

Model Checking and Synthesis for Optimal Use of Knowledge in Consensus Protocols (Preliminary Report)

Kaya Alpturer  

Princeton

Gerald Huang  

UNSW

Ron van der Meyden  

UNSW

Abstract

Logics of knowledge and knowledge-based programs provide a way to give abstract descriptions of solutions to problems in fault-tolerant distributed computing, and have been used to derive optimal protocols for these problems with respect to a variety of failure models. Generally, these results have involved complex pencil and paper analyses with respect to the theoretical “full-information protocol” model of information exchange between network nodes. It is equally of interest to be able to establish the optimality of protocols using weaker, but more practical, models of information exchange, or else identify opportunities to improve their performance. Over the last 20 years, automated verification and synthesis tools for the logic of knowledge have been developed, such as the model checker MCK, that can be applied to this problem. This paper concerns the application of MCK to automated analyses of this kind. A number of information-exchange models are considered, for Simultaneous and Eventual variants of Byzantine Agreement under a range of failure types. MCK is used to automatically analyze these models. The results demonstrate that it is possible to automatically identify optimization opportunities, and to automatically synthesize optimal protocols. The paper provides performance measurements for the automated analysis, establishing a benchmark for epistemic model checking and synthesis tools.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Logic and verification; Theory of computation → Verification by model checking; Theory of computation → Distributed algorithms

Keywords and phrases Logic of Knowledge, Model Checking, Synthesis, Consensus Protocol, Fault-tolerance

Digital Object Identifier 10.4230/LIPIcs...

Funding Alpturer was supported in part by AFOSR grant FA23862114029. The Commonwealth of Australia (represented by the Defence Science and Technology Group) supported this research through a Defence Science Partnerships agreement.

1 Introduction

Reasoning about multi-agent communicating systems can be subtle, particularly in settings where agents and the communication media they use can be faulty. Amongst the abstractions that have been developed to deal with the complexities in this area are formal logics of



© The authors;
licensed under Creative Commons License CC-BY 4.0



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

knowledge [18], which provide a succinct way to express properties of the states of information of agents operating in such environments. It has been shown that for coordination goals, such as consensus [34], formulas of the logic of knowledge can express the precise conditions in which agents can act to achieve these goals [17, 32, 22]. Incorporating these formulas into programs run by the agents yields what are called *knowledge-based programs* [18]. It has been shown that knowledge-based programs can exactly characterize the behaviour of *optimal* solutions for certain coordination tasks, independently of the failure environment in which agents operate. This has led to the development of optimal protocols for these tasks in a range of failure environments [17, 32, 6]. The development of such results, however, requires non-trivial analysis to determine concrete predicates of the local state of the agents that correspond to the situations in which agents have the requisite knowledge in a given failure environment. A protocol that substitutes these equivalent predicates for the knowledge formulas in a knowledge-based program is called an *implementation* of the knowledge-based program.

The application of this type of logical analysis of fault-tolerant distributed computing has been conducted primarily in theoretical pencil and paper work. However, the applicability of logics of knowledge has motivated work on the development of automated verification tools using such logics. In particular, there has been work since the early 2000's to develop *model checkers* for the logic of knowledge [19, 30]. Such a model checker takes as input a model of a (typically, finite state) model of a protocol, and the environment in which it operates, as well as specification in the logic of knowledge and time, and automatically determines whether the protocol satisfies this specification in the given environment. One of these model checkers, MCK, has moreover been extended to enable automated synthesis, from knowledge-based programs, of distributed protocol implementations [23, 24].

The problems to which epistemic model checkers have been applied have included security protocols [37, 4], computer hardware protocols [9] and a range of applications in artificial intelligence [25]. While fault-tolerant distributed computing played a key role in the emergence of the logic of knowledge as an area of study in computer science, there has been comparatively little work on the application of epistemic model checkers to this area. It is the objective of this paper to begin a systematic study to address this lacuna.

We consider the application of the model checker MCK to the analysis of consensus protocols, the most well-developed area for application of the logic of knowledge in fault-tolerant distributed computing. Specifically, we focus on variants of the problem of Byzantine Agreement [34]. In this problem, a set of agents, some of which may be faulty, and each of which has a preferred value for some decision to be made, must come to a consensus agreement amongst the non-faulty agents on one of the initial preferences. In the Simultaneous variant of this problem (SBA) the agents must reach this decision simultaneously (in the same round of computation). Solutions to the problem can be characterized using a knowledge-based program in which agents agree upon a value when the non-faulty agents have *common knowledge* of the fact that a particular value is the initial preference of at least one agent [17]. Other knowledge based programs express solutions to Eventual Byzantine Agreement (EBA), the variant where decisions are not required to be simultaneous [22, 6]

The theoretical literature on the application of the logic of knowledge to Byzantine Agreement has tended to focus on “full-information protocols”, in which agents repeatedly send their complete local state to all other agents, and record in their local state all messages that they receive. This assumption enables an analysis of the problem in which one develops protocols

that are optimal in the sense that agents make their decision at the earliest possible time, and such that no protocol (with any other approach to information exchange) can systematically decide no later, and sometimes earlier. While the full-information approach is useful for this theoretical purpose, it is not necessarily practical, since since the full-information state of an agent grows at an exponential rate over time in Byzantine settings, and still quadratically when optimized in benign failure settings. Indeed, in some circumstances, the optimal solutions must, moreover, perform intractable computations at each step of the protocol [33].

Practical protocols, therefore, will operate with simpler state spaces and a lesser exchange of information than a full-information protocol. Nevertheless, it remains of interest to determine the optimality of such a protocol with respect to the reduced information exchanged. That is, we can ask whether a protocol is optimal in its decision time, amongst protocols following the same rule for information exchange. A formal definition for this notion of optimality was given for Eventual Byzantine Agreement by Alpturer, Halpern and van der Meyden [6], who give a number of protocols for this problem that are optimal in this sense. We can similarly define optimality for Simultaneous Byzantine Agreement with respect to a fixed information exchange. It is shown in [36] that the knowledge-based program of [17] also characterizes SBA protocols that are optimal in this sense.

Reduced information-exchanges are also of benefit for model checking purposes, since complexity in the state space of a model is one of the main factors impacting the computation run time of a model checker. Our contribution in this paper is to apply MCK in an experiment on the feasibility of model checking the logic of knowledge as an approach to understanding Byzantine Agreement protocols with limited information exchange. We do the following:

- We develop formal models in the MCK scripting language of a number of protocols for Byzantine Agreement, with limited information exchanges that have been proposed in the literature, as well as the failure models in which those protocols operate. We consider the following protocols:
 - The FloodSet protocol of Lynch [31]
 - A protocol from [13] in which agents additionally maintain a count of the number of agents from which they received a message in the most recent round.
 - A protocol from [13] in which agents additionally maintain a count of the number of agents from which they received a message in the most recent round, as well as the previous value of that count.
 - The concrete protocol of [17] derived from an analysis of knowledge in the full information protocol for SBA in the crash failures model.
 - Protocols from [6] for the problem of Eventual Byzantine Agreement.

These models take as parameters a number of agents, an upper bound on the number of faulty agents, and the number of possible values for the decision. In this version of the paper, we consider both the crash and the sending omissions failures models.

- We apply MCK to determine automatically whether decisions are made in these protocols at the earliest time that the agents achieve the required state of knowledge. We report results on how running times of the MCK model checking experiments scale in the above-mentioned parameters.

XX:4 Model Checking and Synthesis for Consensus

- In the course of these experiments, we identify a number of situations where protocols, as proposed in the literature, are not optimal with respect to their chosen information exchange. This means that there exist opportunities to optimize these protocols, by having agents make their decisions earlier.
- To better understand these optimization opportunities, and derive optimal protocols, we use MCK to automatically synthesize implementations of the knowledge-based program for Byzantine Agreement with respect to the information exchanges and failure models considered. We report results on how running times of the MCK model checking experiments scale in the above-mentioned parameters. We informally describe the optimal protocols synthesized, in small instances, but do not attempt to give a full characterization or a proof of optimality in the general case.

