantic systems and like in ATEL the epistemic accessibility relations $\sim_a$, for each agent a, are based on global states and cannot be inferred form the local agents representation.

Our choice of a CTL-like language allows a safe construction of a model for knowledge and time over Kripke Structures, as epistemic accessibility relations do not interfere with path quantifiers by definition. The disadvantage of such a choice is the loss of the possibility to directly talk about agents' powers in the language. But it is still possible to reason about what can be achieved by using the traditional path quantifiers of [4], as we may show in the examples to be presented.

A compositional semantics based on local automata for each agent is presented for *KCTL*, quite different from the one defined for ATEL [5]. ATEL semantics is based on a previously defined global automaton representing the whole system and the knowledge relations. For *KCTL* semantics, the global automaton for the concurrent system as well as knowledge relations for each agent are constructed based on a set of local automata.

Model Check algorithms to verify *KCTL* formulas over state transition systems (interactive *MAS*s representations) are defined and described in detail. The alternating bit protocol is used to exemplify the model checking process.

This paper is organized as follows. Section 2 reviews the branching-time temporal logic *CTL* [4] and the corresponding model checking process [11]. Section 3 states the requirements to reason about knowledge in *MAS*s. Our approach to handle knowledge in a concurrent system for Model Checking purposes comes in section 4, as we present the language *KCTL* with its semantics and state the algorithms for model checking. The final remarks are discussed on section 6.

2 *CTL* branching-time temporal logic and Model Checking

In [2], Ben-Ari et al. presented for the first time the logic of branching-time *CTL* with the aim of dealing with the set of every possible execution tree generated by a given program. This logic was specially designed to take care of the consequences of the non-determinism just like the one generated by programs that interact asynchronously.

It was in [4] where Emerson and Clarke gave the final shape to *CTL* providing a decision procedure, and that is the reason why the way we present the logic is close to that of the previously mentioned article.

2.1 CTL *Language*

Definition 2.1 Syntax of *CTL* formulas

Let $\mathcal{P}$ be a set of propositions. The language of *CTL* formulas is defined as follows:

$ForCTL(\mathcal{P})$ is the smallest set For of formulas such that:

- $p \in For$ iff $p \in \mathcal{P}$,
- $\{\neg\phi_1, \phi_1 \vee \phi_2, \exists \mathsf{X}\phi_1, \exists \mathsf{G}\phi, \exists[\phi_1 \mathsf{U} \phi_2]\} \subseteq For$ iff $\{\phi_1, \phi_2\} \subseteq For$.

The rest of the propositional operators are defined in terms of negation ("$\neg$") and disjunction ("$\vee$") in the usual way. Let $\phi, \psi \in ForCTL(\mathcal{P})$, then the rest of the temporal operators are defined as follows: $\exists \mathsf{F}\phi = \exists[\mathtt{true}\mathsf{U}\phi]$, $\forall \mathsf{X}\phi = \neg\exists\mathsf{X}\neg\phi$, $\forall \mathsf{G}\phi = \neg\exists\mathsf{F}\neg\phi$, $\forall[\phi \mathsf{U} \psi] = \neg\exists[\neg\psi \mathsf{U}(\neg\phi \wedge \neg\psi)] \wedge \neg\exists\mathsf{G}\neg\psi$ and $\forall \mathsf{F}\phi = \neg\exists\mathsf{G}\neg\phi$.

The intended meaning of *CTL* formulas is given as usual in terms of Kripke models.

Definition 2.2 Kripke model Let $\mathcal{P}$ be a set of propositions. Then $\mathfrak{M} = \langle S, S_0, R, \mathcal{P}, \mathcal{L}\rangle$ is said to be a Kripke model if it satisfies the following properties:

- S is a non-empty set of states,
- $S_0 \subseteq S$ and $S_0 \neq \emptyset$,
- $R \subseteq S \times S$ and $Dom(R) = S$ [6],
- $\mathcal{L} : S \to 2^{\mathcal{P}}$.

Definition 2.3 Set of runs of a Kripke model Let $\mathfrak{M} = \langle S, S_0, R, \mathcal{P}, \mathcal{L}\rangle$ be a Kripke model. Then, the runs of $\mathfrak{M}$, denoted by $\mathcal{R}^{\infty}_{\mathfrak{M}}$ are characterized as follows:

$$\mathcal{R}^{\infty}_{\mathfrak{M}} = \{\sigma \in seq^{\infty}(S) \mid \pi_1(\sigma) \in S_0 \wedge$$

$$(\forall i \in I\!N : \pi_i(\sigma)\ R\ \pi_{i+1}(\sigma))\}.$$ [7]

[6] *Dom* is the set-theoretical domain function, and this restriction states that every state in S has at least one successor through the accessibility relation R.

[7] We use $seq^{\infty}(S)$ to denote the set of infinite sequences of elements taken from the set S, and π_i as the projection of the i^{th} element of a sequence.

We will use $\mathcal{R}_{\mathfrak{M}}$ to denote the infinite set of finite prefixes of the sequences of $\mathcal{R}^{\infty}_{\mathfrak{M}}$.

