Improved Bounded Model Checking for a Fair Branching-Time Temporal Epistemic Logic⋆

Xiaowei Huang, Cheng Luo, and Ron van der Meyden

The University of New South Wales, Australia.
{xiaoweih,luoc,meyden}@cse.unsw.edu.au

Abstract. Bounded model checking is a verification technique based on searching for counter-examples to the validity of the specification using an encoding to propositional sastisfiability. The paper identifies a number of inefficiencies in prior encodings for bounded model checking for a logic of knowledge and branching time. An alternate encoding is developed, and theoretical and experimental results are presented that show this leads to improved performance of bounded model checking for a range of examples.

1 Introduction

In the context of distributed and multi-agent systems, as well as autonomous systems that must operate in an uncertain environment, it has been argued that *epistemic* logics, i.e., logics of *knowledge*, provide a useful expressiveness for dealing with agents' need to relate their actions to their state of information [5]. This has led to the study of model checking for temporal epistemic logics [6, 10]. There exist a variety of approaches to model checking. Binary Decision Diagram (BDD) techniques use a graph-based encoding to efficiently represent boolean functions and computes the set of states satisfying the specification in this encoding. A more recent approach is Bounded Model Checking (BMC) [1], which works by representing the statement that there exists a counter-example to the specification, of a particular structure and finite size k, as a propositional logic formula, and then using SAT-solving to determine the satisfiability of this formula.

Bounded model checking was first proposed in the context of linear-time temporal logic, where the structure of the counter-examples can be taken to be a run, a linear sequence of states, with the final one equal to one of the intermediate states to represent cyclical behaviour. There have subsequently been proposals to apply BMC to branching-time temporal logics, and to logics combining temporal and epistemic logic. A BMC encoding for ACTL, the universal fragment of the branching time logic CTL, has been proposed in [14], and extended to the richer logic ACTL* (which combines elements of linear- and branching-time logics) in [16]. The encoding for ACTL has been extended to a logic ACTLK$_n$, which also contains epistemic operators, in [13].

We show in this paper that it is possible to significantly improve upon the efficiency of BMC for temporal epistemic logic by means of an improved encoding. We develop

⋆ Work supported by Australian Research Council Linkage Grant LP0882961 and Defence Research and Development Canada (Valcartier) contract W7701-082453. An abstract of this paper also appears in the AAMAS 2010 proceedings.

an efficient encoding for fair ACTLK$_n$ logic, which extends ACTLK$_n$ with a generalized Büchi fairness condition. Two main ideas underly the efficiency of our encoding. First, we sharpen the relationship between the formula and the runs in the counter-example: rather than simply evaluating the semantics of the formula over the counter-example, so that any run could be a candidate for the witness required for an existential claim, our encoding identifies a particular run as providing the required witness. Secondly, we associate particular subformulas with particular points in the counter-example structure, and use atomic propositions to represent the satisfaction of these subformulas in a way that eliminates exponential blowups in previous encodings by means of structure-sharing. Additionally, we use a number of optimizations that enable the number of runs required in the search for a counter-example to be reduced, and the shape of these runs to be simplified in a number of cases.

We show by both theoretical arguments and experimental results that our encoding yields an improved performance of BMC on a range of examples. Theoretically, we present examples where the size of the encoding is reduced from exponential to quadratic. One such example is the "nested knowledge" formula $(K_a K_b)^n p$ expressing that two agents a, b have degree n mutual knowledge of the proposition p.

Such improvements in encoding size are shown to have a significant impact on the runtimes required to find counter-examples in practice. In our experimental results, we have implemented three BMC encodings (that of [13], an earlier improvement [17] and our new encoding) in the epistemic model checker MCK [6]. MCK already supported a range of BDD-based model checking algorithms. We report the results of experiments on several protocols, including Dining Cryptographers [3], Byzantine Generals [9] and a simple Pursuit-Evasion Game. For each example, we consider a number of specification formulas. Both the systems description and the specification formulas involve a numerical parameter, and we show how the run-time of model checking scales with this parameter in each experiment, comparing the BMC techniques and a BDD-based technique. The experimental results show that our new BMC encoding often yields a much better performance than the previous BMC encodings. On the other hand, which of our new BMC encoding and BDD-based model checking is more efficient depends on the example.

The structure of the paper is as follows. Section 2 defines the logic of knowledge and time ACTLK$_n$ that we study, and defines the model checking problem for this logic. Previous bounded model checking approaches for this and related logics are reviewed in Section 3. We describe our new encoding in Section 4, where we also motivate on theoretical grounds why we expect this encoding to yield improved model checking performance. This is followed in Section 5 by a discussion of experimental results which validate and quantify the improved performance. Section 6 discusses related work, and Section 7 concludes with a discussion of future work.

2 Preliminaries

We work with a logic ACTLK$_n$ that combines the branching time logic ACTL (i.e., the universal fragment of the branching time temporal logic CTL) and the logic of knowledge and common knowledge for n agents, as well as its dual ECTLK$_n$. Dually, the logic ECTLK$_n$ can be defined as $\{\neg\psi \mid \psi \in \text{ACTLK}_n\}$; we give an expressively equivalent syntax for this logic below. In model checking these logics, we are interested in verifying that an ACTLK$_n$ formula is valid in a model. To find a counterexample for a specification ψ in the logic ACTLK$_n$ is the same as to find a witness for $\phi = \neg\psi$ in the logic ECTLK$_n$. Since BMC works by searching for such witnesses, we concentrate on the ECTLK$_n$ syntax in what follows.

Let *Prop* be a set of atomic propositions and $Ags = \{1, \ldots, n\}$ be a set of n agents. $\mathbb{T}$ and $\mathbb{F}$ are used to denote the truth values True and False, respectively. The syntax of ACTLK$_n$ is given by the following grammar [1] :

$$\gamma \ :== \ p \mid \neg p \mid \alpha \vee \beta \mid \alpha \wedge \beta \mid AX\alpha \mid AF\alpha \mid AG\alpha \mid A(\alpha U\beta) \mid K_i\alpha$$

where $p \in Prop$, and $i \in Ags$. Similarly, the syntax of ECTLK$_n$ is given by the grammar:

$$\gamma \ :== \ p \mid \neg p \mid \alpha \vee \beta \mid \alpha \wedge \beta \mid EX\alpha \mid EF\alpha \mid EG\alpha \mid E(\alpha U\beta) \mid \overline{K_i}\alpha$$

where $p \in Prop$, and $i \in Ags$. The connection between the two languages is given by the following equivalences: $\neg AX\alpha = EX\neg\alpha$, $\neg AF\alpha = EG\neg\alpha$, $\neg AG\alpha = EF\neg\alpha$, $\neg A(\alpha U\beta) = EG\neg\beta \vee E(\neg\beta U(\neg\alpha \wedge \neg\beta))$, $\neg K_i\alpha = \overline{K_i}\neg\alpha$. These equivalences may be used together with DeMorgans laws to transform $\neg\phi$, for any ACTLK$_n$ formula ϕ, into an equivalent ECTLK$_n$ formula.

We use a semantics for ECTLK$_n$ that is based on a variant of the *interpreted systems* model for the logic of knowledge [5]. Let W be a set, which we call the set of global states. A *run* over W is a function $r : \mathbf{N} \to W$. An *interpreted system* over W for n agents is a tuple $\mathcal{I} = (\mathcal{R}, \sim^1, \ldots, \sim^n, \pi)$, where $\mathcal{R}$ is a set of runs over W, each $\sim^i$ is an equivalence relation on W, and $\pi : W \to \mathcal{P}(Prop)$ is an interpretation function.