Taken together, the results of this paper show that, at least on small scale models, it is feasible to investigate the question of optimality of a fault tolerant distributed protocol by means of model checking using the logic of knowledge. Moreover, it is also possible to automatically synthesize optimal implementations of knowledge based programs, for small scale models.

The structure of the paper is as follows. We begin in Section 2 by laying out the semantic framework for the logic of knowledge in which we work. Section 3 describes the failure models we consider. In Section 4 we give the formal specification for the Byzantine Agreement (BA) problem. The knowledge-based characterization of optimal solutions to the Simultaneous BA problem is described in Section 5. A knowledge-based characterization of optimal solutions to the Eventual BA problem is described in Section 6. The model checker MCK is described in Section 7. Section 8 describes a number of protocols from the literature for solving SBA, and the qualitative results that we obtain from our application of MCK model checking and synthesis to these protocols using the characterization of Section 5. Section 9 presents a similar treatment for protocols for EBA. In Section 10 we describe the running times of our experiments. Section 11 describes related work and Section 12 concludes with a discussion of limitations of our results and possible future directions for research. An appendix gives an example of the MCK synthesis input scripts used in our work, and the result synthesized by MCK.

2 Semantic Model

To model distributed systems semantically, we first introduce the abstract interpreted systems model, with respect to which the logic of knowledge and time has semantics. Some more specific instantiations of this model are described in the following section, in which we describe how it can be used to model scenarios in which agents communicate by message passing and are subject to failures of various sorts.

Interpreted systems [18] model multi-agent scenarios in which some set \mathbf{Agt} of agents communicate and change their states over time. An interpreted system is a pair $\mathcal{I} = (\mathcal{R}, \pi)$, where \mathcal{R} is a set of runs, describing how the system evolves over time, and $\pi : \mathcal{R} \times \mathbb{N} \rightarrow \mathcal{P}(Prop)$ is an interpretation function, that indicates which atomic propositions are true at each *point* of the system, represented by a pair (r, m) where $r \in \mathcal{R}$ is a run and $m \in \mathbb{N}$ is a natural number representing a time. Each run $r \in \mathcal{R}$ is represented as a function $r : \mathbb{N} \rightarrow S_e \times \prod_{i \in \mathbf{Agt}} L_i$, where S_e is a set, representing the possible states of the environment in which the agents operate, and where each L_i is a set, representing the possible *local states* of agent i . Given a

run r , agent i and time m , we write $r_i(m)$ for the $i + 1$ -st component (in L_i) of $r(m)$, and $r_e(m)$ for the first component (in S_e).

Starting with a set of atomic propositions $Prop$, we can build up a logic, extending propositional logic, by introducing various types of modal operators. Propositions are formulas, and, given a formula ϕ , we have the following formulas: $K_i\phi$ saying that agent i knows that ϕ holds, and $\nu X(\phi(X))$ saying that the current situation is in the largest fixed point of an operator on sets of points defined by the formula ϕ . (This is essentially the greatest fixed point operator from the linear-time mu-calculus [38], extended to interpreted systems. It is required that propositional variable X occur only in positive position for this operator to be meaningful.)

The semantics of the logic is given by a relation $\mathcal{I}, (r, m) \models \phi$, where \mathcal{I} is an interpreted system, (r, m) is a point of \mathcal{I} , and ϕ is a formula, defined inductively as follows (we omit the obvious cases for the propositional operators):

- $\mathcal{I}, (r, m) \models p$ if $p \in \pi(r, m)$, for $p \in Prop$,
- $\mathcal{I}, (r, m) \models K_i\phi$ if for all points (r', m') of \mathcal{I} such that $r_i(m) = r'_i(m')$ we have $\mathcal{I}, (r', m') \models \phi$.
- If $\mathcal{I} = (\mathcal{R}, \pi)$, then $\mathcal{I}, (r, m) \models \nu X(\phi(X))$ if (r, m) is in the largest set of points S of \mathcal{I} such that $F(S) = S$, where $F(S)$ is defined to be the set of points (r', m') such that $(\mathcal{R}, \pi[X \mapsto S]), (r', m') \models \phi(X)$. Here $\pi[X \mapsto S]$ is the interpretation defined by $\pi[X \mapsto S](r, m) = (\pi(r, m) \setminus \{X\}) \cup \{X \mid (r, m) \in S\}$ for each point (r, m) .

The intuition for the definition of the knowledge operator is that $r'_i(m) = r_i(m)$, says that agent i considers it possible, when in the actual situation (r, m) , that it is in situation (r', m') , since it is in the same local state there. An agent then *knows* ϕ is ϕ is true in all the situations that the agent considers to be possible.

Using these operators, we can define a notion of common belief, that operates with respect to an *indexical set* N of agents, which differs from point to point in the system. That is, we assume that there is a function N mapping each point of the system to a set of agents. The semantics of the atomic formula $i \in N$ is given by $\mathcal{I}, (r, m) \models i \in N$ if $i \in N(r, m)$.

An agent may not know whether it is in a set N . We can define a notion of belief, relative to the indexical set N , by $B_i^N\phi = K_i(i \in N \Rightarrow \phi)$. Using this, we define the notions of “everyone in N believes” $EB_N\phi = \bigwedge_{i \in N} B_i^N\phi$. Common belief, relative to an indexical set N , is defined by $CB_N\phi = EB_N\phi \wedge EB_N^2\phi \wedge \dots$. Equivalently, $CB_N\phi = \nu X(EB_N(X \wedge \phi))$. It is immediate from the fixpoint characterization that $CB_N\phi \equiv EB_NCB_N\phi$. Provided it is valid that $N \neq \emptyset$, we have that $EB_N\phi \Rightarrow \phi$ and $CB_N\phi \Rightarrow \phi$.

3 Communication and Failure Models

For the problems we consider, it is convenient to consider a two-layer protocol model comprised of an information exchange protocol \mathcal{E} as the base layer, over which we run a decision protocol P . Nonfaulty agents run both the information exchange and decision protocols correctly, but faulty agents may deviate from these protocols, in ways that depend on the failure model. A further parameter is the failure model \mathcal{F} . Thus, our systems are denoted as $\mathcal{I}_{\mathcal{E}, \mathcal{F}, P}$, but we may elide the parameters when they are clear from the context.

XX:6 Model Checking and Synthesis for Consensus

For the communications model we assume message passing as the communications medium. Each agent may send messages from a set \mathcal{M} . Our focus will be on *synchronous* message passing, in which agents operate in a sequence of synchronized rounds. In each round, each agent sends a set of messages to the other agents, receives some of the messages from the other agents that were sent in the same round, and updates its state depending on these events. The information exchange protocol \mathcal{E} describes the possible initial states of each agent (which may include information such as the agent's preference for the outcome of the consensus decision to be made), how it chooses the messages to be sent in each round (which may depend on its local state and the action performed in the round), and how it updates its state in response to its actions and the messages received in a round.

The failures model describes what failures can occur. Typically, a failures model comes with a parameter t that indicates the maximum number agents that may be faulty. The possibility of solutions to consensus problems generally depends on t in some way, e.g., for Byzantine failures, solutions for consensus using deterministic protocols are only possible when $3t$ is less than the number of agents n [34]. A *failure pattern*, or *adversary*, F expresses the failures that actually occur in a particular run. Each failure model is associated with a set of adversaries. We consider the following failure models, parameterized by an upper bound of t on the number of faulty agents:

- **Crash(t)**: in this model, agents may fail by crashing. When an agent crashes in round m , it sends an arbitrary subset of the messages it was supposed to send in round m . In later rounds, it sends no more messages.
- **Sending-Omissions(t)**: a faulty agent may fail to send any message that it was supposed to send, but receives all messages that have been sent to it.

(The knowledge based program we consider can also be applied to other failure models, such as receiving and general omissions failures.)

A decision protocol P is a function from the agent's local state to its action in the next round.