Definition 2.4 Prefix of a run Let $\mathfrak{M} = \langle S, S_0, R, \mathcal{P}, \mathcal{L}\rangle$ be a Kripke model, $T \subseteq \mathcal{R}^{\infty}_{\mathfrak{M}}$, $\sigma \in \mathcal{R}^{\infty}_{\mathfrak{M}}$ and $i \in I\!N$, ${}_i\sigma$ will denote the prefix of length i of σ, defined as ${}_i\sigma = \sigma' \in seq(S) \mid Length(\sigma') = i \wedge (\forall j \in I\!N : 1 \leq j \leq i \implies \pi_j(\sigma') = \pi_j(\sigma))$.

Definition 2.5 Satisfiability relation for *CTL* formulas Let $\mathfrak{M} = \langle S, S_0, R, \mathcal{P}, \mathcal{L}\rangle$ be a Kripke model, the satisfiability relation is defined as follows:

$$
\begin{array}{lll}
\mathfrak{M}, \langle \sigma, i\rangle \models p & \text{iff} & p \in \mathcal{L}(\pi_i(\sigma)) \\
\mathfrak{M}, \langle \sigma, i\rangle \models \neg\phi & \text{iff} & \mathfrak{M}, \langle \sigma, i\rangle \not\models \phi \\
\mathfrak{M}, \langle \sigma, i\rangle \models \phi \vee \psi & \text{iff} & \mathfrak{M}, \langle \sigma, i\rangle \models \phi \text{ or} \\
 & & \mathfrak{M}, \langle \sigma, i\rangle \models \psi \\
\mathfrak{M}, \langle \sigma, i\rangle \models \exists \mathsf{X}\phi & \text{iff} & \exists \sigma' \in \mathcal{R}^{\infty}_{\mathfrak{M}} : {}_i\sigma' = {}_i\sigma \wedge \\
 & & \mathfrak{M}, \langle \sigma', i+1\rangle \models \phi \\
\mathfrak{M}, \langle \sigma, i\rangle \models \exists \mathsf{G}\phi & \text{iff} & \exists \sigma' \in \mathcal{R}^{\infty}_{\mathfrak{M}} : {}_i\sigma' = {}_i\sigma \wedge \\
 & & \forall j : i \leq j \implies \\
 & & \mathfrak{M}, \langle \sigma', j\rangle \models \phi \\
\mathfrak{M}, \langle \sigma, i\rangle \models \exists [\phi \mathsf{U} \psi] & \text{iff} & \exists \sigma' \in \mathcal{R}^{\infty}_{\mathfrak{M}} : {}_i\sigma' = {}_i\sigma \wedge \\
 & & (\exists j \in I\!N : i < j \wedge \\
 & & \mathfrak{M}, \langle \sigma', j\rangle \models \psi \wedge \\
 & & (\forall k \in I\!N : i \leq k < j \implies \\
 & & \mathfrak{M}, \langle \sigma', k\rangle \models \phi))
\end{array}
$$

2.2 CTL *Model Checking*

Given a Kripke Model $\mathfrak{M} = \langle S, S_0, R, \mathcal{P}, \mathcal{L}\rangle$ that represents a finite state concurrent system with its properties of interest and a *CTL* formula f expressing some desired specification, the *model checking problem* is to find the set of states in S that satisfy f [11]: $\{s \in S | \mathfrak{M}, s \models f\}$. In other words, the state-transition system underlying a Kripke structure is checked to see whether it is a model of the specification written in *CTL*.

Normally some states are designated initial states, and we say that the system satisfies the specification provided that all of the initial states are in the set. Formally, $\mathfrak{M}, S_0 \models f$ means $\forall_{s_0 \in S_0} \mathfrak{M}, s_0 \models f$.

A *CTL* formula f can be identified with a set of states in a given model $\mathfrak{M}$, namely those states $Q_{\mathfrak{M}} \subseteq S$ that satisfy the formula: $Q_{\mathfrak{M}}(f) = \{s | \mathfrak{M}, s \models f\}$. Model checking a *CTL* formula therefore entails the manipulation of sets of states: $S_0 \subseteq Q_{\mathfrak{M}}(f)$. Algorithms for doing so are given in [11].

3 Knowledge in Concurrent Systems

For the purpose of this work, we consider a Distributed System a concurrent system composed of a set of agents, each running its corresponding program, that communicate by sending and receiving messages along previously defined communication channels (a formal model for concurrent systems with this characteristics will be given at 4.1).

The interesting fact about this model is that it allows us to talk in separate about local and global computation. An agent is not concerned about the way other agents carry on their local computations. All interaction happens by sending and receiving messages.

This approach is very suggestive to talk about rational agents at the knowledge level. Each agent has its part of local knowledge and uncertainty and, as interaction takes place, this knowledge can be changed by gaining new information and refining uncertainties.

3.1 Multi-Agent Kripke Models for Knowledge

The usual way to deal with knowledge and uncertainty in Kripke Models is by means of "indistinguishable states", presented in [9] and [7].

To accomplish the notion that each agent has its own private information set, we label propositions with its corresponding agent identification, assigning propositions to one agent information set.

We also enhance the Kripke Model with epistemic accessibility relations $\sim_k$. For each agent k, we define an equivalence relation $\sim_k$ over S.