A *point* of $\mathcal{I}$ is a pair (r, m) where $r \in \mathcal{R}$ and $m \in \mathbf{N}$. We say that a run r' is *equivalent to a run r up to time $m \in \mathbf{N}$* if $r'(k) = r(k)$ for $0 \leq k \leq m$. We define the semantics of ECTLK$_n$ by means of a relation $\mathcal{I}, (r, m) \models \phi$, where $\mathcal{I}$ is an intepreted system, (r, m) is a point of $\mathcal{I}$, and ϕ is a formula. This relation is defined inductively as follows:

- $\mathcal{I}, (r, m) \models p$ if $p \in \pi(r(m))$,
- $\mathcal{I}, (r, m) \models \neg p$ if not $\mathcal{I}, (r, m) \models p$
- $\mathcal{I}, (r, m) \models \alpha \vee \beta$ if $\mathcal{I}, (r, m) \models \alpha$ or $\mathcal{I}, (r, m) \models \beta$
- $\mathcal{I}, (r, m) \models \alpha \wedge \beta$ if $\mathcal{I}, (r, m) \models \alpha$ and $\mathcal{I}, (r, m) \models \beta$
- $\mathcal{I}, (r, m) \models EX\alpha$ if there exists a run $r' \in \mathcal{R}$ equivalent to r up to time m such that $\mathcal{I}, (r', m + 1) \models \alpha$
- $\mathcal{I}, (r, m) \models EF\alpha$ if there exists a run $r' \in \mathcal{R}$ equivalent to r up to time m and $m' \geq m$ such that $\mathcal{I}, (r', m') \models \alpha$.
- $\mathcal{I}, (r, m) \models EG\alpha$ if there exists a run $r' \in \mathcal{R}$ equivalent to r up to time m such that $\mathcal{I}, (r, m') \models \alpha$ for all $m' \geq m$
- $\mathcal{I}, (r, m) \models E(\alpha U\beta)$ if there exists a run $r' \in \mathcal{R}$ equivalent to r up to time m and a time m' such that $\mathcal{I}, (r, m') \models \beta$ and $\mathcal{I}, (r, m'') \models \alpha$ for all m'' with $m \leq m'' < m'$
- $\mathcal{I}, (r, m) \models \overline{K_i}\phi$ if for some point (r', m') of $\mathcal{I}$ such that $r(m) \sim_i r'(m')$ we have $\mathcal{I}, (r', m') \models \phi$

[1] In a longer version of the paper we include common knowledge operators, which we omit here for brevity.

For the knowledge operators, this semantics is essentially the same as the usual (observational) interpreted systems semantics. For the temporal operators, it corresponds to a semantics for branching time known as the *bundle semantics* [2, 12].

While they give a clean and coherent semantics to the logic, interpreted systems are not suitable as inputs for a model checking program, since they are infinite structures. We therefore also work with an alternate semantic representation based on transition systems with epistemic indistinguishability relations and fairness condition. A (finite) *system* is a tuple $M = (W, I, \Rightarrow, \sim^1, \ldots, \sim^n, \pi, \chi)$ where W is a (finite) set of global states, $I \subseteq W$ is the set of initial states, $\Rightarrow \subseteq W \times W$ is a serial temporal transition relation, each $\sim_i \subseteq W \times W$ is an equivalence relation representing epistemic accessibility for agent $i \in Ags$, $\pi : W \to \mathcal{P}(Prop)$ is a propositional interpretation, and $\chi \subseteq \mathcal{P}(W) \setminus \emptyset$ is a *generalized Büchi fairness condition*. The system M can also be regarded as a generalized Büchi automaton with χ the set of acceptance sets.

Given a system M over global states W, we may construct an interpreted system $\mathcal{I}(M) = (\mathcal{R}, \sim_1, \ldots, \sim_n, \pi)$ over global states W, as follows. The components $\sim^i$ and π are identical to those in M. The set of runs is defined as follows. We say that a *fullpath* from a state w is an infinite sequence of states $w_0 w_1 ...$ such that $w_0 = w$ and $w_i \Rightarrow w_{i+1}$ for all $i \geq 0$. We use $Path(w)$ to denote the set of all fullpaths from state w. The fairness condition is used to place an additional constraint on fullpaths. A fullpath $w_0 w_1 \ldots$ is said to be *fair* if for all $Q \in \chi$, there exists a state $w \in Q$ such that $w = w_i$ for infinitely many i. A *run* of the system is a fair fullpath $w_0 w_1 \ldots$ with $w_0 \in I$. We define $\mathcal{R}$ to be the set of runs of M. A formula ϕ of ACTLK$_n$ is said to hold in M, written $M \models_A \phi$, if $\mathcal{I}(M), (r, 0) \models \phi$ for all $r \in \mathcal{R}$. Dually, a formula ϕ of ECTLK$_n$ is said to be satisfiable in M, written $M \models_E \phi$, if $\mathcal{I}(M), (r, 0) \models \phi$ for some $r \in \mathcal{R}$.

We say that a state w is *fair* if it is the initial state of some fair fullpath, otherwise the state is *unfair*. A state w is *reachable* if there exists a sequence $w_0 \Rightarrow w_1 \Rightarrow \ldots w_k = w$ where $w_0 \in I$. (Some care with this is required because of the epistemic operators.) A state is fair and reachable iff it occurs in some run. Note that some reachable states may be unfair — we cannot always assume that a transition takes us to a fair state.

2.1 Model Checking Input Format

From now on, we fix a system M, a specification ψ in ACTLK$_n$. We are interested in determining whether $M \models_A \psi$, or equivalently, whether $M \models_E \phi$ for the ECTLK$_n$ formula ϕ corresponding to $\neg\psi$.

We will assume that the system M is presented in a particular format, in which the states of the system are viewed as assignments to a set of boolean variables, and the other components of M are represented by means of propositional logic formulas. In particular, we assume that there are N boolean variables making up a state. A state can therefore be represented as a boolean vector of length N. To refer to an arbitrary state, we may use a vector $\boldsymbol{s} = (s_1, \ldots, s_N)$ of N boolean variables s_i. Given such a vector, let $\boldsymbol{s}' = (s'_1, \ldots, s'_N)$ be the "primed" vector of symbols obtained by adding a prime symbol to each variable name to create N distinct variable names. We assume that the system M is presented as a tuple $\langle \boldsymbol{s}, I(\boldsymbol{s}), T(\boldsymbol{s}, \boldsymbol{s}'), H_1(\boldsymbol{s}, \boldsymbol{s}'), \ldots, H_n(\boldsymbol{s}, \boldsymbol{s}'), \chi \rangle$, where

- $\boldsymbol{s}$ identifies the variables that make up the state, or are used to compute state transitions,

– $I(\boldsymbol{s})$ is a propositional logic formula; a state is initial if it satisfies this formula,
– $T(\boldsymbol{s}, \boldsymbol{s'})$ is a propositional logic formula representing the transition relation $\Rightarrow$; there is a transition for a state represented by an assignment to $\boldsymbol{s}$ to the state represented by an assignment $\boldsymbol{s'}$ if this formula holds with respect to the union of these assignments.
– $H_i(\boldsymbol{s}, \boldsymbol{s'})$ is a propositional logic formula representing the indistinguishability relation $\sim_i$ for agent i,
– $\chi = \{F_1(\boldsymbol{s}), \ldots, F_m(\boldsymbol{s})\}$ is a set (possibly empty) of propositional logic formulas $F_i(\boldsymbol{s})$, each representing one of the sets of states in a generalized Büchi fairness condition χ.

