JOURNAL OF COMPUTER AND SYSTEM SCIENCES 38, 195–237 (1989)

The Complexity of Reasoning about Knowledge and Time. I. Lower Bounds

JOSEPH Y. HALPERN AND MOSHE Y. VARDI

IBM Almaden Research Center, San Jose, California 95120

Received December 8, 1987

We study the propositional model logic of knowledge and time for distributed systems. We consider a number of logics (ninety-six in all!), which vary according to the choice of language and the assumptions made on the underlying system. The major parameters in the language are whether there is a common knowledge operator, whether we reason about the knowledge of one or more than one processor, and whether our temporal operators are branching or linear. The assumptions on distributed systems that we consider are: whether or not processors forget, whether or not processors learn, whether or not time is synchronous, and whether or not there is a unique initial state in the system. We completely characterize the complexity of the validity problem for all the logics we consider. This paper focuses on lower bounds; a sequel will deal with the corresponding upper bounds. Typical results include a Π_1^1-completeness result for the language with common knowledge with respect to systems where processors do not forget, and a corresponding non-elementary-time result for the language without common knowledge. It is shown that, in general, the assumption that processors do not forget or do not learn greatly increases the complexity of reasoning about knowledge and time.

1. INTRODUCTION

It has been argued recently that knowledge is a useful concept for analyzing the behavior and interaction of processors in a distributed system [CM, DM, FI1, Hal, HF, HM1, LR, MT, PR, RK, Ros]. When analyzing a system in terms of knowledge, not only is the current state of knowledge of the processors in the system relevant, but also how that state of knowledge changes over time. A formal propositional logic of knowledge and time was first proposed by Sato [Sa]; others have since been proposed by Lehmann [Le1], Fagin *et al.* [FHV1], Parikh and Ramanajum [PR], and Ladner and Reif [LR]. Still others are implicit in all the other references cited above.

While Sato proved a nondeterministic exponential upper bound for his logic, Lehmann stated a theorem claiming a doubly exponential upper bound for his logic (which included common knowledge), and Ladner and Reif prove that one of their logics is undecidable. This apparant inconsistency is, of course, due to the fact that all these papers actually consider different logics. To add to the confusion, these papers use the same notation with different interpretations.

0022-0000/89 $3.00

571/38/1-13

In this paper we try to bring some order to this confusion by categorizing logics for knowledge and time along two major dimensions: the language used and the assumptions made on the underlying distributed system. By varying these parameters, we end up with ninety-six logics. (Of course, they are not all of equal interest to distributed computing!) All of the logics considered in the papers mentioned above fit into our framework. Our major results involve completely characterizing the complexity of all these logics, showing how the subtle interplay of the parameters can have a tremendous impact on complexity.

The languages considered in the literature vary according to the modalities used for knowledge and time. As far as knowledge goes, the relevant issue is whether the language can talk about the knowledge of more than one agent, and whether we have a modal operator in the language for common knowledge (where common knowledge of a fact φ holds if everyone knows φ, everyone knows that everyone knows φ, etc.). For time, the question is whether we use *branching time* or *linear time* modalities (which essentially amounts to whether or not we can quantify over the possible executions of a program).

It is well known that if we consider either knowledge or time alone, the language used has a great impact on the complexity of the logic. As was shown by Halpern and Moses [HM2], the complexity of reasoning about knowledge for the notion of knowledge most appropriate for distributed systems (which satisfies the axioms of the modal logic S5), the validity problem for the logic is co-NP-complete if we can only reason about one agent or processor in the language, PSPACE-complete with two or more agents, and EXPTIME-complete if we add common knowledge to the language. If we consider time alone, the validity problem for the language with branching time modalities is EXPTIME-complete [EH1], while for the language with linear time modalities it is PSPACE-complete [SC]. Not surprisingly, we find a similar phenomenon here; the complexity of reasoning about knowledge and time depends on the language used. What is perhaps more interesting is how the assumptions made on the underlying distributed system, which essentially place conditions on the interaction between knowledge and time, affect complexity.

The types of assumptions on the system that are typically made include whether or not processors *forget* (the assumption of no forgetting has also been called *unbounded memory* or *cumulative knowledge* in other papers [FHV2, HV, Mo]), whether or not processors can *learn*, whether or not there is a *unique initial state* in the system, and whether time is *synchronous* or *asynchronous*. We now explain each of these parameters in more detail and motivate them in terms of distributed systems.

We first discuss the notion of knowledge in a distributed system. Although there have been many papers that consider this notion, they all have the same essential features. A distributed system is identified with a set of possible *runs* of the system, where a run is a complete history of the system's behavior over time. Thus, the run may include such things as each processor's initial state and its complete message history (i.e., the messages it has sent and received, in the order they were sent and received, time-stamped if the processors have local clocks). Formally, assume we

have a system of m processors, each of which at any time is in some local state. This local state may encode such things as the processor's initial state, part or all of its message history, and the values of relevant variables. A run is a function from time (which, for simplicity, we assume is discrete and ranges over the natural numbers) to *global states* of the form $\langle l_1, ..., l_m \rangle$, where l_i is the local state of processor i.[1] Given a run r and a time n, we can think of the global state $r(n)$ as a "snapshot" describing the current state of the system. We can think of n as denoting the time on some external global clock (not necessarily observable by the processors). Following [HM1], we call such a pair (r, n) a *point*.