Definition 3.1 K-extended Kripke Model Let $\{\mathcal{P}_k\}_{1\leq k\leq j}$ be a set of disjoint sets of propositions. Then $\mathfrak{M} = \langle S, S_0, R, \{\sim_k\}_{1\leq k\leq j}, \cup_{k=1}^{j}\mathcal{P}_k, \mathcal{L}\rangle$ is said to be a K-extended Kripke Model if it satisfies the following properties:

- S, S_0 and R are as defined for a Kripke Model;
- $\mathcal{L} : S \to 2^{\cup_{k=1}^{j}\mathcal{P}_k}$.
- $\{\sim_k\}_{1\leq k\leq j}$ is a set of binary equivalence relations on S.

Intuitively, two states s and t are related by $\sim_k$ if agent k, being at state s, can't tell if the current state is either s or t. In other words, he can't distinguish the two possible situations: "the current state is s" and "the current state is t".

3.2 A Language for Knowledge

To reason about knowledge in a *MAS* it is necessary to assume that agents are able to reason about the world and also about other agents' knowledge. A complete axiomatic characterization of the notion of knowledge and common knowledge, and an accurate analysis of the role played by time in *MAS*' evolution was given in [9].

We adopt a propositional multi-modal language, with a knowledge modality $\mathcal{K}_k$ for each agent k. Knowledge modalities permit to talk about information from each

agent's point of view. Intuitively, formula $\mathcal{K}_k\varphi$ indicates that "agent k knows φ".

The semantics of the knowledge modalities $\mathcal{K}_k$ is based on the epistemic accessibility relations $\sim_k$. Two states s and t of S are related by $\sim_k$ if and only if agent k cannot distinguish them.

We say that an agent "knows" a fact ϕ in a state s if and only if ϕ is the case in all states (or worlds, following the common terminology) he/she considers possible at state s.

Given a state s of a Kripke Model $\mathfrak{M} = \langle S, S_0, R, \{\sim_k\}_{1\leq k\leq j}, \cup_{k=1}^{j}\mathcal{P}_k, \mathcal{L}\rangle$:
$\mathfrak{M}, s \models \mathcal{K}_k(\phi)$ iff $\forall s' \in S | s' \sim_k s \implies \mathfrak{M}, s' \models \phi$

4 Model Checking Knowledge in Multi-Agent Systems

We now present a formal language and the corresponding Model Check process to verify multi-agent systems at the knowledge level. The language proposed is an extension of *CTL*, which we call *KCTL*. After defining *KCTL*, we present Algorithms for checking its semantics in a branching time model.

4.1 Multi-agent architecture

Concurrent systems have complex behaviors. Reasoning about them requires a clear and flexible model. For the purpose of reasoning about knowledge, the model should capture the interactions and information from the perspective of each agent or component, as much as the conjunct behavior.

We consider that each agent has it's own behavior, dictated by a local program. The behavior (or computation) in the Concurrent System is the result of the interaction of it's constituents agents, and so, dictated by the interactions of the local program that each agent runs.

[7] presents a classic event-based model for synchronous *MAS*. The model is basically composed by:

- a network with m agents, connected by communication channels;
- a set R of synchronous runs (distributed computations or parallel runs of all agents involved, dictated by a global clock);
- a set E of events, including internal actions and communication events;
- a set C of global states of the system; and
- a protocol P (or distributed algorithm) corresponding to a set of local programs that specifies the behavior of each agent.

We refer to [10] for an abstraction for representing concurrent and distributed data processing systems. The style of model is operational (rather than axiomatic), what we consider more adequate for model checking purposes. It is based on a simple underlying automaton model for concurrent systems.

So, we represent each agent's behavior (each program in protocol P) as an automaton. Then, we compose the automata corresponding to all agents by identifying actions in their interfaces, thus modelling concurrent systems built from previously

modelled components. Composition yields a single automaton that is a model for the whole system - whose behavior corresponds to the parallel synchronous execution of all agents.

We can easily map the other components of [7] event-based model into the automaton model we have just mentioned and that we will be refine during this section. Each automaton i represents the local states (nodes) and events from E (edges) for an agent, and has a particular set of indexed propositions $p_{j\,j\in\mathcal{I}}$, where $\mathcal{I}$ is an index set for the automaton i, that states relevant properties about the computation carried on by i.

We stress that the automaton for an agent holds the agent's behavior and so all the information the agent can reason about and react to should appear there. The reason to index the propositions is then to avoid that two different automata use the same proposition, possibly with different meanings, what would lead to confusion by the time the composed automaton for the whole system is generated.

The actions of an automaton are classified as input (ones that represent events that are caused by the environment, such as the receipt of a message), output (that an agent can perform and that affect the environment, such as sending a message) and internal (that an agent can perform, but are undetectable to the environment except through their effects on later output events, such as changing the value of a local variable). The automaton generates output and internal actions autonomously, and transmits output actions instantaneously to its environment. The automaton's input is generated by the environment and transmitted instantaneously to the automaton. An automaton is unable to block an input action. In order to describe this classification formally, each automaton comes with an action signature.

Definition 4.1 Action Signature An action signature $\mathfrak{S}$ is a triple consisting of a three pairwise-disjoint sets of action, named $in(\mathfrak{S}), out(\mathfrak{S})$, and $int(\mathfrak{S})$. Actions in these sets are referred to as the input actions, output actions and internal actions of $\mathfrak{S}$, respectively. We also define $ext(\mathfrak{S}) = in(\mathfrak{S}) \cup out(\mathfrak{S})$ as the external actions of $\mathfrak{S}$, $local(\mathfrak{S}) = int(\mathfrak{S}) \cup out(\mathfrak{S})$ as the locally controlled actions of $\mathfrak{S}$, and $acts(\mathfrak{S}) = in(\mathfrak{S}) \cup out(\mathfrak{S}) \cup int(\mathfrak{S})$ as the actions of $\mathfrak{S}$.