In addition to these formulas, we will make use of the formula $H(\boldsymbol{s}, \boldsymbol{s'}) = \bigwedge_{i=1}^{N} s_i \Leftrightarrow s'_i$ which asserts the the states represented by $\boldsymbol{s}$ and $\boldsymbol{s'}$ are identical.

Given this representation of a system, we may represent length k fragments of runs of the system using a sequence $r = r(0), r(1), \ldots, r(k-1)$, where each $r(i) = (r(i)_1, \ldots, r(i)_N)$ is a vector of N variables. We use the following formulas to express properties of such sequences:

1. $\mathtt{Runf}_k(r) = \bigwedge_{i=0}^{k-2} T(r(i), r(i+1))$ expresses that r is a run fragment, in the sense that there is a transition from each state to the next,
2. $\mathtt{CRunf}_k(r, l) = \mathtt{Runf}_k(r) \wedge \bigvee_{h=0}^{k-1}(h = l \wedge T(r(k-1), r(h)))$ expresses that r is a cyclic run fragment. Here l is an additional variable of type $\{0 \ldots k-1\}$, representing the point at which the cycle starts.
3. $\mathtt{FCRunf}_k(r, l) = \mathtt{CRunf}_k(r, l) \ \wedge \ \bigwedge_{t=1}^{m} \bigvee_{h=0}^{k-1}(h \geq l \wedge F_t(r(h)))$ expresses that r is a *fair* cyclic run fragment. Fairness is obtained from the fact that each condition F_t in the generalized Büchi fairness condition holds at some point in the cycle. This implies that when we unfold the cyclic run to an infinite run, each condition F_i will be satisfied infinitely often, as required. (We remark that the use of the variable l helps to reduce the size of this formula by a factor of k.)

3 Previous Bounded Model Checking Algorithms for ACTLK$_n$

Bounded model checking approaches the problem of model checking a formula ψ in a system M via a search for counter-examples to the validity of the formula. These counter-examples are parameterized by their size k, and the existence of a counter-example of size k satisfying the formula $\phi = \neg\psi$ is encoded as a propositional logic formula $[M, \ \phi]_k$. Propositional logic SAT-solvers are then used to search for a satisfying assignment of this formula.

The details of the encoding depend upon the specification logic in question, and for a number of logics there have been several distinct proposals for encodings, with different complexity properties. In this section, we describe two encodings that have been proposed in the past for branching-time temporal and epistemic logics. This sets the context for our proposed optimizations.

3.1 Encoding of Penczeck et al

Penczeck et al [14] first proposed a BMC encoding for the logic ECTL, i.e., the logic ECTLK$_n$ described above, but without the knowledge operators. This encoding was later extended for ECTLK$_n$ [13]. In both cases, the encodings were for systems without fairness conditions, i.e., systems in which $\chi = \emptyset$ in the presentation above.

The basis for the encoding is a representation of forest-like counter-examples as set of run fragments. Intuitively, each time that the encoding needs to deal with an existential formula (such as $EF\alpha$, which requires the existence of a branch from the present point on which α is eventually satisfied), it uses a new run fragment (in the case of $EF\alpha$, this fragment is required to contain a point at which α holds). The BMC parameter k is taken to be the length of the run fragments. The total number of run fragments required to express the expected shape of the counter-example for a given value k for the formula is $1 + f_k(\phi)$, where the function f_k is defined recursively as follows: $f_k(p) = f_k(\neg p) = 0$ for $p \in Prop$, $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$, $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$, $f_k(Y\alpha) = f_k(\alpha) + 1$ with $Y \in \{EX, EF, \overline{K_i}\}$, $f_k(EG\alpha) = k \cdot f_k(\alpha) + 1$, $f_k(E(\alpha U \beta)) = (k-1) \cdot f_k(\alpha) + f_k(\beta) + 1$. A uniform notation is used for these run fragments. We write r_i for the *ith* run fragment. (For $i \neq j$, no variable is shared between the run fragments r_i and r_j.)

The whole encoding is made up of two parts as follows.

$$[M,\ \phi]_k \equiv I(r_0(0)) \wedge \bigwedge_{j=0}^{f_k(\phi)} \texttt{Runf}_k(r_j) \wedge [\phi]_k^{0,0} \qquad (1)$$

The first part simply says that each r_j is a run fragment, and that the first state of r_0 is an initial state of the system. The second part states that this structure supports the formula ϕ. The notation $[\alpha]_k^{m,n}$ is defined in Table 1. Intuitively, this states that formula α holds at state m on run fragment r_n. We take p_i to be the i-th state variable, so that $(r_n(m))_i$ is the instance of this variable at the m-th state of the run fragment r_n.

$[p_i]_k^{m,n}$	$\equiv$	$(r_n(m))_i$
$[\neg p_i]_k^{m,n}$	$\equiv$	$\neg(r_n(m))_i$
$[\alpha \wedge \beta]_k^{m,n}$	$\equiv$	$[\alpha]_k^{m,n} \wedge [\beta]_k^{m,n}$
$[\alpha \vee \beta]_k^{m,n}$	$\equiv$	$[\alpha]_k^{m,n} \vee [\beta]_k^{m,n}$
$[EX\alpha]_k^{m,n}$	$\equiv$	$\bigvee_{j=1}^{f_k(\phi)}(H(r_n(m), r_j(0)) \wedge [\alpha]_k^{1,j})$
$[EG\alpha]_k^{m,n}$	$\equiv$	$\bigvee_{j=1}^{f_k(\phi)}(H(r_n(m), r_j(0)) \wedge \bigwedge_{l=0}^{k-1}[\alpha]_k^{l,j})$
$[EF\alpha]_k^{m,n}$	$\equiv$	$\bigvee_{j=1}^{f_k(\phi)}(H(r_n(m), r_j(0)) \wedge \bigvee_{l=0}^{k-1}[\alpha]_k^{l,j})$
$[E(\alpha U \beta)]_k^{m,n}$	$\equiv$	$\bigvee_{j=1}^{f_k(\phi)}(H(r_n(m), r_j(0)) \wedge \bigvee_{l=0}^{k-1}([\beta]_k^{l,j} \wedge \bigwedge_{t=0}^{l-1}[\alpha]_k^{t,j}))$
$[\overline{K_i}\alpha]_k^{m,n}$	$\equiv$	$\bigvee_{j=1}^{f_k(\phi)}(I(r_j(0)) \wedge \bigvee_{l=0}^{k-1}([\alpha]_k^{l,j} \wedge H(r_n(m), r_j(l))))$

Table 1. Encoding Function $[\gamma]_k^{m,n}$ for ECTLK$_n$ of Penczeck et al

For purposes of comparison with our encoding below, which takes fairness conditions into account, we note that fairness may be incorporated into this encoding by means of a simple change, using $\texttt{FCRunf}_k(r_j)$ where the encoding above uses $\texttt{Runf}_k(r_j)$. We note that this use of cyclic runs is similar to their use in the BMC encodings for ACTL* [16] or ACTL*K$_n$ [11].