Processor i is said to *know* a fact φ (written $K_i\varphi$) at a given point if φ is true at all other points in which it is in the same state. Intuitively, a processor cannot distinguish two points if it is in the same state in both; thus it knows φ if φ is true at all the points it cannot distinguish from the true state of affairs.[2] We say a processor *considers run r' possible* at point (r, n) if for some n', it cannot distinguish (r, n) from (r', n').

We say a processor *does not forget* if the set of runs the processor considers possible stays the same or decreases over time (intuitively, as a result of the processor getting more information). So if at some point (r, n) in run r processor i considers run r' possible, then run r was indistinguishable from r' at all points in the past. Intuitively, a processor that cannot distinguish two runs that it could distinguish at an earlier time must have "forgotten" the information that allowed it to distinguish those runs. Note that no forgetting intuitively requires unbounded memory, so that a processor can store all the information it has received. Thus, the distinction between forgetting and no forgetting essentially corresponds to whether we view our processors as finite-state machines or Turing machines.

The dual notion to "no forgetting" is "no learning." A processor *does not learn* if the set of runs it considers possible stays the same or increases over time. More formally, if at some point (r, n) processor i considers run r' possible, then processor i will consider run r' possible at all times in the future (i.e., at all points (r, n') with $n' \geqslant n$). If processor i cannot distinguish two points $(r, 0)$ and $(r', 0)$ in a system with no learning and no forgetting, then i goes through the same sequence of states in both r and r', regardless of what messages i may receive. Such a system essentially corresponds to a *non-adaptive* algorithm; a processor does not modify its actions in response to signals from the outside world. In this precise sense, we can say that no learning takes place.

In some systems the assumption is made that each processor has a *unique initial state*. This means that there is a unique initial global state for the system (i.e., for all

[1] In a more general model we might augment the global state to include a component describing the *environment*, which intuitively consists of all the relevant features of the system not described by the processors' local states, such as messages in transit but not yet delivered, and so on (cf. [FHV2]). The environment component plays no role in the complexity analysis, so we omit it here.

[2] This interpretation of knowledge is called a *state-based interpretation* in [HM1], and is essentially the interpretation used in [PR, HF, Ros, RK, FI1]. We will not consider the more general *epistemic interpretations* discussed in [HM1].

runs r and r', the global states $r(0)$ and $r'(0)$ are identical). The assumption of a unique initial state seems fairly innocuous. After all, we can always add a new initial state to every run and then let it develop as it did before. However, as we shall see, this assumption is not so innocuous when combined with the assumption of no learning.

In a *synchronous* system, we assume that a processor has access to a global clock that ticks at every instant of time and the clock reading is part of its state, so the processor always knows the time. Note that protocols that proceed in rounds can be viewed as running in synchronous systems.

An *interpreted system* is a pair (R, π), where R is a system and π is a truth assignment to the primitive propositions at every point of R. There is a straightforward way to extend π to all formulas (the details are discussed in the next section). For the rest of our discussion, it will be useful to have notation for different classes of interpreted systems and different languages. We use $\mathscr{C}$ to represent the class of all interpreted systems. We then use subscripts nf, nl, uis, sync to indicate restrictions to interpreted systems where, respectively, processors do not forget, processors do not learn, where there is a unique initial state, and where the system is synchronous. Thus, for example, $\mathscr{C}_{(\text{nf, sync, uis})}$ represents the class of interpreted systems where processors do not forget, the system is synchronous, and there is a unique initial state.

We use the notations $CKL_{(m)}$, $CKB_{(m)}$, $KL_{(m)}$, and $KB_{(m)}$ to describe the languages we use. The L and the B tell us whether linear time or branching time modalities are used, the presence or absence of C in the name indicates whether or not common knowledge is included, and the subscript indicates the number of agents. Thus, $CKL_{(2)}$ is the language that uses linear time modalities and has modal operators K_1, K_2, and C for the knowledge of agent 1, agent 2, and common knowledge. (We describe the language and give its semantics in detail in the next section.) Similarly, $KB_{(3)}$ is the language that uses branching time modalities and K_i, $i = 1, 2, 3$, but has no modal operator for common knowledge.