To connect the above sketch to the semantic model of Section 2, we describe how an information exchange protocol \mathcal{E} , a failure model \mathcal{F} and a decision protocol P determine an interpreted system $\mathcal{I}_{\mathcal{E},\mathcal{F},P}$. In this system, states of the environment S_e record a failure pattern F from \mathcal{F} . An *initial global state* consists of a failure pattern F for the environment state, and for each agent i , an initial state of the information exchange protocol in L_i . For each initial global state, a run r with that initial state is uniquely determined by the information exchange protocol \mathcal{E} , the failure model \mathcal{F} , and the decision protocol P . In each round of this run, the action protocol P determines what action each agent performs in the round, and the information exchange protocol \mathcal{E} determines what messages agents are required to send. The failure pattern F determines which of these messages are actually sent, and which are actually received. As a result of the action taken and the messages received, each agent updates its local state as required by the information exchange protocol \mathcal{E} . The set of runs of $\mathcal{I}_{\mathcal{E},\mathcal{F},P}$ consists of all runs generated in this way from some initial global state.

4 Specifications

We consider consensus problems in which each of the agents in the system is required to make a decision on a value in some set V , and the agents are required to ensure that they make the

same decision. There exists a range of definitions of this problem in the literature. We focus first on a version where *nonfaulty* agents are required to make their decisions consistently and *simultaneously*. We later consider a version where they may decide at different times.

The local states L_i of agent i contain the agent's initial preference $init_i \in V$ for the decision to be made, as well as components derived from the messages that the agent has received. (Details of these components and how the agent updates them depend on the specifics of the information exchange protocol. We give a number of examples below.) Each agent i has, at the start of a run, an initial value $init_i \in V$ representing its preference for the decision to be made. The possible actions of agent i are `noop` and `decidei(v)` for each $v \in V$, representing agent i 's action of making the decision on value v . We write $decides_i(v)$ for the proposition stating the next action that agent i performs according to its decision protocol is `decidei(v)`.

Some of the agents may be faulty; we write N for the set of nonfaulty agents. We note N is *indexical*, in the sense that different runs have different values for the set N . The set of faulty agents, and how they fail, is given in the adversary F of the run. We consider the following requirements:

- **Unique-Decision:** Each agent i performs an action `decidei(v)` (for some v) at most once.
- **Simultaneous-Agreement(N):** If $i \in N$ and $decides_i(v)$ then, at the same time, $decides_j(v)$ for all $j \in N$.
- **Validity(N):** If $i \in N$ and $decides_i(v)$ then $init_j(v)$ for some agent j .

A protocol guaranteeing these properties is said to be a *Simultaneous Byzantine Agreement* (SBA) Protocol. (We remark that termination is not a requirement of this specification. There exist knowledge-based characterizations of the problem where various termination and alternative agreement and validity properties are used, but we do not delve into these alternatives in the present paper.)

Optimality: A concern in the literature has been to develop SBA protocols that are optimal in the sense that decisions are made as early as possible. In order to compare two protocols, we need to be able to compare runs of the two protocols. This has typically been done in the literature by comparing protocols (with an arbitrary information exchange) with the full information protocol, and showing that the full information protocol is optimal [17, 32, 18]. We consider here a definition of optimality that compares protocols using the *same* information exchange. A definition of this kind was first given in [6] for Eventual Byzantine Agreement protocols. We consider here a similar definition from [36] for SBA protocols.

We rely here on the fact that the transitions are deterministic once deterministic protocols P and \mathcal{E} and an initial global state (containing an adversary F that resolves all nondeterminism) have been identified. Given an information exchange protocol \mathcal{E} and a failure model \mathcal{F} , we say that a run r in $\mathcal{I}_{\mathcal{E},\mathcal{F},P}$ of a decision protocol P *corresponds* to a run r' in $\mathcal{I}_{\mathcal{E},\mathcal{F},P}$ of decision protocol P' , if r and r' have the same initial global state. This means that all agents start with the same initial preferences, and face the same pattern of failures over the two runs.

Using this, we can define an order on decision protocols P, P' with respect to an information exchange protocol \mathcal{E} and failure model \mathcal{F} . Define $P \leq_{\mathcal{E},\mathcal{F}} P'$ if for all corresponding runs r

XX:8 Model Checking and Synthesis for Consensus

of $\mathcal{I}_{\mathcal{E},\mathcal{F},P}$ and r' of $\mathcal{I}_{\mathcal{E},\mathcal{F},P'}$, and for all agents i , if agent i decides at time t in r then agent i does not decide earlier than t in run r' .

A decision protocol P is *optimum* for SBA relative to \mathcal{E} and \mathcal{F} , if it is an SBA protocol for \mathcal{E} and \mathcal{F} , and for all SBA protocols P' for \mathcal{E} and \mathcal{F} , we have $P \leq_{\mathcal{E},\mathcal{F}} P'$. That is, P always makes decisions no later than any other SBA protocol for \mathcal{E} and \mathcal{F} .

A decision protocol P is *optimal* for SBA relative to \mathcal{E} and \mathcal{F} , if it is an SBA protocol for \mathcal{E} and \mathcal{F} , and for all SBA decision protocols $P' \leq_{\mathcal{E},\mathcal{F}} P$ we have $P \leq_{\mathcal{E},\mathcal{F}} P'$. That is, there is no decision protocol for \mathcal{E} and \mathcal{F} in which decisions are made no later than they are in P , and sometimes earlier.

5 Knowledge-Based Program - Simultaneous Byzantine Agreement

For the problem of Simultaneous Byzantine Agreement, we work with the knowledge-based program for SBA from [32, 18]. However, whereas these works focus on full information implementations, we will also consider weaker models of the underlying information flow.

For a value $v \in V$, we write $\exists v$ for $\bigvee_{i \in \text{Agt}} \text{init}_i = v$. The results of [32, 18] lead to the following knowledge-based decision program \mathbf{P} , for each agent i :

```
do noop until  $\exists v \in V (B_i^N CB_N \exists v)$ ;  
let  $v$  be the least value in  $V$  for which  $B_i^N CB_N \exists v$  (1)  
in  $\text{decide}_i(v)$ 
```

This program characterizes the conditions under which decisions are made in a way that abstracts from the concrete details of how the environment operates, how information exchange protocol states are maintained, and how agents compute what they know. (The program can be shown to be equivalent to a program in [17], for the crash failures model, that uses instead the knowledge conditions $K_i CK_{\mathcal{A}} \exists(v)$, where \mathcal{A} is the set of agents that are *active*, that is, have not yet crashed.

An implementation P of such a knowledge-based program, relative to an information exchange protocol \mathcal{E} and a failure model \mathcal{F} , is obtained by substituting concrete predicates of the local state of each agent i for the knowledge conditions $B_i^N CB_N \exists v$, that are equivalent to these knowledge conditions in the interpreted system $\mathcal{I}_{\mathcal{E},\mathcal{F},P}$.

The knowledge-based program is shown to be correct in [32] the sense that if any SBA protocol exists in a full information exchange context, then an implementation of the program solves this problem in that full information exchange context. Moreover, an implementation of this program is optimum in this context. Concrete implementations using the full information exchange protocol have been derived for crash failures [17] and for omission failures [32].

In our model checking and synthesis work, we model this knowledge based program in a variety of failure environments and information exchange protocols. A correctness and optimality result can be established that generalizes that of [17, 32]. Whether we obtain an optimal or an optimum protocol as the implementation of the knowledge based program depends on the information exchange protocol \mathcal{E} . We say that \mathcal{E} *does not transmit information about actions* if the messages transmitted by an agent, and the state updates that the agent performs, do not depend on the action the agent performs in the round. We say that \mathcal{E} *does not transmit information about decisions* if the messages transmitted by an agent, and the state updates that the agent performs, may depend on whether the agent performs *noop* or a

$\text{decide}_i(v)$ action, but are the same for all actions $\text{decide}_i(v)$ and $\text{decide}_i(v')$ for $v, v' \in V$. For proof of the following result, see [36].

► **Proposition 1.** *Let \mathcal{E} be an information exchange protocol and let \mathcal{F} be failure model, and let protocol P be an implementation of \mathbf{P} relative to \mathcal{E} and \mathcal{F} .*

- *If \mathcal{E} does not transmit information about actions, then P is an optimum SBA decision protocol relative to \mathcal{E} and \mathcal{F} .*
- *If \mathcal{E} does not transmit information about decisions, then P is an optimal SBA decision protocol relative to \mathcal{E} and \mathcal{F} .*

6 Knowledge Based Program - Eventual Byzantine Agreement

In the Eventual Byzantine Agreement (EBA) problem, the Simultaneous Agreement requirement is replaced by the following property:

- **Agreement(N):** If $i, j \in N$ and $\text{decided}_i(v)$ and $\text{decided}_j(v')$ then $v = v'$.