Definition 4.2 Automaton Let $\mathfrak{S}$ be an action signature, and $\mathcal{I}$ a countable set. An automaton is a structure $\mathfrak{A} = \langle \mathfrak{S}, S, S_0, E, \{p_j\}_{j\in\mathcal{I}}, \mathcal{L}\rangle$ such that S is a set of states, $S_0 \subseteq S$ is a nonempty set of start states, $E \subseteq S \times acts(\mathfrak{S}) \times S$ a set of edges or transition relations, with the property that for every state s' and input action π there is a transition (s', π, s) in E. $\{p_i\}_{i\in\mathcal{I}}$ is a set of propositions, and $\mathcal{L} : S \to 2^{\{p_j\}_{j\in\mathcal{I}}}$ the function that assigns to each state a subset of the propositions.

When we put all the agents running together, we get a global automaton that is the parallel composition of the automata for all agents.

Defining composition of signatures is a preliminary step to define composition of automata. The composition of each component's action signature gives the action signature applicable to the whole system. This composition is only defined in case the component automata satisfy some simple compatibility conditions.

Definition 4.3 Compatible set of Action Signatures Let I be an index set that is at most countable. A collection $\{\mathfrak{S}_i\}_{i\in I}$ of action signatures is said to be compatible if:

(i) $out(\mathfrak{S}_i) \cap out(\mathfrak{S}_j) = \emptyset$, for all $i, j \in I, i \neq j$

(ii) $int(\mathfrak{S}_i) \cap acts(\mathfrak{S}_j) = \emptyset$, for all $i, j \in I, i \neq j$ and

(iii) no action is in $acts(\mathfrak{S}_i)$ for infinitely many i.

This restrictions will ensure that an action can not be under the control of more then one component of the system, and that internal actions of one component are not detectable by other components, when we define the composition of automata.

Definition 4.4 Composition of Action Signatures The composition $\mathfrak{S} = \Pi_{i\in I}\mathfrak{S}_i$ of a collection of compatible action signatures $\{\mathfrak{S}_i\}_{i\in i}$ is the action signature with:

- $in(\mathfrak{S}) = \cup_{i\in I}\ in(\mathfrak{S}_i) - \cup_{i\in I}\ out(\mathfrak{S}_i)$
- $out(\mathfrak{S}) = \cup_{i\in I}\ out(\mathfrak{S}_i)$
- $int(\mathfrak{S}) = \cup_{i\in I}\ int(\mathfrak{S}_i)$

Thus, output actions are those that are outputs for any of the component signatures, and similarly for internal actions. Input actions are any actions that are inputs to any of the component signatures, but outputs of no component signature. Note that interactions among components are outputs of the composition.

Now we can define composition of automata. The idea is that the states of the composed automaton are n-tuples whose components corresponds to a local state of each of the agents, and the edges corresponds to all edges of the local automata. The composition operation links output actions of one automaton with identically named input actions of any number of other automata.

Definition 4.5 Parallel Composition of Automata A collection $\{\mathfrak{A}_i = \langle \mathfrak{S}_i, S_i, (S_0)_i, E_i, \{p_j\}_{j\in\mathcal{I}_i}, \mathcal{L}_i\rangle\}_{i\in I}$ of automata is said to be compatible if their action signatures are compatible. The composition $\mathfrak{A} = \Pi_{i\in I}\mathfrak{A}_i$ of a finite compatible collection of automata $\{\mathfrak{A}_i\}_{i\in I}$ has the following components:

- $\mathfrak{S} = \Pi_{i\in I}\ \mathfrak{S}_i$,
- $S = \Pi_{i\in I} S_i$,
- $S_0 = \Pi_{i\in I}(S_0)_i$, and
- E is the set of triples $(s\prime, \pi, s)$ such that for all $i \in I$,
  · if $\pi \in \mathfrak{S}_i$, then $(s\prime[i], \pi, s[i]) \in E_i$, and
  · if $\pi \notin \mathfrak{S}_i$, then $s\prime[i] = s[i]$. [8]
- $\{p_j\}_{j\in\mathcal{I},\mathcal{I}\in\ \{\mathcal{I}_i\}_{i\in I}} = \bigcup\{p_j\}_{j\in\mathcal{I}_i}$
- $\mathcal{L}$ is a function that assigns to each global state a subset of propositions, according to the previous defined local functions for each agent: $(s, p_j), j \in \mathcal{I}$ is in $\mathcal{L}$ iff $j \in \mathcal{I}_i$ and $(s[i], p_j)$ is in $\mathcal{L}_i$.

[8] The notation $[i]$ denotes the i^{th} component of the state vector s.

States in the composed automaton corresponds to the *MAS*'s global states, and are the elements in the set of global states C. The set of runs can be easily obtained from the global automaton: each run in R is a path in the global automaton's computation tree.