The logics that have been considered in other papers can now be classified as follows. Sato [Sa] and Lehmann [Le1] restrict attention to $\mathscr{C}_{(\text{nf,sync})}$: synchronous systems where processors do not forget. Lehmann uses the languages $CKL_{(m)}$; Sato essentially does as well (although his language does not have explicit temporal operators). Halpern and Fagin [HF], Parikh and Ramanujam [PR], and the *tree logic of protocols* of Ladner and Reif [LR] also assume no forgetting, but do not require that time be synchronous, so the class of interpreted systems considered for these logics is $\mathscr{C}_{(\text{nf})}$. On the other hand, these papers differ in the languages they consider: $CKL_{(m)}$ in [HF] and $KB_{(m)}$ in [PR, LT].[3] Ladner and Reif's *linear logic*

[3] Actually, in [PR] there are also modal operators for what is called *implicit knowledge* in [HM1, HM2]. In addition, the branching time modalities used in [PR] and [LR] only give us a subset of the language $KB_{(m)}$ (a different subset in each of the papers). However, these differences have no impact on the complexity, so we do not focus on them further here. Also, the fact that the systems in [LR] are actually trees instead of sets of runs imposes another mild condition that we briefly discuss in the next section. Again, this difference has no impact on complexity.

of protocols, despite the name, also uses (a subset of) branching time, but restricts attention to the class of interpreted systems $C_{(nf,nl,uis)}$, where processors neither forget nor learn, and there is a unique initial state. In the remaining papers that consider formal models of knowledge and time [CM, FI1, Ros, RK], the assumption of no forgetting is not imposed; all the interpreted systems in $\mathscr{C}$ are considered. However, in [Ros, RK] linear time is used, while [CM, FI1] implicitly use branching time, although neither of these latter two papers explicitly has temporal operators in their logics.

We do not discuss here which of these logics is most appropriate. Our feeling is that the choice should be guided by the application at hand (see [Pn, La2, EH2] for a discussion of these issues in the context of linear vs. branching time logics). Instead, we focus our attention on the complexity of the decision procedures for each of them.

At a high level, we can view our results as saying that assuming either no forgetting or no learning tends to make the complexity of reasoning about knowledge and time much worse. For example, if we have common knowledge in the language (and at least two agents, since common knowledge reduces to knowledge if we have only one agent), then the validity problem with respect to many classes of interpreted systems where processors do not forget or do not learn, such as $\mathscr{C}_{(nf)}$ and $\mathscr{C}_{(nl)}$, is wildly undecidable, in fact, $\prod_1^1$-complete. (A precise definition of $\prod_1^1$ appears in Section 3.) This means that there can be no complete axiomatization for these cases (since a complete axiomatization would imply that the set of valid formulas was r.e.).[4] On the other hand, for classes such as $\mathscr{C}$ or $\mathscr{C}_{(sync,uis)}$, where we do not make the assumption that processors do not learn or do not forget, the complexity of the validity problem for the language with common knowledge is (only!) EXPTIME-complete.

A similar situation arises if we consider the language without common knowledge. Although the validity problem in the presence of no forgetting or no learning is in general decidable, it is non-elementary; if we do not make the assumption that processors do not learn or do not forget, the validity problem is either PSPACE-complete or EXPTIME-complete (depending on whether we consider linear time or branching time).

There are some anomalous situations though, mainly those involving the combination of no learning and a unique initial state. For example, Ladner and Reif show that the validity problem for $KB_{(2)}$ is undecidable (even without common knowledge in the language) with respect to $\mathscr{C}_{(nf,nl,uis)}$. An easy extension of their proof shows it is actually $\prod_1^1$-complete; these results also hold for the language $KL_{(2)}$. On the other hand, if we consider the class of interpreted systems $\mathscr{C}_{(nf,nl,sync,uis)}$, where we impose the additional condition of synchrony, the situation

[4] As we remarked above, Lehmann claimed a doubly exponential time decision procedure for his logic, which is $CKL_{(m)}$ interpreted over interpreted systems in $\mathscr{C}_{(nf,sync)}$. He also claimed a complete axiomatization [Le1]. Lehmann later retracted these claims and only claimed these results for the one-agent case, without common knowledge [Le2]. Of course, our results show that the original claims were in fact incorrect.

collapses. The validity problem for this logic is EXPSPACE-complete, even with common knowledge in the language! Intuitively, the reason is that the combination of these assumptions implies that no expressive power is gained by having common knowledge or more than one agent in the language.

Our results are summarized in Fig. 1. The results given in the table are tight: the upper bounds match the lower bounds (to within constant factors). In order to explain the results for the languages $KL_{(m)}$ and $KB_{(m)}$, $m \geq 2$, in the first two rows of the table in a little more detail, we must introduce some notation. Let $\mathrm{ex}(m, n)$ be defined inductively via $\mathrm{ex}(0, n) = n$, $\mathrm{ex}(m+1, n) = 2^{\mathrm{ex}(m,n)}$ (so that, intuitively, $\mathrm{ex}(m, n)$ is a stack of m 2's, with the top 2 having exponent n), let the *alternation depth* of φ, written $\mathrm{ad}(\varphi)$, be the number of alternations of distinct knowledge modalities (K_i's) in φ, and let $|\varphi|$ be the length of φ when viewed as a string of symbols. The nonelementary time bound means that there is an algorithm for deciding if a formula φ is valid which runs in time $\mathrm{ex}(1 + \mathrm{ad}(\varphi), c\,|\varphi|)$, for some constant $c > 0$. Furthermore, any algorithm for deciding validity must run in time $\mathrm{ex}(1 + \mathrm{ad}(\varphi), d\,|\varphi|)$ for some constant $d > 0$ and infinitely many formulas φ. The explanation of the nonelementary space bound is analogous. Note that, by definition, for any formula φ of $KL_{(1)}$ or $KB_{(1)}$ we have $\mathrm{ad}(\varphi) \leq 1$. Thus, the bounds for $KL_{(1)}/KB_{(1)}$ in the first two rows of the table are special cases of the bounds for $KL_{(m)}/KB_{(m)}$. In particular, Lehmann's doubly exponential time upper bound for KL_1 is a special case of ours.