This allows nonfaulty agents to make their decision at different times, but they are still required to agree on the decisions that they make.

We work with the following knowledge based program \mathbf{P}^0 for EBA in the sending omissions failures model from [6], which is similar to a knowledge based program for the crash failures model in [12, 13]. It is shown in [6] that, subject to some technical side conditions on the information exchange \mathcal{E} , implementations of this knowledge based program with respect to are optimal EBA protocols with respect to \mathcal{E} . The side conditions state, in effect, that (1) the only way that an agent can learn that some agent has $\text{init}_i = 0$ is through a chain of messages, with some agent deciding 0 at each point of the chain, and (2) if an agent considers it possible that some agent is deciding 0, then it considers it possible both that it is nonfaulty itself and that some *nonfaulty* agent is deciding 0. Intuitively, these conditions are satisfied in information exchanges, like the flooding protocol, in which agents explicitly transmit information about 0 values that they learn, but never store enough information about messages received to be able to make deductions about which agents are faulty.

■ Algorithm 1 \mathbf{P}_i^0

```

repeat
if  $\text{init}_i = 0 \vee K_i(\bigvee_{j \in \text{Agt}} j\text{decided}_j = 0)$  then  $\text{decide}_i(0)$ 
else if  $K_i(\bigwedge_{j \in \text{Agt}} \neg(\text{deciding}_j = 0))$  then  $\text{decide}_i(1)$ 
else noop
until  $\text{decided}_i$ 

```

Another, more complex, knowledge based program is also considered in [6], which does not require these side conditions, and also yields an optimal EBA protocol for the sending omissions failure model with respect to the full information exchange. This knowledge based program adds some cases to the program \mathbf{P}^0 , expressed using common knowledge, that can only be satisfied when the information exchange allows agents to learn which agents are faulty. Since we do not consider the full information exchange in the present paper, but only information exchanges that satisfy the side conditions for \mathbf{P}^0 , we focus on this simpler protocol.

7 The Model Checker MCK

Model checking is an automated verification technology [14] applicable to concurrent systems. Given, as input, code for the behaviour of a concurrent system and a formula in a modal logic, a model checker will automatically check whether the formula is true of all runs of the system. The modal logic used in model checking has traditionally been a form of temporal logic. *Epistemic* model checkers [19, 30] extend this technology to specifications in a logic of knowledge and time. MCK [19], the model checker we use in this paper, supports linear time and branching time temporal logic, knowledge operators, as well as least and greatest fixpoint operators including the operator ν defined above.

Concurrent systems are described in MCK by describing an environment, declaring the agents in the system and the protocols run by each of the agents. The environment is described by declaring variables for the environment states in S_e , specifying (by means of a formula or code) which of these states are initial states, and giving nondeterministic code that computes the transitions on the environment states. This code may take as input the actions performed by the agents in a round of the computation. Since these actions are treated as being performed simultaneously, but transition code execution is treated as taking a single step of computation, the MCK modelling language is in the class of *Synchronous languages*. All variable types in MCK (boolean, enumerated or finite numerical ranges) are finite; this implies that the model is finite state.

A description of an agent's protocol includes the binding of the protocol's input parameters to environment variables, a declaration of additional local variables, and code that describes the action performed at each round of the computation of the system. Actions may be a sequence of assignments to local and parameter variables or a signal sent to the environment as an input to the environment transition code.

Some of the agent's variables may be declared to be *observable*. The collection of values of the agent's observable variables makes up the agent's *observation*. A number of different semantics for the knowledge operators are supported in MCK, depending on the degree of recall that an agent has of its observations. In the present paper, we apply the *clock* semantics, which states that an agent's local state in L_i is a pair $(time, o)$, where $time \in \mathbb{N}$ is the current time, and o is the observation that the agent makes at that time. A formal description of how the MCK modelling language relates to the interpreted systems model is given in the MCK manual [20].

In addition to model checking, MCK now supports automated synthesis of implementations of knowledge based programs. In a synthesis input to MCK, agent protocols may declare boolean *template variables*, which function as placeholders for predicates over the agent's local variables, which will be automatically synthesized. The template variables may be used in conditions within the agent's protocol code. For each template variable x , there is a *requirement* that relates x to a formula of the logic of knowledge. This enables knowledge based programs such as the program **P** above to be expressed in MCK.

Synthesis algorithms for a number of different semantics of knowledge have been developed [26, 27, 28]. In the present work, we apply the algorithm for the clock semantics of knowledge. In this case, the requirement for agent i 's template variable x (at the time that the template variable is used in the agent's protocol), is of the form $x \Leftrightarrow \phi$ where ϕ is a boolean combination of formulas of the form $K_i\phi$, which may not contain temporal operators, but may contain knowledge operators and fixpoint operators. The formulas $B_i^N CB_N \exists v$ satisfy

these constraints, so we may write the knowledge based program for SBA in MCK’s input language. An example of an MCK script providing a synthesis input for the SBA problem (for the FloodSet information exchange described below) is given in Appendix A.

Theoretical results of [18] imply that for the clock semantics, knowledge based programs, subject to the constraints imposed by MCK, have a unique implementation. MCK’s synthesis algorithms automatically compute the predicates on the agent’s local states that, when substituted for the template variables, yields the concrete protocol that implements the knowledge based program. Both model checking and synthesis algorithms in MCK are implemented using Order Binary Decision Diagram techniques [11].

8 Information Exchange Protocols for SBA

We now describe the information exchange protocols that we have modelled in MCK, and with respect to which we have conducted an evaluation of MCK for model checking and synthesis from knowledge based programs. The present section describes information exchanges we have considered for the SBA problem, the next section considers information exchanges for EBA. In the present version of the paper, we report results for the crash and sending omissions failures model. In the following, n denotes the number of agents in a scenario, and t denotes the maximum number of these agents that may crash during the running of the protocol (so $t \leq n$). It is well-known that in some runs, a decision cannot be made before round $t + 1$, so protocols often defer a decision to that round.

As protocols are usually modelled, decisions in round $t + 1$ are made after all the messages from that round have been received. In our modelling, to determine an agent’s knowledge at the end of a round, we capture an agent’s state at the end of round m as the state at time m , so the decision would be made in round $t + 2$, as a function of knowledge computed at time $t + 1$. We are interested in determining the earliest possible decision time in each run, given the information being exchanged by the protocol.

8.1 FloodSet Protocol

The FloodSet protocol is described in Section 6.2.1 of Lynch’s text *Distributed Algorithms* [31] (simplifying ideas from other protocols in the literature). Each agent maintains a set of values that it has seen. Initially, each agent will only have seen its own initial value. In each round, each non-faulty agent broadcasts the set of all values that they have historically seen, and updates their set of values by adding all of the values in messages received in the current round. A decision is made on the lowest value received by the end of round $t + 1$.

In our modelling, the local state of agent i has the form $\langle w, time \rangle$, consisting of an array $w : V \rightarrow Bool$ indicating which values have been seen, and the current time $time$ (number of rounds executed). Based on the presentation in [31], one expects that the earliest time at which the condition $B_i^N CB_N \exists v$ holds for some value v is $t + 1$. Our model checking experiments automatically identify a situation in which this *false* in some runs. For example, it is not true in the case of $n = 3$ and $t = 2$.

We have conducted a theoretical analysis which shows that in the case $t \geq n - 1$, the common knowledge condition $B_i^N CB_N \exists v$ holds for some value v already at time $n - 1$, so a decision can be made earlier. This results in a revised hypothesis for the earliest time at which the

XX:12 Model Checking and Synthesis for Consensus

condition $B_i^N CB_N \exists v$ holds with respect to this information exchange, namely

$$(t \geq n - 1 \wedge time = n - 1) \vee (t < n - 1 \wedge time = t + 1) \quad (2)$$

The model checking experiments we have conducted support this hypothesis: it is reported as true in all the instances we were able to check.¹

When we apply MCK to automatically synthesize an implementation of the knowledge based program \mathbf{P} , we find that it synthesizes a decision condition that is equivalent to condition (2) in all the cases we were able to check. (The appendix gives an example of an MCK synthesis script and the result produced by the model checker.)