3.2 Improved Encoding for ECTL by Zbrzezny

Zbrzezny [17] noted that the ECTL encoding of Penczeck et al [14] assumes that there exist sufficient run fragments in the counter-example to satisfy the existential subformulas encountered, but it needs to evaluate *all* of these fragments to check whether it provides the required witness. For example, in the clause for $EG\alpha$ in Table 1 the purpose of the top level disjunction over $j = 1 \ldots f_k(\phi)$ is to assert that one of the run fragments in the counter-example satisfies α at all points. This is inefficient: since the number $f_k(\phi)$ of run fragments is deliberately chosen to be large enough to supply all the witnesses required, we can allocate a *specific* run fragment to each witness ahead of time, and replace the check against all run fragments by a check against the specific run fragment that is supposed to provide the witness.

Zbrzezny gives a BMC encoding for ECTL that is based on this observation, and shows that his encoding leads to improved performance for model checking ECTL. We skip the full details of his encoding here: it requires some careful bookkeeping of run numbers during the encoding. Our own encoding below incorporates this idea with a slightly different formulation, but goes on to deal with the full logic ECTLK$_n$ in a way that incorporates further optimizations. For purposes of comparison we give just the clause for $EF\alpha$ (a simplication of the clause for $E(\alpha U\beta)$ actually presented), which defines the encoding $[EF\alpha]_k^{[m,n,A]}$ as

$$H(r_n(m), r_{min(A)}(0)) \wedge \bigvee_{j=0}^{k-1} [\alpha]_k^{[j,min(A),A\setminus\{min(A)\}]} .$$

Intuitively, A is a set of indices of free run fragments, $min(A)$ is index of the next available run fragment, and $A \setminus \{min(A)\}$ is the set of run fragments remaining for the encoding of witnesses required by α.

4 Improved encoding for ACTLK$_n$

In this section we define an encoding for ACTLK$_n$ that improves upon the encodings discussed in the previous section. We begin by noting some inefficiencies in these encodings, and noting some opportunities for optimization.

4.1 Motivation

Note that both the encodings of Penczeck et al and Zbrzezny construct the encoding $[\alpha]_k$ recursively, but in the process introduce some large disjunctions or conjunctions when dealing with modal operators. For example, for $[EF\alpha]_k$, both encodings use a subformula of the form $\bigvee_{j=0}^{k-1}[\alpha]_k^j$. In the case of formulas with deeply nested operators, this leads to an exponential blowup. For example for the formula ϕ_h defined by $\phi_0 = p_0$ and $\phi_{i+1} = p_{i+1} \wedge EF(\phi_i)$, even using the more efficient Zbrzezny approach we would obtain an encoding $[\phi_h]_k$ of the structure

$$... \bigvee_{j_1=0}^{k-1}(... \bigvee_{j_2=0}^{k-1} \ldots (... \bigvee_{j_h=0}^{k-1}(...)))))$$

which has size of the order k^h. On the other hand, the set of run fragments $\{r_0 \dots r_h\}$ necessary for this encoding is of size merely $h + 1$, and each run fragment has k states. (Although Zbrzezny does not discuss knowledge, application of his ideas would involve dropping only the outer disjunction in the case for $\overline{K}_i$ in Table 1, so a similar blowup would be obtained for formulas such as $(K_a K_b)^h p$ that have been of interest when dealing with knowledge.)

We note that it is possible to encode this example more efficiently by introducing propositions $e_\phi^{r,n}$ representing that ϕ holds at the point (r, n). By Zbrzezny's ideas, we can witness each subformula ϕ_i by a particular run r_{h-i}, so we would like to have that

$$e_{\phi_h}^{r_0,0} \wedge \bigwedge_{j=1}^{h} \bigvee_{i=1}^{k} e_{\phi_{h-j}}^{r_j,i}$$

which states that ϕ_h holds at $(r_0, 0)$ and ϕ_{h-j} holds at some point i of r_j, plus

$$\bigwedge_{j=0}^{h-1} \bigwedge_{i=0}^{k-1} e_{\phi_{h-j}}^{r_j,i} \Rightarrow (r_j(i)_{h-j} \wedge H(r_j(i), r_{j+1}(0)))$$

i.e., p_{h-j} holds at (r_j, i) and r_{j+1} is a branch extending from (r_j, i). This gives an encoding that is of size $O(h \cdot k)$ rather than exponential. (We remark that since we only seek to *construct one* counter-example, rather than *detect all*, the converse implications are not needed.)

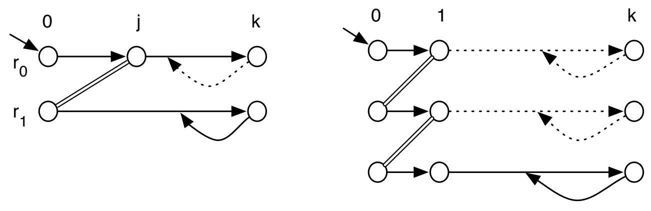


Fig. 1. Shapes of Counterexamples

We also note that a further optimization opportunity arises from considerations concerning loops and fairness conditions. Consider the specification $\psi_3 = AGAF\neg p$ in a system with fairness condition. Following the BMC approaches describe above, the expected shape of the counterexample is shown in the left graph of Figure 1. (Lines with arrows indicate transitions, double lines indicate identity of states.) However, we can in fact drop the cycle on the run fragment r_0, since we need it only to justify fairness of states up to the point j where we switch to the run fragment r_1, which contains a cycle that justifies fairness of all states that can reach this cycle. A similar consideration applies to the specification $AXAX\neg p$, whose counter-example form, consisting of 3 run fragments, is depicted on the right in Figure 1. However, not only can we drop loops here, we in fact need only the first step of the first two fragments.

4.2 Encoding

We now develop a new encoding for bounded model checking ACTLK$_n$. The encoding is based on the optimization ideas discussed above.

The encoding uses three types of resources: singleton run fragments, acyclic run fragments and cyclic run fragments, for which we use the symbols s, a, c respectively. We use the symbol r to represent an instance of any type of resource. Each resource r is associated with a length, denoted $|r|$, which depends on the BMC parameter. For BMC parameter k, singleton run fragment has length 1, and acyclic and cyclic run fragments have length k. We use the term *point* to represent a pair (r, n) where r is a resource and $n < |r|$ is an index, up to the length of that resource. Intuitively, these points will correspond to points of the system being model checked.

A *context* is a triple $R = (n_\mathsf{s}, n_\mathsf{a}, n_\mathsf{c})$ of numbers, representing the index number of the state, acyclic run fragment and cyclic run fragment that is the next free resource available for consumption by the encoding. For a resource type $T \in \{\mathsf{s}, \mathsf{a}, \mathsf{c}\}$, we write $new_T(R)$ for the resource of type T with index n_T. We treat contexts as elements of the lattice $\mathbb{N}^3$, with basis $\mathsf{s} = (1, 0, 0)$, $\mathsf{a} = (0, 1, 0)$, $\mathsf{c} = (0, 0, 1)$. Thus, we can represent a situation in which we consume one singleton run fragment in context R as consuming $new_\mathsf{s}(R)$ and changing the context to $R + \mathsf{s}$.

The encoding of each formula α consumes some number of each type of resource. We denote these numbers by $u^k_\mathsf{s}(\alpha)$, $u^k_\mathsf{a}(\alpha)$ and $u^k_\mathsf{c}(\alpha)$, respectively. The definitions of these functions can be read from the encoding rules: we describe how this can be done after we have given these rules. Using the lattice representation, we also write $u^k(\alpha) = (u^k_\mathsf{s}(\alpha), u^k_\mathsf{a}(\alpha), u^k_\mathsf{c}(\alpha))$.