The difference between the nonelementary time bounds in the first row of the table, and the nonelementary space bounds in the second row of the table can roughly be explained by noting that allowing learning gives us the ability to encode alternation. More precisely, when we have no forgetting but allow learning, we can encode alternating Turing machines that run in space $\mathrm{ex}(\mathrm{ad}(\varphi), c\,|\varphi|)$ (which corresponds to time $\mathrm{ex}(\mathrm{ad}(\varphi)+1, c\,|\varphi|)$; once we impose the assumption of no

	$CKL_{(m)}/CKB_{(m)}$, $m \geq 2$	$KL_{(m)}/KB_{(m)}$, $m \geq 2$	$KL_{(1)}/KB_{(1)}$
$\mathscr{C}_{(\mathrm{nf})}$, $\mathscr{C}_{(\mathrm{nf,sync})}$, $\mathscr{C}_{(\mathrm{nf,uis})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$	Π^1_1	Nonelementary (time $\mathrm{ex}(\mathrm{ad}(\varphi)+1, c\,\|\varphi\|)$)	Double-exponential time
$\mathscr{C}_{(\mathrm{nl})}$, $\mathscr{C}_{(\mathrm{nf,nl})}$, $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nl,sync})}$	Π^1_1	Nonelementary (space $\mathrm{ex}(\mathrm{ad}(\varphi), c\,\|\varphi\|)$)	EXPSPACE
$\mathscr{C}_{(\mathrm{nf,nl,uis})}$	Π^1_1	Π^1_1	EXPSPACE
$\mathscr{C}_{(\mathrm{nl,uis})}$	co-r.e.	co-r.e.	EXPSPACE
$\mathscr{C}_{(\mathrm{nl,sync,uis})}$, $\mathscr{C}_{(\mathrm{nf,nl,sync,uis})}$	EXPSPACE	EXPSPACE	EXPSPACE
$\mathscr{C}$, $\mathscr{C}_{(\mathrm{sync})}$, $\mathscr{C}_{(\mathrm{sync,uis})}$, $\mathscr{C}_{(\mathrm{uis})}$	EXPTIME	PSPACE for $KL_{(m)}$, EXPTIME for $KB_{(m)}$	PSPACE for $KL_{(1)}$, EXPTIME for $KB_{(1)}$

FIG. 1. The complexity of the validity problem for logics of knowledge and time.

learning, we can only encode deterministic Turing machines that run in space $\mathrm{ex}(\mathrm{ad}(\varphi), c|\varphi|)$.

Given the number of results, we concentrate on the lower bound proofs in this paper, deferring proofs of upper bounds and complete axiomatizations (in the cases where such axiomatizations are possible) to a sequel. The rest of this paper is organized as follows. The next section describes the languages and the various kinds of interpreted systems discussed above in detail. In Section 3 we present all our lower bound results for the languages $CKL_{(m)}$ and $CKB_{(m)}$, $m \geq 2$, while in Section 4 we consider the situation for the languages without common knowledge. We conclude in Section 5 with some of the philosophical implications of these results.

2. The Formal Model: Language and Systems

The logics we are considering are all propositional. Thus, we start out with primitive propositions p, q, ... and we close the logics under negation and conjunction, so that if φ and ψ are formulas, so are $\sim\varphi$ and $\varphi \wedge \psi$. In addition, we close off under modalities for knowledge and time, as discussed below. As usual, we view *true* as an abbreviation for $\sim(p \wedge \sim p)$, $\varphi \vee \psi$ as an abbreviation for $\sim(\sim\varphi \wedge \sim\psi)$, and $\varphi \Rightarrow \psi$ as an abbreviation for $\sim\varphi \vee \psi$. We assume that $\wedge$ and $\vee$ bind more tightly than $\Rightarrow$, so that we write, for example, $\varphi \Rightarrow \psi \wedge \psi'$ rather than $\varphi \Rightarrow (\psi \wedge \psi')$.

If we have m agents (in distributed systems applications, this would mean a system with m processors), we add the modalities $K_1, ..., K_m$. Thus, if φ is a formula, so is $K_i\varphi$ (read "processor i knows φ"). In some case we also want to talk about common knowledge, so we add the modalities E and C into the language; $E\varphi$ says that everyone knows φ, while $C\varphi$ says φ is common knowledge.