8.2 Counting the number of crashed agents

A number of variants of the information exchange in the FloodSet protocol are described by Castañeda et al. [13], in a study of a range of early stopping conditions. The specification considered in that paper is for *Eventual Byzantine Agreement* (EBA) — we consider the information exchange instead for the simultaneous specification SBA to determine its usefulness in that case.

One of these variants sends the same messages as in the floodset protocol, but has each agents also keep a count of the number of agents that it knows have *actually* crashed. Since a message is broadcast in each round, for a pair of agents i and j , if agent i does not receive a message from agent j in a round, then i knows that j has crashed. Each agent maintains a variable *count* that is updated in each round to be the number of messages received by the agent in the round. An agent is treated as sending itself a message in each round. An agent's local state in this model has the form $\langle w, count, time \rangle$, where w and *time* are as in the FloodSet model.

In view of the conclusions above about the FloodSet information exchange, a reasonable null hypothesis is that condition (2) captures the earliest time at which $B_i^N CB_N \exists v$ holds for some v , so that a decision can be made. Model checking shows this to be *false*, indicating that that this information exchange protocol gives the agents extra information that allows an earlier stopping time.

Plainly, if $count \leq 1$ (something that is possible only if $t \geq n - 1$), then this implies that at most one message has been received by the agent. Since the agent *always* receives their own message, this implies that every other agent must have crashed and so, it is safe to make a decision at the current round. Correspondingly, when just one agent remains, there is common knowledge amongst the nonfaulty agents reduces to that agent's own knowledge, and common knowledge of a value necessarily holds. This gives an immediate early exit condition: $count \leq 1$.

For this model, both our model checking and synthesis experiments confirm that the earliest time at which the common knowledge condition for SBA holds is when

$$count \leq 1 \vee (t \geq n - 1 \wedge time = t) \vee (t < n - 1 \wedge time = t + 1) \quad (3)$$

¹ We have developed a theoretical proof that this condition captures, for all n and t , the earliest time at which the common knowledge condition is satisfied for the FloodSet protocol [5]. Although, to our knowledge, the result has not been stated in this precise form in the literature, we note that a construction in [16] shows that a protocol that is correct for up to $t' = n - 2$ crashes can be transformed into one that is correct for up to $t = n$ crashes. The effect of this construction is a protocol that stops at time $t' + 1 = n - 1$ in this case.

In particular, even condition $count \leq 2$ does not suffice to enable a decision unless the FloodSet condition (2) holds.

Model checking and synthesis can give information only about small numbers n of agents, but we have subsequently validated this result for general n using a theoretical analysis [5].

8.3 Diff: Memory of Count

A further protocol P_{diff} considered by by Castañeda et al. [13], makes use of not just a count of the number of messages received, but also remembers the value of this variable from the round before the last. It is shown that the difference between these values can be used to give an earlier stopping condition for EBA than that obtained by using a count alone. We have also modelled this information exchange. The only modification required to the model for the single count is to add another variable for the earlier round count and to assign the count value to this variable at the start of each round, before determining the number of messages received in the current round.

While for the EBA problem, [13] show that for the crash failures model, the difference between the most recent count of messages received and the previous value of this variable allows early decisions to be made, in our model checking experiments for this information exchange and failure model, we did not find a condition allowing a decision for the SBA problem that is stronger than that for the version with just a single count variable, as in the previous subsection. We have subsequently validated this result using a theoretical analysis for the general case of an arbitrary number of agents [5].

8.4 Dwork-Moses

Finally, we have modelled the protocol derived by Dwork and Moses [17] as a result of an analysis of common knowledge in the full-information protocol. (The protocol is presented in Fig. 2 of that paper.) This protocol works for the set of decision values $\{0, 1\}$. In this model, we do not attempt to represent the full-information state; rather, we represent just the variables of the protocol of Dwork and Moses. These consist of variables F, NF, RF , which are sets of agents, representing the set of agents known to be faulty, the set of agents newly discovered by the agent to be faulty, and the set of faulty agents that the agent has heard about from other agents. Each agent also maintains a variable `exists0` representing whether it is aware that some agent has initial value 0. In each round, the protocol broadcasts the pair $(NF, \text{exists0})$. There is also an integer variable `current_waste` that represents the agent's estimate of the number of rounds that have been *wasted* in the current run, where a round is wasted if more than one agent crashes in the round.

Intuitively, the amount of waste can be used to determine that there has been a *clean* round, meaning a round in which no new failures were detected. After a clean round, all nonfaulty agents have received the same set of values, so are guaranteed to make the same same decision on the least value received. However, to guarantee a simultaneous decision, they must still wait until the existence of a clean round is common knowledge. The condition `current_waste \neq t + 1 - \text{time}` is used to detect the point at which the existence of a clean round is common knowledge.

9 Information Exchanges for Eventual Byzantine Agreement

For the EBA problem, Alpturer et al. [6] discuss two information exchanges \mathcal{E} that satisfy the constraint under which implementations of the knowledge based program \mathbf{P}^0 are optimal EBA protocols with respect to \mathcal{E} .

We describe these exchanges \mathcal{E}^{basic} and \mathcal{E}^{min} in the present section.

9.1 Information Exchange \mathcal{E}^{min}

In the information exchange \mathcal{E}^{min} , agent i 's local state is a tuple $\langle time_i, init_i, decided_i, jd_i \rangle$, where $time_i$ is the time, $init_i$ is the agent's initial value, $decided_i$ records whether the agent has decided, and jd_i , intuitively, records either a value that the agent has heard that some agent has just decided, or \perp . When the agent decides a value v , it sends a message comprised just of the value v to all agents. Otherwise, it sends no message. When such a value 0 is received by agent i , it sets the value of the variable jd_i to 0, otherwise, if a message 1 is received, it sets the value of the variable jd_i to 1, otherwise the value of this variable is \perp .

An implementation of \mathbf{P}^0 with respect to this information exchange is straightforward. Agent i waits until either $init_i = 0$ or $jd_i = 0$, and decides 0 if this becomes true before time $t + 1$. Otherwise, at time $t + 1$, the agent decides 1.

9.2 Information Exchange \mathcal{E}^{basic}

The information exchange \mathcal{E}^{basic} , is an extension of \mathcal{E}^{min} in which agent i 's local state is a tuple $\langle time_i, init_i, decided_i, jd_i, num1 \rangle$, where $time_i$ is the time, $init_i$ is the agent's initial value, $decided_i$ records whether the agent has decided, jd_i again, records either a value that the agent has just heard that some agent has just decided, or \perp . The additional variable $num1$ records a number. When the agent decides a value v , it sends a message comprised just of the value v to all agents. Otherwise, if it has initial value 1, it sends the message $(init, 1)$, or if it has initial value 0, it sends no message. When such a value 0 is received by agent i , it sets the value of the variable jd_i to 0, otherwise the value of this variable is \perp . In each round, the variable $num1_i$ is set to the number of messages of the form $(init, 1)$ that the agent has received in the last round.

It is shown in [6] that this enables an early stopping condition for the EBA problem. An implementation of the knowledge based program has agent i deciding 0 when either $init_i = 0$ or $jd_i = 0$. The agent decides 1 when either $num1_i > n - time_i$ or $jd_i = 1$.

10 Model Checking and Synthesis Performance Results

The running times of our model checking and synthesis experiments are reported in this section. Our experiments were conducted on a machine with 3.7 GHz 6-core Intel Core i5 with 256 KB L2 cache, 9 MB L3 cache and 32 GB memory. (This machine is multicore, but the present version of MCK runs in a single core.) The timeout TO for long running computations was taken to be 10 minutes. (This was selected to obtain a reasonable runtime for our final full experimental run. In our early testing, a few additional cases terminated within 5 hours, but many TO entries in the tables below ran for as much as 2 days without termination.)