We work with "obligations", which are tuples $\langle r, n, \psi, R\rangle$ representing that the encoding is required to contain components sufficient to express that formula ψ holds at the point (r, n) in the counter-example, with the encoding operating with respect to context R. The latter is used to determine precisely which resource instances will be used in the encoding.

Associated to these obligations are atomic propositions of the form $e^{r,n}_\psi$. We call these atomic propositions *skeleton variables*. (We note that because the encoding shares resources for disjunctive cases, a run fragment may be a potential witness for several different subformulas, so we do not take the extra step of indexing run fragments by subformulas.) In general, satisfying the obligation $\langle r, n, \psi, R\rangle$ recursively requires the introduction of new obligations, and a formula that relates the atomic proposition $e^{r,n}_\psi$ to other atomic propositions in the encoding. This formula uses several other atomic propositions as abbreviations of various formulas that need to be represented: $b^{r,n,r',n'}_H$ expresses $H(r(n), r'(n'))$, $b^{r,n,r',n'}_{H_i}$ expresses $H_i(r(n), r'(n'))$, $b^{r,n,r',n'}_X$ expresses $T(r(n), r'(n'))$, $b^{r,n}_I$ expresses $I(r(n))$.

We present a set of obligation rewrite rules, parameterized by the BMC parameter k, which represents the maximal run length in the counterexamples to be encoded. Each rule is of the form $o \to_k f, O$, where o is an obligation, f is a propositional logic formula, and O is a set of obligations. Intuitively, this rule means the obligation o can be satisfied by including in the encoding the formula f, but that the additional obligations in O need to be satisfied. The obligation rewrite rules are represented in Table 2. Some of the rules consume resources, at most one item in each case, the type T of which is given in the 3rd column. However, recursive satisfaction of the new obligations introduced by a rule leads to further resource consumption. The total resource consumption when rewriting each type of formula, including this recursive consumption, is given in the second column, which gives the recursive definition of the resource consumption function u^k with range $\mathbf{N}^3$.

The encoding uses several boolean functions of formulas to handle fairness issues. Define $tf(\gamma) = \mathbb{T}$ if γ is in the form $EY\alpha$ with $Y \in \{X, F, G\}$ or $E(\alpha U \beta)$. Then let $tpf(\gamma) = \mathbb{T}$ if $tf(\gamma)$, or $\gamma = \alpha \wedge \beta$ and either $tpf(\alpha)$ or $tpf(\beta)$, or $\gamma = \alpha \vee \beta$ and both $tpf(\alpha)$ and $tpf(\beta)$. Intuitively, this expresses that all ways of satisfying the formula involve satisfying a temporal formula at some point. We also use the boolean variable ϵ to represent that the fairness condition χ in the system is non-trivial, i.e., $\chi \neq \emptyset$. The condition $\epsilon \wedge \neg tpf(\alpha)$ is used to capture situations where fairness constraints need to be applied to states in the present run fragment, but we cannot rely upon the fact that some other run fragment will ensure that all relevant states on the present run are fair.

These rules, which operate on individual obligations, are lifted to *set* rewriting rules, that operate on pairs F, O consisting of a set of propositional formulas F and a set of obligations O, as follows: $F, O \rightarrow_k F', O'$ if there exists an obligation $o \in O$, a rule $o \rightarrow_k f, O''$ and $F' = F \cup \{f\}$ and $O' = (O \setminus \{o\}) \cup O''$.

For a formula ϕ, we start with an initial set of formulas F_0 and set of obligations O_0 that depend on the type of ϕ and the existence of fairness constraints. If $\epsilon \wedge \neg tpf(\alpha)$, then we take r_0 to be the individual state with index 0, and $O_0 = \langle r_0, 0, \phi, \mathtt{s}\rangle$. Otherwise we take r_0 to be the cyclic run fragment with index 0, and $O_0 = \langle r_0, 0, \phi, \mathtt{c}\rangle$. In either case, we take $F_0 = \{e_\phi^{r_0,0}\}$. If there is a sequence of set rewrites $F_0, O_0 \rightarrow_k^* F_k(\phi), \emptyset$ to a pair in which the set of obligations is empty, then we take $F_k(\phi)$ to be the set of propositional logic formulas that represents the semantics of ϕ on the counterexample. Let $\mathcal{B}_k(\phi)$ be the set of skeleton variables occurring in $F_k(\phi)$.

The complete encoding of the model checking problem is then

$$[M, \phi]_k = b_I^{r_0,0} \wedge \bigwedge_{f \in F_k(\phi)} f \wedge Resources_k(\phi) \wedge Encode(\mathcal{B}_k(\phi)).$$

Here $Resources_k(\phi)$ expresses that the resources used in the encoding of ϕ have the proper structure, and $Encode(\mathcal{B})$ expresses that the boolean variables in $\mathcal{B}$ have the intended meaning. More specifically, for each acyclic run fragment r with index $j \leq u_{\mathtt{a}}^k(\phi)$ (cyclic run fragment r with index $j \leq u_{\mathtt{c}}^k(\phi)$), $Resources(\phi)$ contains a conjunct $\mathtt{Runf}_k(r)$ (respectively, $\mathtt{FCRunf}_k(r)$). Similarly, for each $b \in \mathcal{B}_k(\phi)$, the formula $Encode(\mathcal{B}_k(\phi))$ contains a conjunct $b \Rightarrow f$, where f is its intended meaning. For example, for $b = b_H^{r,m,r',m'}$ we include the conjunct $b_H^{r,m,r',m'} \Rightarrow H(r(m), r'(m'))$. See above for the meanings of the remaining cases.

The correctness of the encoding is stated in the following theorem. Note that the bound on the parameter k also establishes termination (in principle) of BMC.

Theorem 1 *Let M be a (finite) system, ψ an ACTLK$_n$ formula. Then $M \not\models \psi$ iff $[M, \phi]_k$ is satisfiable for some $k \leq |M|$, where $\phi = \neg\psi$.*

We can also state a general result on the size of the encoding, compared with the complexity of the previous encodings.