When making the Parallel Composition of a set of Automata where each automaton dictates the behavior of an agent, it is possible and reasonable to get a composed global automaton $\mathfrak{A} = \langle \mathfrak{S}, S, S_0, E, \{p_i\}_{i\in\mathcal{I}}, \mathcal{L}\rangle$ where many states from S corresponds to the same local state component for a particular agent. When two such states s and t with the same local state component for agent i are connected by an edge in E, this means that agent i is incapable of noticing that a global state change has occurred when the global state passes from s to t. This edges denote local actions made by other agents different from i, and for this reason are imperceptible for agent i.

A relation for each agent can be defined over the states with this "indistinguishable" property.

Definition 4.6 Possibility Relation $\sim_i$ for agent i Let $\mathfrak{A} = \langle \mathfrak{S}, S, S_0, E, \mathcal{P}, \mathcal{L}\rangle$ be the composition of a compatible collection of automata $\{\mathfrak{A}_i\}_{i\in I} = \langle \mathfrak{S}_i, S_i, (S_0)_i, E_i, \{p_i\}_{i\in\mathcal{I}}, \mathcal{L}_i\rangle$ for all $i \in [1, \ldots, n]$. The possibility relation $\sim_i \in S \times S$ for each agent i is the smallest equivalence relation containing all pairs $(s\prime, s)$, $s\prime, s \in S$ such that there is an edge $(s\prime, \pi, s) \in E$ and $s\prime[i] = s[i]$.

Intuitively, two states are related by $\sim_i$ if agent i, being in one of them, considers it possible that the other one is the current state. In other words, agent i can't tell the current state from the other possible states.

It is important to notice that not all global states of the automaton where agent i has the same local state are connected by $\sim_i$, just the states that have an edge (corresponding to actions taken by agents other then i) connecting them.

4.2 Extended CTL *Language:* KCTL

We describe here a logic to reason about knowledge in state transition systems as the ones presented in the previous section. The language we use is an extension of *CTL* with an operator ($\mathcal{K}_k$) that provides the capability of observing the occurrence of an event from the point of view of one of the automaton (agent) involved in the system. From now on, this language will be referred as *KCTL*.

KCTL is useful to express properties about knowledge, time and events like 'Agent 1 knows that if he sends a message, agent 2 will eventually know that message"(in KCTL: $\mathcal{K}_1(msgsent \rightarrow \exists \mathsf{F}\mathcal{K}_2 msg)$,considering propositions msgsent for "agent 1 sent a message msg to agent 2" and proposition msg as the content of message). As in *CTL*, *KCTL* formulas reason about properties of computation trees. The tree is formed just by unwinding the global automaton (or Kripke Structure) that represents the *MAS* from its initial state. The computational tree describes all possible runs in the set of runs R.

Definition 4.7 Syntax of *KCTL* formulas Let $j \in \mathbb{N}$ and $\{\mathcal{P}_k\}_{1\leq k\leq j}$ be the set of

disjoint sets of propositions. The language of *KCTL* formulas is defined as follows:

$ForCTL(j, \{\mathcal{P}_k\}_{1\leq k\leq j})$ is the smallest set For of formulas such that:

- $p \in For$ iff exists k such that $1 \leq k \leq j$ and $p \in \mathcal{P}_k$,
- $\mathcal{K}_i(\phi) \in For$ iff $1 \leq i \leq j$ and $\phi \in For$,
- any other compound formula is formed in the same way as in *CTL* (see definition 2.1).

The semantics of *KCTL* formulas is given in terms of Kripke models, using the possibility relation $\sim_k$ for each agent k.

Thinking about the way the automaton-based model is constructed, and the assumptions made when the model was introduced, an agent k should be aware only of local states change, as k can't tell two global states apart when his local state did not change. So, to compare two global states from the point of view of k, we would have to consider just the state changes that k is able to perceive.

The previously defined possibility relation $\sim_k$ over global states is the key to identify sequences of global states in a run where the local state of agent k is the same.

Definition 4.8 Satisfiability relation for *KCTL* formulas Let $\{\mathcal{P}_k\}_{1\leq k\leq j}$ be a set of disjoint sets of propositions and $\mathfrak{M} = \langle S, S_0, R, \{\sim_k\}_{1\leq k\leq j}, \cup_{k=1}^{j}\mathcal{P}_k, \mathcal{L}\rangle$ be a K-extended Kripke Model, the satisfiability relation is defined in exactly the same way that it was done for *CTL* formulas, except for the new operator that is interpreted as follows:

$\mathfrak{M}, s \models \mathcal{K}_k(\phi)$ iff $\forall s' \in S | s' \sim_k s \implies \mathfrak{M}, s' \models \phi$

4.3 A Model Checking Process for Knowledge

We now present algorithms for the model checking problem described in section 2.2. We use an explicit representation of K-extended Kripke Models $\mathfrak{M} = \langle S, S_0, R, \{\sim_k\}_{1\leq k\leq j}, \cup_{k=1}^{j}\mathcal{P}_k, \mathcal{L}\rangle$ as automata where each state is labelled with the propositions associated by $\mathcal{L}$.

The process is the usual model checking process presented in [11]: "To check whether a *KCTL* formula f is satisfied in some state(s) of S, the process consists on labelling each state $s \in S$ with the set *label(s)* of subformulas of f which are true in s. Initially, *label(s)* is $\mathcal{L}(s)$. Then the algorithm goes through a series of iterations, adding subformulas to *label(s)*. During i^{th} iteration, subformulas with $i-1$ nested *KCTL* operators are processed and added to the labels of states where it is satisfied. At the end, $\mathfrak{M}, s \models f$ if and only if $f \in$ *label(s)*".