10.1 Results for SBA

The following table shows the time required to execute the floodset and count floodset protocols on n agents, t maximum failures, and the number of values fixed at $v = 2$.

n	t	floodset protocol		count floodset protocol	
		model checking	synthesis	model checking	synthesis
2	1	0m0.069	0m0.239	0m0.085	0m0.336
2	2	0m0.085	0m0.344	0m0.097	0m0.569
3	1	0m0.150	0m0.587	0m0.292	0m1.245
3	2	0m0.228	0m1.011	0m0.407	0m2.401
3	3	0m0.278	0m1.408	0m0.538	0m4.511
4	1	0m0.672	0m2.706	0m5.274	0m24.263
4	2	0m2.122	0m8.164	0m23.328	2m0.892
4	3	0m2.264	1m9.663	0m23.059	TO
4	4	0m3.333	5m40.488	0m27.705	TO
5	1	0m6.214	0m25.784	TO	TO
5	2	0m34.635	2m42.015	TO	TO
5	3	0m41.724	TO	TO	TO
5	4	1m8.150	TO	TO	TO
5	5	1m12.180	TO	TO	TO
6	1	1m12.863	TO	TO	TO
6	2	TO	TO	TO	TO

■ **Table 1** Running times for number of agents n , maximum number of faulty agents t

From these results, we see that adding even a single count variable has a significant impact on performance, with model checking and synthesis of the counting version of floodset scaling less well, and timing out at a smaller numbers of agents. Synthesis is more complex than model checking. A similar poor performance is therefore expected for the even more complex Differential Protocol and the Dwork-Moses protocols. To determine the impact of the number of rounds on the performance, we have considered versions in which fewer than the required $t + 1$ rounds are executed. The results for model checking are given in Table 2.

XX:16 Model Checking and Synthesis for Consensus

		differential protocol		Dwork and Moses	
n	t	no. rounds	time	no. rounds	time
2	1	1	0m0.098	0m0.490	
2	1	2	0m0.104	0m0.576	
2	2	1	0m0.110	0m0.490	
2	2	2	0m0.116	0m0.574	
2	2	3	0m0.114	0m0.637	
3	1	1	4m18.072	TO	
3	1	2	5m4.243	TO	
3	2	1	4m17.531	TO	
3	2	2	4m27.026	TO	
3	2	3	4m27.951	TO	
3	3	1	4m23.998	TO	
3	3	2	4m23.384	TO	
3	3	3	4m28.896	TO	
3	3	4	4m24.074	TO	
4	1	1	TO	TO	

■ **Table 2** Running times for model checking SBA, Diff and Dwork Moses protocols.

We see that adding the additional “previous count” variable to the count floodset model results in model checking scaling less well, with models with one fewer agent terminating within the timeout. However, the number of rounds appears to have a minimal impact on the performance.

10.2 Performance Results for EBA

In the case of EBA, we report just the results for synthesis. The optimality result of [6] for the knowledge based program \mathbf{P}^0 applies to the sending omissions model, but this includes the crash failures model. We have therefore modelled both failures models.

		\mathcal{E}^{min}		\mathcal{E}^{basic}	
n	t	crash	omissions	crash	omissions
2	1	0m0.413	0m0.336	0m1.478	0m1.245
2	2	0m0.623	0m0.408	0m3.651	0m4.976
3	1	0m31.219	0m11.371	TO	TO
3	2	2m43.594	0m14.046	TO	TO
3	3	2m58.142	1m6.735	TO	TO
4	1	TO	TO	TO	TO

■ **Table 3** Running times for EBA synthesis

Table 3 gives the performance results for synthesis. It appears that, even though the knowledge based program \mathbf{P}^0 does not involve common knowledge operators, the performance scales less well than in the SBA case. Whereas for SBA, the FloodSet information exchange scaled to examples with 5 agents, in the case of EBA, the similar information exchange \mathcal{E}^{min} scales to just 4 agents before the blowup in computation time. The performance for the information exchange \mathcal{E}^{min} is worse. This is expected because, like the Count information exchange considered for SBA, this protocol has a additional variable that counts the number of messages received by an agent.

We have also modelled receiving omissions and general omissions, and the performance results obtained are similar, with successful computations in the same cases.

11 Related Work

Knowledge based analyses of consensus protocols have generally focussed on full-information protocols, yielding theoretically optimal, but not necessarily practical implementations [17, 22, 32, 33]. Alpturer, Halpern and van der Meyden [6] have previously considered the knowledge based analysis of consensus protocols and optimality with respect to limited information exchange. However, the focus of that paper is on Eventual Byzantine Agreement, where agents do not need to decide simultaneously. The theory underlying knowledge based programs and the existence of their implementations that we draw upon is developed in [18].

Automated synthesis of concurrent systems is an established area with its own workshops [35]. In general the focus is on synthesis of protocols from temporal specifications; early works in this area are [1, 2, 7, 8]. This approach does not guarantee optimality of the resulting protocol in the way that implementing a knowledge based program is able to achieve. There exists some work on synthesis from epistemic specifications [10, 21, 29] but in general, the focus is on specifications about a single agent's knowledge, rather than than forms of knowledge involving multiple agents, like the common knowledge condition that use in the present paper.

A number of epistemic model checkers exist [19, 30, 15], but MCK remains the only such system to address the automated synthesis of implementations of knowledge based programs. Of the range of problems that have been studied using epistemic model checking, closest to the SBA problem we consider here is a work by Al-Bateineh and Reynolds [3] that the considers Byzantine Atomic Commitment problem. Synthesis is not attempted in this work.

12 Conclusion

In this paper, we have investigated the application of epistemic model checking and synthesis simultaneous and eventual versions of Byzantine Agreement. We have shown that this technique, at least on some small examples, is able to automatically identify situations where the information being exchanged in a protocol provides an opportunity to make a decision earlier than in the protocol as originally designed, leading to implementations that are optimal relative to that information exchange.

One disappointing aspect of our results is that both model checking and synthesis time out at a low number of agents, with a dramatic blowup of performance at the threshold. To some extent, a blowup is expected, since the information exchanges we have considered inherently involve $n(n - 1)$ messages per round. The model checking algorithms that we use for this

exercise use Binary Decision Diagrams, which tend to scale to models involving just 100-200 variables in the boolean representation.

Temporal model checking usually scales better using SAT-based bounded model checking techniques, but because we need to check a complex common knowledge fixpoint, and negative occurrences of knowledge operators, this technique cannot be applied to our problem. However, we remark that we are able to model check the purely temporal specification of SBA using MCK's SAT-based model checking, with significantly better scaling. For example, model checking the SBA specification on the $n = 5$, $t = 4$ model of the Dwork-Moses protocol takes just 2 minutes 5 seconds.

Nevertheless, our results demonstrate in principle feasibility, and motivate future research aimed at improving upon the benchmarks for epistemic model checking and synthesis set in our experiments.

Various directions for future work suggest themselves from this. One is to consider consensus protocols with linear messaging - conceivably, the BDD blowup experienced in the present study will be ameliorated in such protocols. Another is to develop alternate epistemic model checking and synthesis algorithms that scale better on these examples. Of particular interest would be parametric epistemic model checking and synthesis techniques that address the issue of inherent quadratic scaling of the model checking inputs. As we have written generators for arbitrary size models, one of the outcomes is a set of challenge problems for work on epistemic model checkers.