The temporal modalities (sometimes called *operators* or *connectives*) that we use depend on whether we are considering linear time or branching time. In the linear time case, we have a unary operator $\bigcirc$ and a binary operator U. Thus, if φ and ψ are formulas, then so are $\bigcirc\varphi$ (read *nexttime* φ) and $\varphi\, U\, \psi$ (read φ *until* ψ). We view $\Diamond\varphi$ as an abbreviation for *true* $U\, \varphi$, while $\Box\varphi$ is an abbreviation for $\sim\Diamond\sim\varphi$. Intuitively, $\bigcirc\varphi$ says that φ is true at the next point (one time unit later), $\varphi\, U\, \psi$ says that φ holds until ψ does, $\Diamond\varphi$ says that φ is eventually true (either in the present or at some point in the future), and $\Box\varphi$ says that φ is always true (in the present and at all points in the future). In the branching time case, we also have quantifiers over runs, so that if φ and ψ are formulas, so are $\forall\varphi\, U\, \psi$, $\exists\varphi\, U\, \psi$, $\forall\bigcirc\varphi$, and $\exists\bigcirc\varphi$. A formula of the form $\forall\bigcirc\varphi$ is true at the point (r, n) if $\bigcirc\varphi$ is true at (r', n) for all runs r' extending (r, n), where the notion of *extending* will be made precise below. Similarly, $\exists\varphi\, U\, \psi$ is true at (r, n) if $\varphi\, U\, \psi$ is true at (r', n) for some run r' extending r. Again, we view $\forall\Diamond\varphi$ (resp. $\exists\Diamond\varphi$) as an abbreviation for $\forall true U\varphi$ (resp. $\exists true U\varphi$), and $\forall\Box\varphi$ (resp. $\exists\Diamond\varphi$) as an abbreviation for $\sim\exists\Diamond\sim\varphi$ (resp. $\sim\forall\Diamond\sim\varphi$). Thus, for example, $\forall\Diamond\varphi$ is true at the point (r, n) if φ is eventually true for all runs r' extending (r, n). It has been argued that a nexttime

operator ($\bigcirc$) is inappropriate for reasoning about asynchronus systems (cf. [La1]); after all, the processors do not have access to an external clock in such systems, so it is not even clear that the notion of the ticking of such a clock makes sense. We remark that all our lower bounds also hold if the language does not have a nexttime operator.[5]

As we mentioned in the Introduction, we take $|\varphi|$ to be the length of the formula φ viewed as a string of symbols, while in the languages without C and E (i.e., $KL_{(m)}$ and $KB_{(m)}$) we define $\mathrm{ad}(\varphi)$ to be the greatest number of alternations of distinct K_i's along any branch in φ's parse tree. For example, $\mathrm{ad}(K_1 \sim K_2 K_1 p) = 3$; temporal operators do not count, so that $\mathrm{ad}(K_1 \,\Box\, K_1 p) = 1$. Note that $\mathrm{ad}(\varphi) \leqslant |\varphi|$, and if φ is in $KL_{(1)}$ or $KB_{(1)}$, then $\mathrm{ad}(\varphi) \leqslant 1$.

A *system* for m processors consists of a set R of runs, where each run $r \in R$ is a function from $\mathbb{N}$ to L^m, where L is some set of *local states*. Thus, $r(n)$ has the form $\langle l_1, \ldots, l_m \rangle$; such a tuple is called a *global state*. (Formally, we could view a system as a tuple (R, L, m), making the L and m explicit; we have chosen not to do so in order to simplify notation. The L and m should always be clear from context.) An *interpreted system M for m processors* is a tuple (R, π), where R is a system for m processor and π maps every point $(r, n) \in R \times \mathbb{N}$ to a truth assignment $\pi(r, n)$ on primitive propositions (so that $\pi(r, n)(p) \in \{\textbf{true}, \textbf{false}\}$ for each primitive proposition p).

We now give semantics to $CKL_{(m)}$ and $KL_{(m)}$. Given an interpreted system $M = (R, \pi)$, we write $(M, r, n) \models \varphi$ if the formula φ is *true at* (or *satisfied by*) the point (r, n) of interpreted system M. We define $\models$ inductively for formulas of $CKL_{(m)}$ (for $KL_{(m)}$ we just omit the clauses involving C and E). In order to give the semantics for formulas of the form $K_i\varphi$, we need to introduce one new notion. If $r(n) = \langle l_1, \ldots, l_m \rangle$, $r'(n') = \langle l'_1, \ldots, l'_m \rangle$, and $l_i = l'_i$, then we say that $r(n)$ and $r'(n')$ are *indistinguishable to processor i* and write $(r, n) \sim_i (r', n')$. Of course, $\sim_i$ is an equivalence relation on global states. $K_i\varphi$ will be defined to be true at (r, n) exactly if φ is true at all the points whose associated global state is indistinguishable to i from that of (r, n). We proceed as follows:

- $(M, r, n) \models p$ for a primitive proposition p iff $\pi(r, n)(p) = \textbf{true}$
- $(M, r, n) \models \varphi \wedge \psi$ iff $(M, r, n) \models \varphi$ and $(M, r, n) \models \psi$
- $(M, r, n) \models \sim\varphi$ iff $(M, r, n) \not\models \varphi$
- $(M, r, n) \models K_i\varphi$ iff $(M, r', n') \models \varphi$ for all (r', n') such that $(r, n) \sim_i (r', n')$

[5] The G, F, and U operators of [PR] correspond to our $\forall\Box$, $\forall\Diamond$, and $\forall U$, respectively. Parikh and Ramanujam do not have a nexttime operator in their language. The $\Box$, $\Box^*$, $\Diamond$, and $\Diamond^*$ of [LR] correspond to our $\forall\bigcirc$, $\forall\Box$, $\exists\bigcirc$, and $\exists\Diamond$, respectively. Ladner and Reif have neither $\forall\Diamond$ nor an until operator. All our results are easily seen to hold for these restricted languages. We could, of course, also allow more complicated mixtures of modalities, such as $\forall\Box\Diamond\varphi$, as in the logics CTL* [EH2] or MPL [Ab]. Doing this seems to increase the complexity of the decision procedure by at least one exponential (cf. [VS]).

- $(M, r, n) \models E\varphi$ iff $(M, r', n') \models K_i\varphi$ for $i = 1, ..., m$
- $(M, r, n) \models C\varphi$ iff $(M, r', n') \models E^k\varphi$, for $k = 1, 2, ...$ (where $E^1\varphi = E\varphi$ and $E^{k+1}\varphi = EE^k\varphi$)
- $(M, r, n) \models \bigcirc\varphi$ iff $(M, r, n+1) \models \varphi$
- $(M, r, n) \models \varphi\, U\, \psi$ iff there is some $n' \geqslant n$ such that $(M, r, n') \models \psi$, and for all n'' with $n \leqslant n'' < n'$, we have $(M, r, n'') \models \varphi$.

There is a graphical interpretation of the semantics of E^k and C which we shall find useful in the sequel. Fix an interpreted system M. We say a point (r', n') in M is *reachable* from a point (r, n) *in k steps* if there exist points $(r_0, n_0), ..., (r_k, n_k)$ such that $(r, n) = (r_0, n_0)$, $(r', n') = (r_k, n_k)$, and for all $j = 0, ..., k-1$ there exists i such that $(r_j, n_j) \sim_i (r_{j+1}, n_{j+1})$. We say (r', n') is *reachable* from (r, n) if it is reachable in k steps for some k. It is easy to check that $(M, r, n) \models E^k\varphi$ iff $(M, r', n') \models \varphi$ for all points (r', n') reachable from (r, n) in k steps, and $(M, r, n) \models C\varphi$ iff $(M, r', n') \models \varphi$ for all points (r', n') reachable from (r, n).

We remark here that we could have presented the semantics in a slightly different way, more closely related to the standard Kripke semantics for knowledge (see, for example, [HM2]). Instead of associating to each point (r, n) the global state $r(n)$, we could view points as more abstract entities, without this additional structure. An interpreted system would now consist of a set of runs, a truth assignment π, and equivalence relations $\sim_1, ..., \sim_m$ on the points. The semantics of formulas such as $K_i\varphi$ could be defined using these equivalence relations just as above. This approach was taken in an earlier version of this paper [HV] and is taken by Lehmann [Le1]. The two definitions are equivalent in an obvious way: once we associate a global state with each point in such a way that two points are indistinguishable to i iff they are equivalent. We will use this observation in a number of our proofs global state with each point in suchηa way that two points are indistinguishable to i iff they are equivalent. We will use this observation in a number of our proofs below. We have chosen to use global states here in order to emphasize the intuitions coming from distributed systems. This choice also allows us to define branching time semantics in a natural way.

Given an interpreted system $M = (R, \pi)$, we say that $r' \in R$ *extends* the point $(r, n) \in R \times \mathbb{N}$ if $r'(n') = r(n')$ for all $n' \leqslant n$; i.e., if r and r' go through the same sequence of global states up to time n. With this definition, we can now give semantics to branching time formulas as follows:

- $(M, r, n) \models \exists\bigcirc\varphi$ iff $(M, r', n+1) \models \varphi$ for some run r' extending (r, n)
- $(M, r, n) \models \forall\bigcirc\varphi$ iff $(M, r', n+1) \models \varphi$ for all runs r' extending (r, n)
- $(M, r, n) \models \exists\varphi\, U\, \psi$ iff for some run r' extending (r, n) there exists some $n' \geqslant n$ such that $(M, r', n') \models \psi$, and for all n'' with $n \leqslant n'' < n'$, we have $(M, r', n'') \models \varphi$