For the intermediate stages of the algorithm, it is necessary to handle seven cases: atomic formulas, $\neg, \vee, \exists\mathsf{X}, \exists\mathsf{G}, \exists\mathsf{U}$ and $\mathcal{K}$. The six first cases are the same for *CTL*. For formulas of the form:

- Atomic formulas, already handled;
- $\neg f_1$, label those states that are not labelled by f_1;
- $f_1 \vee f_2$, label those states that are labelled by either f_1, f_2 or both;

- $\exists \mathsf{X} f_1$, label those states that have a successor labelled by f_1;
- $\exists \mathsf{G} f_1$, first construct a restricted Kripke Model $\mathfrak{M}'$ [9], then partition the graph (S', R') into strongly connected components, next find those states that belong to nontrivial components, and then work backwards using the converse of R' and find all of those states that can be reached by a path in which each state is labelled with f_1, finally label these states with $\exists \mathsf{G} f_1$;
- $\exists [f_1 \mathsf{U} f_2]$, first find all states labelled with f_2, then work backwards using the converse relation R and find all states that can be reached by a path in which each state is labelled by f_1, then label all this states with $\exists [f_1 \mathsf{U} f_2]$;

Detailed algorithms for this cases with time complexity of $O(|S| + |R|)$ are given in [11].

We shall give special treatment to the seventh case, where the knowledge operator must be handled.

Following the intuitive meaning and the semantics defined for $\mathcal{K}_k$ operators in *KCTL*, to model check a formula of the form $\mathcal{K}_k f$ we must look to the indistinguishable states for agent k, related by $\sim_k$ equivalence possibility relation.

First, find the set of all states s labelled with f. Then, for each found state s, recursively check if all states t related to s (the current one) by $\sim_k$ (all t such that $s \sim_k t$) are labelled with f. If this is the case, label all them (the current state s and all states t, $s \sim_k t$) with $\mathcal{K}_k f$.

In spite of being a recursive process, this procedure is linear on the number of pairs in $\sim_k$. This is achieved because $\sim_k$ is an equivalence relation, what makes $s \sim_k t$ the same as $t \sim_k s$. The algorithm chooses a component and look for the possibilities for the second component. Each state is elected as first component just once, because we keep in track the states already elected in set L.

Once we have algorithms to the seven cases listed, to handle an arbitrary *KCTL* formula f just successively apply the state-labelling algorithm to the subformulas of f, starting with the shortest and most deeply nested one, and work outward until f is entirely checked. The complete process takes time $O(|f| \cdot (|S| + |R| + \Sigma_{k=1}^{j} | \sim_k |))$.

We can estimate the number of pairs in the relations $\sim_k$, as the number of indistinguishable states to agent k depends on the number of actions each other agent in the system can do while k doesn't act. This is proportional to the number of edges (transitions) in the global automaton. For an agent k, the number of pairs in $\sim_k$ is the number of edges of the whole system minus the number of actions local to k. Considering that N is the number of agents in the system, $\Sigma_{k=1}^{j} | \sim_k | \leq (|R| * (N-1))$, and thus we have $O(|f| \cdot (|S| + N * |R|)$ as an upper limit for complexity.

The pseudo-code algorithms to model check formulas of the form $\mathcal{K}_k f$ are stated in algorithms 1 and 2.

[9] $\mathfrak{M}' = \langle S', S'_0, R', \{\sim_k\}'_{1 \leq k \leq j}, \cup_{k=1}^{j} \mathcal{P}_k, \mathcal{L}' \rangle$ is obtained from $\mathfrak{M}$ by deleting from S all those states at which f_1 does not hold and restricting R and L accordingly. R' may not be total

Algorithm 1 *CheckK(G, f, k)*
G [in parameter] is a (global) automaton;
f [in parameter] is a formula to be evaluated;
k [in parameter] is an agent identifier;
L is a set of states, initially empty;

$L := \emptyset$; $S := \{s | f \in \text{label}(s)\}$
For Each *state s in S* ***do***
If *RecursiveCheckK(G, f, k, L, s)*
then For Each *state t in L* ***do***
$label(t) := label(t) \cup \{\mathcal{K}_k f\}$

Algorithm 2 *RecursiveCheckK(G, f, k, L, s)*
G [in parameter] is a (global) automaton;
f [in parameter] is a formula to be evaluated;
k [in parameter] is an agent identifier;
L [in parameter] is a set of states, initially empty;
s [in parameter] is a state;

If $s \notin L$ ***Then***
If $f \in \text{label}(s)$ ***Then***
Begin
$L := L \cup \{s\}$
For Each *state* $t \in G$, *so that* $t \sim_k s$ ***do***
If *RecursiveCheckK(G, f, k, L, t)* ***Then***
return True
Else *return False*
End
Else *return True*

5 Example: The Alternating Bit Protocol

The *alternating bit protocol* is a well-known basic communication protocol. It is often used as a test case, either for some algebraic formalism or for the analysis or verification of concurrent systems. It consists of three components, or "agents": a **sender** agent, a **receiver** agent and a **communication channel**. The sender has a set of messages to send to the receiver over the communication channel. However, the channel isn't reliable, it can lose any messages going through it.