γ	$u^k(\gamma)$	T	f	O	conditions
p_i	0		$e^{r,n}_\gamma \Rightarrow r(n)_i$		$p_i \in Prop$
$\neg p_i$	0		$e^{r,n}_\gamma \Rightarrow \neg r(n)_i$		$p_i \in Prop$
$\alpha \wedge \beta$	$u^k(\alpha) + u^k(\beta)$		$e^{r,n}_\gamma \Rightarrow e^{r,n}_\alpha \wedge e^{r,n}_\beta$	$\langle r, n, \alpha, R\rangle, \langle r, n, \beta, R + u^k(\alpha)\rangle$	
$\alpha \vee \beta$	$max(u^k(\alpha), u^k(\beta))$		$e^{r,n}_\gamma \Rightarrow e^{r,n}_\alpha \vee e^{r,n}_\beta$	$\langle r, n, \alpha, R\rangle, \langle r, n, \beta, R\rangle$	
$EX\alpha$	$\mathsf{c} + u^k(\alpha)$	c	$e^{r,n}_\gamma \Rightarrow b^{r,n,r',0}_X \wedge e^{r',0}_\alpha$	$\langle r', 0, \alpha, R + \mathsf{c}\rangle$	$\epsilon \wedge \neg tpf(\alpha)$
	$\mathsf{s} + u^k(\alpha)$	s	$e^{r,n}_\gamma \Rightarrow b^{r,n,r',0}_X \wedge e^{r',0}_\alpha$	$\langle r', 0, \alpha, R + \mathsf{s}\rangle$	otherwise
$EG\alpha$	$\mathsf{c} + k \cdot u^k(\alpha)$	c	$e^{r,n}_\gamma \Rightarrow b^{r,n,r',0}_H \wedge \bigwedge_{i=0}^{k-1} e^{r',i}_\alpha$	$\langle r', i, \alpha, R + \mathsf{c} + i \cdot u^k(\alpha)\rangle$, $i = 0 \ldots k-1$	
$EF\alpha$	$\mathsf{c} + u^k(\alpha)$	c	$e^{r,n}_\gamma \Rightarrow b^{r,n,r',0}_H \wedge \bigvee_{i=0}^{k-1} e^{r',i}_\alpha$	$\langle r', i, \alpha, R + \mathsf{c}\rangle$, $i = 0 \ldots k-1$	$\epsilon \wedge \neg tpf(\alpha)$
	$\mathsf{a} + u^k(\alpha)$	a	$e^{r,n}_\gamma \Rightarrow b^{r,n,r',0}_H \wedge \bigvee_{i=0}^{k-1} e^{r',i}_\alpha$	$\langle r', i, \alpha, R + \mathsf{a}\rangle$, $i = 0 \ldots k-1$	otherwise
$\overline{K}_i\alpha$	$\mathsf{c} + u^k(\alpha)$	c	$e^{r,n}_\gamma \Rightarrow b^{r',0}_I \wedge \bigvee_{j=0}^{k-1} (b^{r,n,r',j}_{H_i} \wedge e^{r',j}_\alpha)$	$\langle r', i, \alpha, R + \mathsf{c}\rangle$, $i = 0 \ldots k-1$	$\epsilon \wedge \neg tpf(\alpha)$
	$\mathsf{a} + u^k(\alpha)$	a	$e^{r,n}_\gamma \Rightarrow b^{r',0}_I \wedge \bigvee_{j=0}^{k-1} (b^{r,n,r',j}_{H_i} \wedge e^{r',j}_\alpha)$	$\langle r', i, \alpha, R + \mathsf{a}\rangle$, $i = 0 \ldots k-1$	otherwise
$E[\alpha U \beta]$	$\mathsf{c} + u^k(\beta) + (k-1)u^k(\alpha)$	c	$e^{r,n}_\gamma \Rightarrow b^{r,n,r',0}_H \wedge \bigvee_{i=0}^{k-1} (\bigwedge_{j=0}^{i-1} e^{r',j}_\alpha \wedge e^{r',i}_\beta)$	$\langle r', j, \alpha, R + \mathsf{c} + j \cdot u^k(\alpha)\rangle, \langle r', i, \beta, R + \mathsf{c} + (k-1) \cdot u^k(\alpha)\rangle$, $i = 0 \ldots k-1, j = 0 \ldots k-2$	$\epsilon \wedge \neg tpf(\beta)$
	$\mathsf{c} + u^k(\beta) + ku^k(\alpha)$	a	$e^{r,n}_\gamma \Rightarrow b^{r,n,r',0}_H \wedge \bigvee_{i=0}^{k-1} (\bigwedge_{j=0}^{i-1} e^{r',j}_\alpha \wedge e^{r',i}_\beta)$	$\langle r', j, \alpha, R + \mathsf{c} + j \cdot u^k(\alpha)\rangle, \langle r', i, \beta, R + \mathsf{c} + k \cdot u^k(\alpha)\rangle$, $i = 0 \ldots k-1, j = 0 \ldots k-2$	otherwise

Table 2. Obligation rewriting rules $\langle r, n, \gamma, R\rangle \rightarrow_k f, O$, where $r' = new_T(R)$.

Theorem 2 *For our new encoding, the size of* $[M, \phi]_k$ *is* $O(lrk^2)$*, where* r *is the number of consumed run fragments and* l *is the size of formula* ϕ*.*

This is to be compared with a size of $O(lr^{d+2}k^{d+2})$ for the encoding of Penczek and $O(lrk^{d+2})$ for the encoding of Zbrzezny, where d is the depth of nesting of modalities in ϕ.

5 Experimental Results

We argued above that it is possible to obtain an exponential improvement in the size of the encoding, so there are good theoretical grounds to believe that our approach will improve the performance of bounded model checking, particularly as the encoding is an input to a SAT-solver, which deals with an NP-complete problem. In this section we experimentally validate the expectation that our encoding yields a performance improvement over the earlier BMC encodings.

We conducted experiments using several classical multi-agent protocols, varying several aspects of the model checking problem. Each experiment measured runtime as a function of some parameter n of the problem: in some cases n was the number of agents, in others it concerned the depth of nesting, in others it was the size of the state space. Information about protocols, specifications and fairness conditions is listed in Table 3. Here n is the problem scale, NoS is the size of state space, and NoV is the number of state variables. For each of these protocols, we collect data on three specifications. The specifications all have the form $AG(\kappa)$ or $AF(\kappa)$ where κ is a formula that uses epistemic operators, but no temporal operators. For these specifications, we state the depth of modality nesting d. All the specifications are invalid, and we state the number of run fragments NoR in the BMC encoding as a function of the problem scale n and BMC parameter k. The minimal value of the BMC parameter yielding a counterexample is also stated (bound[k]).

Each specification is model checked using a BDD-based model checker (MCK based on CUDD [15] with sifting optimization), and three different BMC encodings: that of Penczek et al (`BMC_P`), Zbrzezny (`BMC_Z`), and our new BMC encoding (`BMC_H`), all implemented as extensions to MCK, so that the inputs to all four algorithms are the same. We included our fairness optimization in the `BMC_Z` implementation.

We report performance results on a 2× 3GHz Quad-core Intel Xeon MacPro with 16GB 667 MHz RAM. (Parallelism in the architecture is not used by the implementation.) BMC performance results are the cumulative timing for all values of the parameter k until a counter-example is found. Since the examples show exponential growth patterns as a function of the problem scale n, we plot results using a log-scale for run-times s. Thus, fitting a line $s = an + b$ to the data corresponds to a model of $O(e^{an})$, and an increase in the slope a corresponds to a polynomial order increase in running time.

The first protocol (DC) is Chaum's Dining Cryptographers [3], a protocol for anonymous broadcast. In this protocol, n agents first share the outcomes of coins they flip in a pairwise fashion around a ring, and then each agent i makes a public announcement determined from the two coinflips for which they know the outcome and a proposition $paid_i$ (representing whether or not they paid for the meal – at most one is assumed

Sys	scale[n]	NoS	NoV	fair.	spec.	depth[d]	valid	NoR[r]	bound[k]
DC	agents	$O(2^{2n})$	$O(n \log n)$		ψ_{dc1}	2	$\mathbb{F}$	2	2
	≥ 3				ψ_{dc2}	2	$\mathbb{F}$	n	2
					ψ_{dc3}	2	$\mathbb{F}$	$k+1$	3
BG	msgs	$O(2n)$	O($n \log n$)	χ_{bg1}	ψ_{bg1}	3	$\mathbb{F}$	3	6
	≥ 2			χ_{bg2}	ψ_{bg2}	n	$\mathbb{F}$	n	$n+3$
					ψ_{bg3}	2	$\mathbb{F}$	$2k+1$	3
PE	length	$O(6n^2)$	$O(\log n)$		ψ_{pe1}	2	$\mathbb{F}$	3	$\lfloor n/2 \rfloor + 2$
	≥ 3				ψ_{pe2}	2	$\mathbb{F}$	$n+2$	$\lfloor n/2 \rfloor + 2$
					ψ_{pe3}	2	$\mathbb{F}$	$k+1$	$2n+1$

Table 3. Parameter values in the experiments

to have paid.) The proposition *stop* is used to indicate completion of the protocol. This protocol is scaled according to the number of agents, i.e., the problem parameter n is the number of agents. The characteristics of this protocol are that the size of its state space is $O(2^{2n})$ and the number of state variables is $O(n \log n)$. The formulas we consider are given in Table 4.