• $(M, r, n) \models \forall\varphi\, U\, \psi$ iff for all runs r' extending (r, n) there exists some $n' \geqslant n$ such that $(M, r', n') \models \psi$, and for all n'' with $n \leqslant n'' < n'$, we have $(M, r', n'') \models \varphi$.[6]

As usual, we define a formula φ to be *valid with respect to a class $\mathscr{D}$ of interpreted systems* iff $(M, r, n) \models \varphi$ for all interpreted systems $M \in \mathscr{D}$, runs r in M, and times n. A formula φ is *satisfiable with respect to $\mathscr{D}$* iff for some $M \in \mathscr{D}$, r, and n we have $(M, r, n) \models \varphi$. It will often be more convenient for us to consider the satisfiability problem rather than the validity problem in proving lower bounds.

We now turn our attention to formally defining the classes of interpreted systems discussed in the Introduction.

We say processor i *does not forget in* $M = (R, \pi)$ if all runs $r, r' \in R$ and times n, n', k, if $(r, n) \sim_i (r', n')$ and $k \leqslant n$, then there exists $k' \leqslant n'$ such that $(r, k) \sim_i (r', k')$. In order to motivate this definition, define *processor i's history at the point (r, n)* to be the sequence $l_0, \ldots, l_k$ of states that processor i takes on in run r up to time n, with consecutive repetitions omitted. For example, if from time 0 through 4 in run r processor i goes through the sequence l, l, l', l, l of states, its history at $(r, 4)$ just l, l', l. Roughly speaking, processor i does not forget if it "remembers" its history. More precisely we have

LEMMA 2.1. *Processor i does not forget in a system R iff for all runs $r, r' \in R$, if $(r, n) \sim_i (r', n')$ then processor i's history is the same at (r, n) and (r', n').*

Proof. The fact that remembering the history implies no forgetting is immediate from the definition. The converse can be proved by a straightforward induction on $n + n'$. ∎

This lemma shows that no forgetting requires an unbounded number of local states in general, since processor i may have an infinite number of distinct histories in a given system. There is one more observation about systems where processors do not forget that we frequently use; this is captured in the following lemma.

LEMMA 2.2. *If processor i does not forget in R and $(r, n) \sim_i (r, n')$, then $(r, n) \sim_i (r, n'')$ for all n'' with $n \leqslant n'' \leqslant n'$.*

Proof. We proceed by induction on n. Note that since $(r, n) \sim_i (r, n')$ and $n'' \leqslant n'$, by definition of no forgetting there must be some $k \leqslant n$ such that $(r, k) \sim_i (r, n'')$. If $n = 0$, we must have $k = n$. If $n > 0$, then if $k = n$ we are done, while if $k < n$,

[6] The notion of branching time we have defined here differs slightly from that defined in [LR] and an earlier version of this paper [HV]. In these papers, the set of runs has a tree-like structure, which guarantees that the set is *limit closed*. As defined here, the set of runs is not necessarily limit closed, making it more like Abrahamson's MPL [Ab] than CTL (see [Em, EH2] for a detailed discussion of this issues). In our framework, we can say that a set R of runs is limit closed if, for all runs r, the fact that for all n there is a run $r_n \in R$ extending (r, n) implies that $r \in R$. By imposing the additional condition of limit closure on our classes of runs, we get precisely the classes considered in [LR]. This condition has no impact on the complexity of the decision procedure, although it does slightly affect the axioms for the logic. In practice we would not want to impose this condition since it is easier to consider issues of fairness without it.

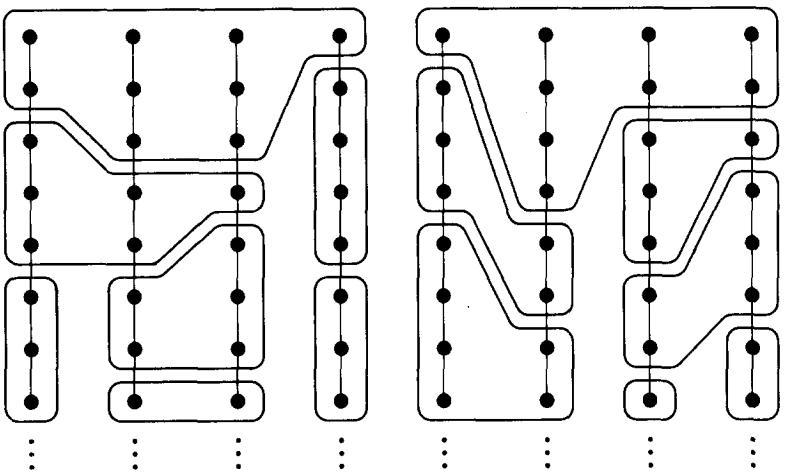

FIG. 2. A system where processor i does not forget.

by the induction hypothesis (where k plays the role of n, n plays the role of n'', and n'' plays the role of n'), it follows that $(r, n) \sim_i (r, k)$, and by transitivity we get $(r, n) \sim_i (r, n'')$. ∎

A system where processor i does not forget is shown in Fig. 2, where the vertical lines denote runs (with time 0 at the top) and all points that i cannot distinguish are enclosed in the same region.