It is assumed that the channel can only transport one message at a time and that it is bi-directional, that is, it can transport messages from the sender to the receiver and it can also from the receiver to the sender.

The protocol starts with the sender selecting the first message to send. This message is extended with a control bit (initially 0) to form a frame and this frame is sent along the communication channel. Right as the sending of the frame starts, the sender also starts a timer. When this timer counts down to zero, the sender

will assume the frame was lost (timeout) and will retransmit it.

The communication channel then transmits the frames from the sender to the receiver. Two situations are possible: the frame is properly transmitted, or the frame is lost during transmission.

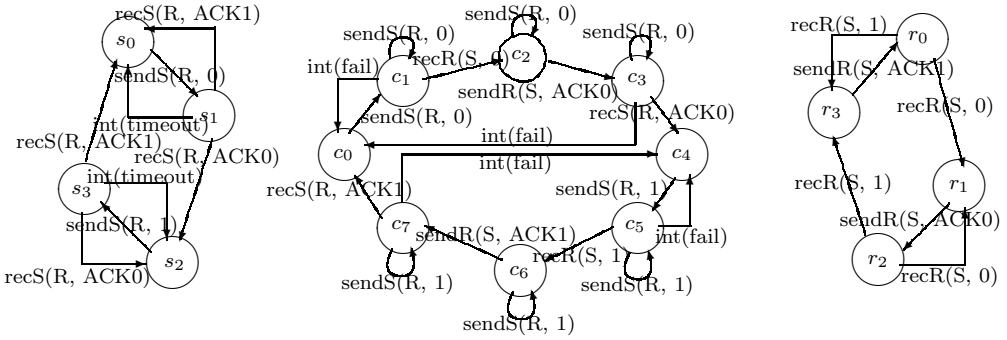
If the frame wasn't lost, the receiver reads the frame from the channel. The receiver then checks the control bit in the frame. If this bit matches the internal control bit of the receiver, the message in the frame is acknowledged, that is, the receiver sends an acknowledgement message with the control bit to the sender over the communications channel. Receiver then flips his internal control bit and waits for another frame. If the bit of the received frame was wrong, the receiver sends a negative acknowledgement (with a flipped control bit), and waits for a retransmission of the frame.

As the channel is not reliable, there is also the chance that the acknowledgement is lost. If it happens, there is nothing to do but wait until the timer runs down to zero. The sender will then retransmit the frame and, assuming the frame reaches the receiver, it will cause the receiver to transmit a new acknowledgement equal to the one which was lost.

The process continues until the sender receives the acknowledgement of a successful transmission over the communications channel. Such acknowledgements are the ones with the control bit matching the internal control bit of the sender. If the bit doesn't match, the acknowledgement message is ignored. After a successful transmission, the sender flips the control bit, selects the next message to send and starts all over again.

5.1 System specification

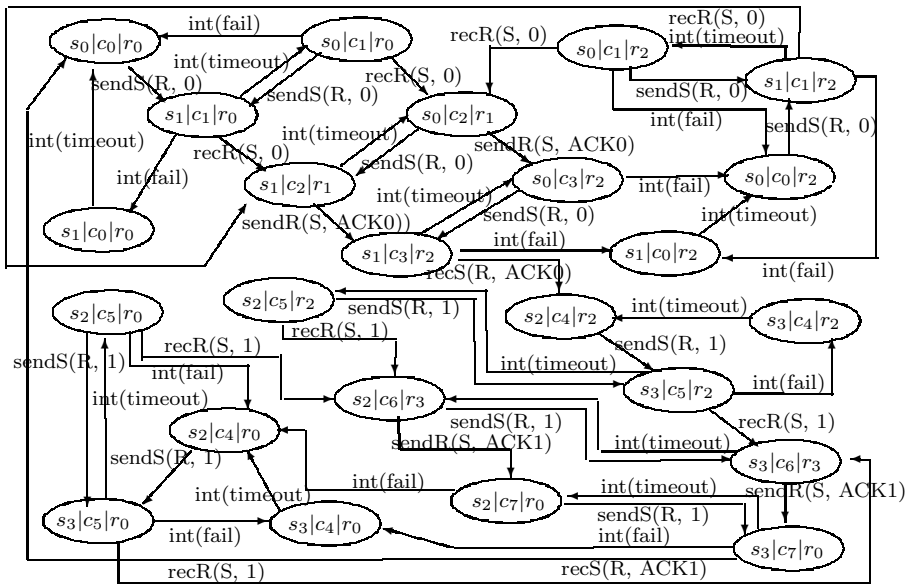
Each agent is modelled as one automaton, which represents the agent's behavior and local information. Here we present the local automata for sender, channel and receiver, respectively.



The local information available for sender is incorporated at the model as the set of propositions that hold on each state. The same happens with the receiver. We consider that the communication channel does not keep track of any local information besides the current state. A simple list of the information valid for sender and receiver at each local state follows.