References

- 1 M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 16th Int. Colloquium on Automata, Languages and Programming*, volume 372, pages 1–17. Lecture Notes in Computer Science, Springer-Verlag, July 1989.
- 2 F. Afrati, C. Papadimitriou, and G. Papageorgiou. The synthesis of communication protocols. In *PODC '86: Proc. 5th ACM symposium on Principles of Distributed Computing*, pages 263–271, 1986.
- 3 Omar I. Al-Bataineh and Mark Reynolds. Epistemic model checking of distributed commit protocols with byzantine faults. In Stefania Gnesi, Nico Plat, Nancy A. Day, and Matteo Rossi, editors, *Proceedings of the 7th International Workshop on Formal Methods in Software Engineering, FormaliSE@ICSE 2019, Montreal, QC, Canada, May 27, 2019*, pages 51–60. IEEE / ACM, 2019. URL: <https://doi.org/10.1109/FormaliSE.2019.00014>, doi:10.1109/FormaliSE.2019.00014.
- 4 Omar I. Al-Bataineh and Ron van der Meyden. Epistemic model checking for knowledge-based program implementation: An application to anonymous broadcast. In *Proc. SecureComm*, pages 429–447, 2010.
- 5 K. Alpturer, R. van der Meyden, S. Ruj, and G. Wong. Optimality of simultaneous byzantine agreement with limited information exchange. submitted for publication, 2024.
- 6 Kaya Alpturer, Joseph Y. Halpern, and Ron van der Meyden. Optimal eventual byzantine agreement protocols with omission failures. In *Proc. ACM Symp. on Principles of Distributed Computing, PODC*, pages 244–252. ACM, 2023. doi:10.1145/3583668.3594573.
- 7 A. Anuchitanukul and Z. Manna. Realizability and synthesis of reactive modules. In *Computer-Aided Verification, Proc. 6th Int'l Conference*, pages 156–169, Stanford, California, June 1994. Springer-Verlag, Lecture Notes in Computer Science 818.

- 8 P. C. Attie and E.A. Emerson. Synthesis of concurrent systems with many similar sequential processes. In *Proc. 16th ACM Symposium on Principles of Programming Languages*, pages 191–201, Austin, January 1989.
- 9 Kai Baukus and Ron van der Meyden. A knowledge based analysis of cache coherence. In *Proc. ICFEM*, pages 99–114, 2004.
- 10 S. Bensalem, D. Peled, and J. Sifakis. Knowledge based scheduling of distributed systems. In *Time for Verification, Essays in Memory of Amir Pnueli*, volume 6200 of *Lecture Notes in Computer Science*, pages 26–41. Springer, 2010.
- 11 J. R. Burch, E. M. Clarke, D. L. Dill, J. Hwang, and K. L. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–171, 1992.
- 12 A. Castañeda, Y. A. Gonczorowski, and Y. Moses. Unbeatable consensus. In *Proc. 28th International Conference on Distributed Computing (DISC '14)*, pages 91–106, 2014.
- 13 Armando Castañeda, Yoram Moses, Michel Raynal, and Matthieu Roy. Early decision and stopping in synchronous consensus: A predicate-based guided tour. In *Networked Systems - 5th International Conference, NETYS 2017, Proc.*, pages 206–221, 2017. doi:10.1007/978-3-319-59647-1\16.
- 14 E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.
- 15 P. Dembinski, A. Janowska, P. Janowski, W. Penczek, A. Pólrola, M. Szreter, B. Wozna, and A. Zbrzezny. Verics: A tool for verifying timed automata and Estelle specifications. In *Proc. Conf. Tools and Algorithms for the Construction and Analysis of Systems*, pages 278–283, 2003.
- 16 Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, 1990.
- 17 C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine environment: crash failures. *Information and Computation*, 88(2):156–186, 1990.
- 18 R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning About Knowledge*. The MIT Press, 1995.
- 19 P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In Rajeev Alur and Doron Peled, editors, *Proc. Conf. on Computer Aided Verification (CAV)*, volume 3114 of *LNCS*, pages 479–483. Springer, 2004.
- 20 MCK User Manual. Available from <http://www.cse.unsw.edu.au/~mck>.
- 21 Susanne Graf, Doron Peled, and Sophie Quinton. Achieving distributed control through model checking. *Formal Methods in System Design*, 40(2):263–281, 2012.
- 22 Joseph Y. Halpern, Yoram Moses, and Orli Waarts. A characterization of eventual byzantine agreement. *SIAM J. Comput.*, 31(3):838–865, 2001.
- 23 X. Huang and R. van der Meyden. Symbolic synthesis of knowledge-based program implementations with synchronous semantics. In *Proc. TARK*, pages 121–130, 2013.
- 24 X. Huang and R. van der Meyden. Symbolic synthesis for epistemic specifications with observational semantics. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 455–469, 2014. doi:10.1007/978-3-642-54862-8\39.
- 25 Xiaowei Huang, Patrick Maupin, and Ron van der Meyden. Model checking knowledge in pursuit evasion games. In *Proc. IJCAI*, pages 240–245, 2011.

- 26 Xiaowei Huang and Ron van der Meyden. Symbolic synthesis of knowledge-based program implementations with synchronous semantics. In *Proc. TARK*, 2013.
- 27 Xiaowei Huang and Ron van der Meyden. Symbolic synthesis for epistemic specifications with observational semantics. In *Proc. TACAS*, pages 455–469, 2014.
- 28 Xiaowei Huang and Ron van der Meyden. The complexity of approximations for epistemic synthesis (extended abstract). In *Proc. Workshop on Synthesis, SYNT*, pages 120–137, 2015.
- 29 Gal Katz, Doron Peled, and Sven Schewe. Synthesis of distributed control through knowledge accumulation. In *Proc. Int. Conf on Computer Aided Verification*, pages 510–525, 2011.
- 30 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, *Proc. Conf. on Computer Aided Verification (CAV)*, volume 5643 of *Lecture Notes in Computer Science*, pages 682–688. Springer, 2009.
- 31 N. Lynch. *Distributed Algorithms*. MIT Press, 1996.
- 32 Y. Moses and M. R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988. doi:10.1007/BF01762112.
- 33 Yoram Moses. Optimum simultaneous consensus for general omissions is equivalent to an NP oracle. In Idit Keidar, editor, *Proc. Distributed Computing, 23rd International Symposium, DISC 2009*, volume 5805 of *Lecture Notes in Computer Science*, pages 436–448. Springer, 2009. doi:10.1007/978-3-642-04355-0_45.
- 34 M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.
- 35 Synt workshop series. <https://cgi.csc.liv.ac.uk/~sven/synt/>, 2012-2023.
- 36 R. van der Meyden. Optimal simultaneous byzantine agreement, common knowledge and limited information exchange. manuscript, 2024.
- 37 Ron van der Meyden and Kaile Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proc. CSFW*, pages 280–291, 2004.
- 38 M. Y. Vardi. A temporal fixpoint calculus. In *POPL*, 1988.

A An example of an MCK Synthesis Script

The following is an example of an MCK synthesis script for the FloodSet protocol, for $n = 3$ agents, $t = 1$ possible failures, and $k = 2$ values.

The script begins with a statement selecting the semantics for knowledge to be used in the synthesis process. We used the clock semantics `clk` in order to ensure that there is a unique implementation of the knowledge based program. In this semantics, an agent’s local state consists just of the variables it observes, as well as the clock value (number of rounds). This is followed by a declaration of some enumerated types, and a declaration of variables in the environment of the agents. For efficiency of the model, we have chosen to model the information exchange protocol, including the way that local states are updated, as part of the environment, rather than in the agents’ protocol section.

```
KBP_semantics = clk
```

```
type Crash_Status = {ALIVE, CRASHING, CRASHED}
```

```

type Time = {0..3}
type Values = {0..1}

vote : Values[Agent]
time : Time
w: Bool[Agent][Values]
--old_w stores value of w at the start of a transition
old_w: Bool[Agent][Values]
status : Crash_Status[Agent]
max_crashed : Time
crashed : Time

```

The variable `vote` is an array of values indexed by agents; `vote[i]` represents the initial preference of agent i . Next, the `init_cond` statement gives a formula that constrains the possible initial values of these variables.

```

init_cond =
time == 0 /\ max_crashed == 1 /\ crashed == 0 /\
Forall i:Agent (Forall v:Values ( (w[i][v] <=> vote[i] == v) /\ neg
  → old_w[i][v])) /\
Forall i:Agent ( status[i] == ALIVE )

```

This is followed by a declaration of the three agents in the system, indicating (in quotes) which protocol each runs, and the binding of the parameters of the protocol to variables in the environment.

```

agent D0 "decider" (status[D0],time,w[D0])
agent D1 "decider" (status[D1],time,w[D1])
agent D2 "decider" (status[D2],time,w[D2])