ψ_{dc1}	$AG((stop \wedge \neg paid_0) \Rightarrow K_0(\bigvee_{i=1}^{n-1} paid_i))$
ψ_{dc2}	$AG((stop \wedge \neg paid_0 \wedge odd) \Rightarrow \bigvee_{i=1}^{n-1} K_0 paid_i)$
ψ_{dc3}	$AF(\neg paid_0 \Rightarrow K_0(\bigvee_{i=1}^{n-1} paid_i)))$

Table 4. Specifications for Dinning Cryptographers

Performance results for these formulas are given in Figure 2. (In all these figures, f, g and g_i are propositional logic formulas used as abbreviations.) Counterexamples for these formulas require only small bounds of k but may need a large number of runs.

The second protocol (BG) is the two agent Byzantine Generals Problem, first proposed in [9], in which two agents repeatedly send each other acknowledgements through a lossy channel to increase their mutual knowledge of receipt of a message. This protocol is scaled according to the total number of messages sent by the agents. The characteristics of this protocol are that the size of its state space is $O(2n)$ and the number of state variables is $O(n \log n)$. The formulas for this protocol are given in Table 5, and performance results for these formulas are given in Figure 3.

The third protocol (PE) is a two agent Pursuit-Evasion Game on a very simple discrete linear terrain consisting of positions 0 to n — the pursuer needs to determine if the evader is in the terrain or not, and has perfect visibility on its present location. The game starts with the evader at the rightmost position n and the pursuer at leftmost position 0. The evader moves randomly between position 0 and n, while the pursuer patrols between position 0 and $n-1$. The game ends with a successful capture when they are either at the same position or cross over, exchanging their positions in two successive rounds. This example is scaled according to the length of terrain. The characteristics of this protocol are that the size of its state space is $O(6n^2)$ and the number of state variables is only $O(\log n)$. Formulas for this protocol are given in Table 6. Here *ep* is

ψ_{bg1}	$AG(sndmsg_0 \Rightarrow K_{Alice}K_{Bob}sndmsg_0)$
χ_{bg1}	$\{\neg sndmsg_0 \vee rcvmsg_0, \neg sndack_0 \vee rcvack_0\}$
ψ_{bg2}	$\begin{cases} AG(rcvmsg_{\frac{n}{2}-1} \wedge \neg rcvack_{\frac{n}{2}-1} \Rightarrow (K_{Alice}K_{Bob})^{\frac{n-2}{2}} K_{Alice}rcvmsg_0) & \text{if } n \text{ is even} \\ AG(rcvack_{\frac{n-3}{2}} \wedge \neg rcvmsg_{\frac{n-1}{2}} \Rightarrow (K_{Bob}K_{Alice})^{\frac{n-1}{2}} rcvmsg_0) & \text{if } n \text{ is odd} \end{cases}$
χ_{bg2}	$\begin{cases} \bigcup_{i=0}^{\frac{n}{2}-1}\{\neg sndmsg_i \vee rcvmsg_i, \neg sndack_i \vee rcvack_i\} & \text{if } n \text{ is even} \\ \{\neg sndmsg_{\lfloor\frac{n}{2}\rfloor} \vee rcvmsg_{\lfloor\frac{n}{2}\rfloor}\} \cup \bigcup_{i=0}^{\lfloor\frac{n}{2}\rfloor-1}\{\neg sndmsg_i \vee rcvmsg_i, \neg sndack_i \vee rcvack_i\} & \text{if } n \text{ is odd} \end{cases}$
ψ_{bg3}	$AF(K_{Bob}sndmsg_0 \vee K_{Alice}rcvmsg_0)$

Table 5. Specifications for Byzantine Generals

the Evader's position, pp is the Pursuer's position and n is the length of terrain. Performance results for these formulas are given in Figure 4. The specifications need large but linear bounds to find their counterexamples.

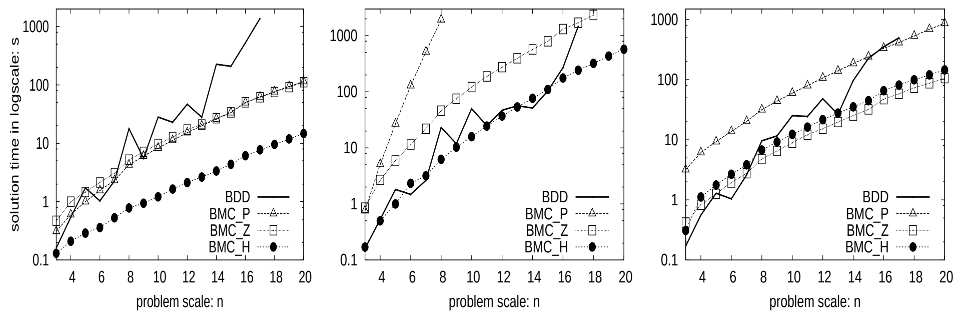
ψ_{pe1}	$AG(found \wedge direction = 0 \Rightarrow (K_{pursuer}ep = pp) \vee (K_{pursuer}ep = pp - 1))$
ψ_{pe2}	$AG(found \Rightarrow \bigvee_{i=0}^{n} K_{pursuer}ep = i)$
ψ_{pe3}	$AF(K_{pursuer}ep = pp)$

Table 6. Specifications for Pursuit Evation Game

In all cases, our new BMC encoding (BMC_H) gives a significant improvement in running time over the Penczek et al encoding (BMC_P). In some cases, we find a constant factor improvement, indicated by parallel curves in the logscale plot with differing initial points. E.g., for ψ_{bg1} and ψ_{bg3} we have roughly a 100-fold speedup, and for ψ_{dc1} and ψ_{dc3} we have roughly a 10-fold speedup. In other cases, we see in the logscale plot roughly linear curves in both cases but with a lower slope for our encoding, implying that for some $c > 1$, the new encoding performs as $f(n)^{1/c}$ where the Penczek et al encoding performs as $f(n)$, e.g., for ψ_{dc2}, ψ_{bg2} and ψ_{pe1}- ψ_{pe3}. In the latter cases, we obtain a very substantial improvement in the scale of example that the method is able to handle in reasonable running times.