Sender	**Receiver**
S0: sent_msg_bit_1 received_Ack1	R0: receiving_msg_bit_0 sent_Ack1
S1: sending_msg_bit_0 receiving_Ack0	R1: received_msg_bit_0 sending_Ack0
S2: sent_msg_bit_0 received_Ack0	R2: receiving_msg_bit_1 sent_Ack0
S3: sending_msg_bit_1 receiving_Ack1	R3: received_msg_bit_1 sending_Ack1

The global automaton is obtained from the composition of the local automata, as stated in definition 4.5.



5.2 Checking the model

There are many interesting formulas expressible in KCTL about this model.

Here we sketch the partial results of the model checking process for a KCTL formula involving temporal and knowledge operators:

$G, (s_1|c_0|r_0) \models \mathcal{K}_S\mathcal{K}_R\exists((\exists \mathsf{F} sending\_msg\_bit\_0)\mathsf{U} received\_ACK0)$: At the starting point, does the Sender knows that the Receiver knows that Sender will keep trying to send frame 0 until he receives and Acknowledgement for frame 0?

Result for *sending_msg_bit_0*: All global states that contain the local state s_1.

Result for $\exists \mathsf{F} sending\_msg\_bit\_0$: All global states that contain either the local state s_1 or the local state s_0.

Result for $received\_ACK0$: All global states that contain the local state s_2.

Result for $\exists \mathsf{F} sending\_msg\_bit\_0) \mathsf{U} received\_ACK0$: All global states that contain the local state s_0, or the local state s_1, or the local state s_2.

Result for $\mathcal{K}_R \exists((\exists \mathsf{F} sending\_msg\_bit\_0) \mathsf{U} received\_ACK0)$: $\{(s_0|c_0|r_0)$, $(s_1|c_1|r_0)$, $(s_1|c_0|r_0)$, $(s_0|c_1|r_0)$, $(s_1|c_2|r_1)$, $(s_0|c_2|r_1)\}$

Result for $\mathcal{K}_S \mathcal{K}_R \exists((\exists \mathsf{F} sending\_msg\_bit\_0) \mathsf{U} received\_ACK0)$: $\{(s_0|c_0|r_0)$, $(s_1|c_0|r_0)\}$. As we have $(s_0|c_0|r_0)$ among the states where the formula is valid, then the model checking process returns **true**.

6 Conclusions

In this work we present an approach to construct a global model for concurrent systems from local automata-based specifications, and algorithms to automatically verify properties over this model. The global model obtained captures both behavior and epistemic properties, so we propose a model checking process for a combined CTL and knowledge modal logics - *KCTL*.

The language *KCTL* is a CTL language extend with knowledge operators. Algorithms to model check knowledge properties of concurrent systems were explored in many recent works like [3], [6], [12], [8] and [5]. Our work benefits from the results stated by these previous works, and focus on the way knowledge relations can be captured from automata-based local specifications.

Our main contribution is to describe a model suitable for model checking knowledge in a basic pre-defined way, but with the advantage that the knowledge relations for this would be extracted directly from the automata-based model. Doing so, we avoid the tedious task of identifying and formally stating the knowledge relations (or epistemic states) together with the system specification, what is also a way to keep a clean specification.

We presented a compositional semantics for *KCTL*, constructed upon local automata for each agent. These automata are used to automatically generate the global automaton for the concurrent system and also the knowledge relations for each agent.

With adequate model and language in hands, Model Checking algorithms for checking $\mathcal{K}_k$ formulas are defined and examined in detail.

References

[1] Alur, R., T. A. Henzinger and O. Kupferman, *Alternating-time temporal logic*, J. ACM **49** (2002), pp. 672–713.

[2] Ben-Ari, M., Z. Manna and A. Pnueli, *The temporal logic of branching time*, in: *POPL '81: Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (1981), pp. 164–176.

[3] Benerecetti, M., F. Giunchiglia and L. Serafini, *A model checking algorithm for multiagent systems*, in: *ATAL '98: Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages* (1999), pp. 163–176.

[4] Clarke, E. M. and Emerson, *Design and synthesis of synchronization skeletons using branching time temporal logic*, in: *Workshop on Logics of Programs, Lecture Notes in Computer Science 131* (1981), pp. 52–71.

[5] der Hoek, W. V. and M. Wooldridge, *Cooperation, knowledge and time: Alternating-time temporal epistemic logic and its applications*, Studia Logica (2003), pp. 125–157.

[6] Engelfriet, J., C. M. Jonker and J. Treur, *Compositional verification of multi-agent systems in temporal multi-epistemic logic*, in: *Pre-proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages, ATAL'98* (1998), pp. 91–106.

[7] Fagin, R., J. Y. Halpern and Y. Moses, "Reasoning about knowledge," MIT Press, Cambridge, Massachusetts, 1995.

[8] Kacprzak, M., A. Lomuscio and W. Penczek, *Verification of multi-agent systems via unbounded model checking* (2004).

[9] Lehmann, D., *Knowledge, common knowledge and related puzzles (extended summary)*, in: *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing* (1984), pp. 62–67.

[10] Lynch, N., M. Merritt, W. Weihl and A. Fekete, "Atomic Transactions," Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[11] Peled, D. A., E. M. Clarke and O. Grumberg, "Model Checking," MIT Press, Cambridge, Massachusetts, 2000.

[12] van der Meyden, R. and N. V. Shilov, *Model checking knowledge and time in systems with perfect recall (extended abstract)*, in: *Foundations of Software Technology and Theoretical Computer Science*, 1999, pp. 432–445.