```

Next, we have a `transitions` statement giving code describing how the environment variables are updated in each round. The statement form `[[var | formula(var,var')]]` is a refinement calculus/TLA action style statement, meaning that `var` are the only variables that may change when this statement runs, and its effect is such that the new values `var'` after its execution are related to the values `var` at the start of the execution by the given formula. This statement allows nondeterminism to be represented. Similarly, the `if` construct is a Dijkstra style nondeterministic guarded statement. If more than one guard holds, any corresponding branch may be taken. An example of this is when `status[j] = CRASHING`, when either agent j 's message maybe either delivered (updating `w[i]`), or not (by choosing the `skip` branch).

```

transitions
begin
if time < 2 -> begin
-- make a copy, the values to be transmitted
[[ old_w | Forall i:Agent (Forall v:Values ( old_w[i][v]' <=> w[i][v] ))
  → ]];
-- select the agents that crash, keeping the total number crashed at most
  → max_crashed
for i in Agent do
  begin

```

XX:22 Model Checking and Synthesis for Consensus

```
[[ status[i] | status[i]' in {ALIVE, CRASHING, CRASHED} /\
    (status[i]' == CRASHING => crashed < max_crashed ) /\
    (status[i] == CRASHED <=> status[i]' == CRASHED) ]];
if status[i] == CRASHING then crashed := crashed + 1 else skip
end ;
for i in Agent do
for j in Agent do
    if status[j] in {ALIVE, CRASHING} ->
        for v in Values do w[i][v] := w[i][v] \/ old_w[j][v]
        [] status[j] in {CRASHING, CRASHED} -> skip
    fi;
for i in Agent do if status[i] == CRASHING -> status[i] := CRASHED fi
end fi ;
-- wipe out the old values, to remove correlations in the BDD
[[ old_w | Forall i:Agent (Forall v:Values (neg old_w[i][v]')) ]];
time := time + 1
end
```

The following are the formulas that we verify after synthesis has been performed. These include formulas for the temporal specification for SBA, as well as epistemic formulas that capture the situations in which the agent has common knowledge. The temporal operators used here are from the branching time logic CTL. The operator A represents “on all branches”, the operator G means “at all future times”, X means “at the next moment of time” (after the next round) and exponentiation is used to repeat an operator a give number of times. The keyword “spec_obs” indicates that the observational semantics should be used to interpret the knowledge operators, but since the time is one of the observable variables, this is equivalent to the clock semantics.

```
spec_obs =
    "Agreement: no conflicting decisions by non-failed agents"
    AG ( Forall i:Agent:"decider" (Forall j:Agent:"decider" (
        (status[i] /= CRASHED /\ i.decided /\
        status[j] /= CRASHED /\ j.decided) =>
        (i.decision == j.decision))))

spec_obs = "Uniform Agreement: all agents that decide agree"
    AG ( Forall i: Agent:"decider" (Forall j:Agent:"decider" (
        (i.decided /\ j.decided) => (i.decision == j.decision) )))

spec_obs =
    "Strong Validity: any decision value is the initial vote of some agent"
    AX3 ( Forall i:Agent:"decider" (Forall v:Values (
        (i.decision == v /\ i.decided) => Exists j:Agent (vote[j] == v))))

spec_obs =
    "Termination: all nonfaulty agents eventually decide"
    AX3 ( Forall i:Agent:"decider" (status[i] == ALIVE => i.decided))

spec_obs =
```

```

"agent D0's knowledge test for deciding D0 never holds at time 1"
AX1 neg Knows D0 (status[D0] == ALIVE =>
  (gfp _X (Forall i:Agent (status[i] == ALIVE =>
    (Knows i (status[i] == ALIVE =>
      ( (Exists j:Agent (vote[j] == 0)) /\ _X ))))))))

spec_obs =
  "at time 2, agent D0's knowledge test for deciding 0 is equivalent to
  → the test used by agent D0"
  AX2 (D0.values_received[0] <=> Knows D0 (status[D0] == ALIVE =>
    (gfp _X (Forall i:Agent (status[i] == ALIVE =>
      (Knows i (status[i] == ALIVE =>
        ( (Exists j:Agent (vote[j] == 0)) /\ _X ))))))))

```

Finally, we have the knowledge based program run by the agents. The variables in the parameters of the protocol are aliased to environment variables in the agent declarations above. Variables that are declared `observable` make up the agent's local state for purposes of determining what an agent knows. Variables declared `template` are boolean variables that correspond to “holes” in the protocol that are to be filled by the synthesizer. These variables occur in conditions in the conditional statements in the protocol code. The `define` statements are macros defining some abbreviations. These are used in the `require` statements, which state properties that must be satisfied by the concrete predicates over the observable variables that are constructed by the synthesizer. In the case of this knowledge based program, these properties state that each template variable is equivalent to the agent's knowledge that there is common knowledge amongst the nonfailed (`status = ALIVE`) agents. The operator `X` in these formulas is the linear temporal logic “at the next time” operator.

```

protocol "decider"(status: Crash_Status,
  time: observable Time,
  values_received : observable Bool[Values])

decision : Values
decided : Bool

c_1_0 : template
c_1_1 : template
c_2_0 : template
c_2_1 : template

init_cond = neg decided

define someone_voted0 = Env.vote[D0] == 0 /\ Env.vote[D1] == 0 /\
  → Env.vote[D2] == 0

define someone_voted1 = Env.vote[D0] == 1 /\ Env.vote[D1] == 1 /\
  → Env.vote[D2] == 1

```

XX:24 Model Checking and Synthesis for Consensus

```
define decide_condition0 = Knows Self (status == ALIVE => (gfp _X (
  Forall i:Agent:"decider" (i.status == ALIVE => Knows i (i.status ==
  ↪ ALIVE => someone_voted0 /\ _X))))))

define decide_condition1 = Knows Self (status == ALIVE => (gfp _X (
  Forall i:Agent:"decider" (i.status == ALIVE => Knows i (i.status ==
  ↪ ALIVE => someone_voted1 /\ _X))))))

require = X^1(c_1_0 <=> decide_condition0)
require = X^1(c_1_1 <=> decide_condition1)
require = X^2(c_2_0 <=> decide_condition0)
require = X^2(c_2_1 <=> decide_condition1)

begin
skip;
if (status /= CRASHED /\ neg decided) ->
if c_1_0 then <| decision := 0; decided := True |> else
if c_1_1 then <| decision := 1; decided := True |> else
skip
fi
;
if (status /= CRASHED /\ neg decided) ->
if c_2_0 then <| decision := 0; decided := True |> else
if c_2_1 then <| decision := 1; decided := True |> else
skip
fi
end
```

When MCK runs on this input script, it synthesizes predicates over the observable variables for each of the template variables, and replaces the template variable statements by the following definition statements. These statements contain a disjunct for each agent that is running the knowledge based program. In general, the synthesis result can be different for each agent, but because the present situation is symmetric, the same predicate is synthesized for each agent.

```
define c_1_0 =
  ((Self == D0) /\ False) \/
  (((Self == D1) /\ False) \/
  ((Self == D2) /\ False))

define c_1_1 =
  ((Self == D0) /\ False) \/
  (((Self == D1) /\ False) \/
  ((Self == D2) /\ False))

define c_2_0 =
  ((Self == D0) /\ ((time == 2) /\ values_received[0])) \/
  (((Self == D1) /\ ((time == 2) /\ values_received[0])) \/
```



```
((Self == D2) /\ ((time == 2) /\ values_received[0]))  
  
define c_2_1 =  
  ((Self == D0) /\ ((time == 2) /\ values_received[1])) \/  
  (((Self == D1) /\ ((time == 2) /\ values_received[1])) \/  
  ((Self == D2) /\ ((time == 2) /\ values_received[1])))
```

We see that MCK has calculated that there is not common knowledge of either value at time 1, and that at time 2, an agent has common knowledge that some agent i had an initial value of $v \in \{0, 1\}$ iff `values_received[v]` holds. (This local variable is an alias for the variable `w[i][v]` in the environment.) As a result, the formulas above are all evaluated to hold by the model checker for the concrete program produced by the synthesis process.