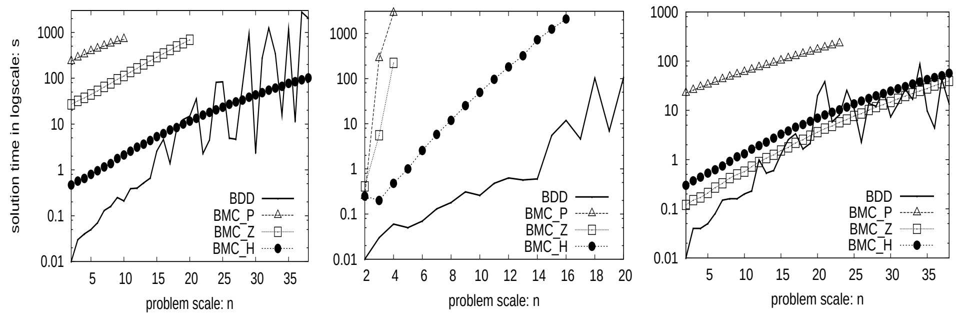
Performance of the Zbrzezny BMC encoding (BMC_Z) is generally intermediate between the Penczek et al BMC encoding and ours. On deeply nested examples (e.g., ψ_{bg2}) our encoding performs significantly better, as expected. However, the depth of nesting does not need to be deep for an order of magnitude improvement to be visible (e.g., ψ_{dc1}, ψ_{dc2}, ψ_{pe1} and ψ_{pe2}). Finally, in some shallowly nested cases (ψ_{dc3}, ψ_{bg3} and ψ_{pe3}) the performance is very similar to ours, and slightly faster by a small factor. (This may be due to the overhead of constructing our slightly more intricate encoding.)

The performance comparison between the bounded model checking approaches and the BDD approach depends on the example. BDD model checking outperforms all the BMC approaches in all the pursuit-evasion game examples. On the Dining Cryptographers example, the BDD model checker initially has comparable performance to BMC_H, but eventually BMC_H wins out, and by more than a constant factor: we did not get termination for the BDD on problems of scale >18, whereas BMC continued to perform steadily in logscale. For the Byzantine Generals, the BDD approach sometimes



$\psi_{dc1} : AG(f \Rightarrow K_0(\bigvee_{i=1}^{n} g_i))$ $\psi_{dc2} : AG(f \Rightarrow \bigvee_{i=1}^{n} K_0(g_i))$ $\psi_{dc3} : AF(f \Rightarrow K_0(\bigvee_{i=1}^{n} g_i))$

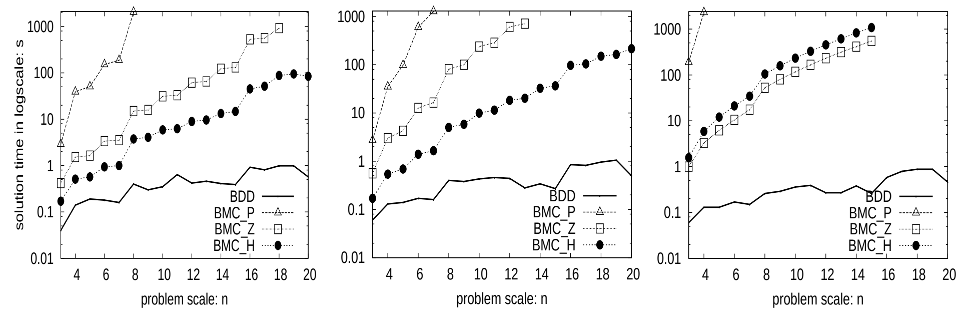
Fig. 2. Dining Cryptographers



$\psi_{bg1} : AG(f \Rightarrow K_A K_B(g))$ $\psi_{bg2} : AG(f \Rightarrow (K_A K_B)^{n/2}(g))$ $\psi_{bg3} : AF(K_A(f) \vee K_B(g))$

$|\chi_{bg1}| = O(1)$ $|\chi_{bg2}| = O(n)$

Fig. 3. Byzantine Generals



$\psi_{pe1} : AG(f \Rightarrow K_P(g_1) \vee K_P(g_2)))$ $\psi_{pe2} : AG(f \Rightarrow \bigvee_{i=1}^{n} K_P(g_i))$ $\psi_{pe3} : AF(K_P(g))$

$|g_i| = O((\log n)^2)$ $|g_i| = O(\log n)$ $|g_i| = O(\log n)$

Fig. 4. Pursuit-Evasion Games

(the deeply nested example ψ_{bg2}) performs significantly better, or (ψ_{bg1} and ψ_{bg3}) performs better on small examples but eventually performs chaotically around our BMC approach, but still better than the older BMC encodings.

6 Related work

Kacprzak et al [7] have previously compared performance of the Penczek et al BMC encoding, as implemented in the model checker Verics [8], with BDD based model checking, implemented in MCMAS [10]. They study the Dining cryptographers protocol. We note that whereas we work from a single common model representation, they need to work with different input representations. For BMC they report only 5 data points, for BDD, up to 11. They conclude that the BDD approach is generally faster, but that BMC may handle larger models. By contrast, we find that with our new encoding, BMC eventually has better performance. (This also seems to hold for `BMC_Z`, though for ψ_{dc2} this is not clear.)

Another comparison of epistemic model checkers is by van Ditmarsch et al [4], who compare MCK, MCMAS and DEMO, principally from the point of view of ease of encoding of specifications of the Russian cards problem. In fact, the encodings developed are somewhat different and are not directly comparable for performance purposes.

7 Conclusion and Future Work

In this paper, we have proposed a new BMC encoding function for fair ACTLK$_n$. Compared with previous encodings [14, 13] whose complexity increases exponentially with respect to the bound k and the number r of runs, the complexity of our encoding is only quadratic on k and linear on r. We conduct experiments on it for several protocols, including Dining Cryptographer, Byzantine Generals, and a Pursuit-Evasion Game. These experiments show that the new encoding often performs much better than the old encodings. The performance comparison with BDD model checking gives mixed results, but we note that unlike BDD model checking, BMC is able to return a counterexample.

For future work, we are investigating generalizing this counterexample-based encoding to some more expressive logics, e.g., an universal fragment of modal μ-calculus with epistemic operators. We have already developed an encoding function for synchronous systems with perfect recall semantics, and will report on its performance elsewhere.

References

1. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.
2. J. P. Burgess. Logic and time. *J. Symb. Log.*, 44(4):566–582, 1979.
3. D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology*, 1(1):65–75, 1988.

4. H.P. van Ditmarsch, W. van der Hoek, R. van der Meyden, and J. Ruan. Model checking russian cards. *Electronic Notes in Theoretical Computer Science*, 149(2):105–123, 2006. Proc. of MoChart 2005.
5. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
6. P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In Rajeev Alur and Doron Peled, editors, *CAV*, volume 3114 of *LNCS*, pages 479–483. Springer, 2004.
7. M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. *Fundam. Inform.*, 72(1-3):215–234, 2006.
8. M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Wozna, and A. Zbrzezny. Verics 2007 - a model checker for knowledge and real-time. *Fundam. Inform.*, 85(1-4):313–328, 2008.
9. L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
10. A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 682–688. Springer, 2009.
11. X. Luo, K. Su, A. Sattar, and M. Reynolds. Verification of multi-agent systems via bounded model checking. In A. Sattar and B. H. Kang, editors, *Australian Conf. on Artificial Intelligence*, volume 4304 of *LNCS*, pages 69–78. Springer, 2006.
12. R. van der Meyden and K. Wong. Complete axiomatizations for reasoning about knowledge and branching time. *Studia Logica*, 75(1):93–123, 2003.
13. W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In *AAMAS*, pages 209–216. ACM, 2003.
14. W. Penczek, B. Wozna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundam. Inform.*, 51(1-2):135–156, 2002.
15. Fabio Somenzi. CUDD: CU Decision Diagram Package. *http://vlsi.colorado.edu/~fabio/CUDD*.
16. B. Wozna. ACTLS properties and bounded model checking. *Fundam. Inform.*, 63(1):65–87, 2004.
17. A. Zbrzezny. Improving the translation from ECTL to SAT. *Fundam. Inform.*, 85(1-4):513–531, 2008.