In a system where processor i *does not learn*, we have the opposite situation: If $(r, n) \sim_i (r', n')$, then for all $k \geq n$ there must be some $k' \geq n'$ such that $(r, k) \sim_i (r', k')$. A system where processor i does not forget and does not learn is shown in Fig. 3. With no learning, the equivalence relations do not refine. Note how i goes

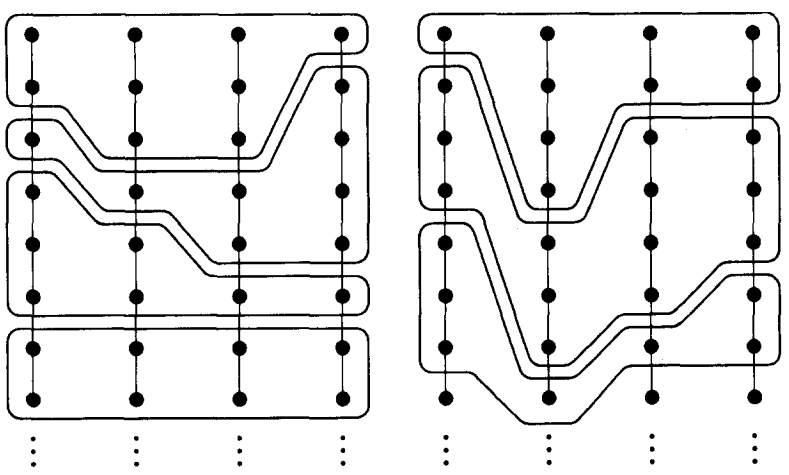

FIG. 3. A system where processor i does not forget and does not learn.

through the same sequence of states in all runs it cannot distinguish (modulo *stuttering*, i.e., the same state repeating at consecutive points). (We remark that if we consider no learning but allow forgetting, the situation is slightly more complicated. If processor i cannot distinguish $(r, 0)$ and $(r', 0)$, then there may be a set S of states such that i is in every state of S infinitely often in both runs r and r', but it does not go through the states in the same sequence in r and r'.)

In a *synchronous* system, we assume that every processor has access to a global clock that ticks at every instant of time, and the clock reading is part of its state. Thus, in a synchronous system, each processor always "knows" the time. More formally, we say a time is synchronous in R if for all processors i and all runs r, r', if $(r, n) \sim_i (r', n')$, then $n = n'$. We remark that in a previous version of this paper [HV], we took a slightly weaker definition: we required that for all runs r, if $(r, n) \sim_i (r, n')$ then $n = n'$. Let us call a system that satisfies that latter condition *weakly synchronous*. Note that the definition of weakly synchronous only considers one run r rather than two runs r and r'. It is easy to show (by induction on n) that the two definitions are equivalent for systems where processors do not forget. However, in general they are different. (We remark that the notion of weak synchrony is important in some of our proofs.) Observe that in a synchronous system where $(r, n) \sim_i (r', n)$, an easy induction on n shows that if i does not forget and $n > 0$, then $(r, n-1) \sim_i (r', n-1)$, while if i does not learn, then $(r, n+1) \sim_i (r', n+1)$.

Finally, we say that a system R has a *unique initial state* if for all runs $r, r' \in R$, we have $r(0) = r'(0)$. Thus, if R is a system with a unique initial state, then we have $(r, 0) \sim_i (r', 0)$ for all runs r, r' in R and all processors i.

We say that $M = (R, \pi)$ is an interpreted system where processors do not forget (resp. processors do not learn, time is synchronous, there is a unique initial state) exactly if R is a system with that property. As we mentioned in the Introduction, we use the notation $\mathscr{C}$ to represent the class of all interpreted systems, and add the subscripts *nf*, *nl*, *sync*, and *uis* to denote particular subclasses of $\mathscr{C}$.

3. Lower Bounds for $CKL_{(m)}$ and $CKB_{(m)}$

In this section we prove the results claimed in the Introduction on the complexity of the validity problem for $CKL_{(m)}$ and $CKB_{(m)}$.

We begin with a brief review of the notions of $\prod_1^1$ and its dual $\sum_1^1$. Further details can be found in [Rog] or any other standard textbook of recursive function theory.

Formulas of *second-order arithmetic with set variables* consist of formulas of first-order arithmetic, augmented with expressions of the form $x \in X$, where x is a number variable and X is a set variable, together with quantification over set variables and number variables. A *sentence* is a formula with no free variables. Second-order arithmetic with set variables is a very powerful language. For example, the following (true) sentence of the language expresses the law of mathematical induction over $\mathbb{N}$:

$$\forall X(0 \in X \wedge \forall x((x \in X \Rightarrow x + 1 \in X) \Rightarrow \forall x(x \in X))).$$

A $\prod_1^1$ sentence (resp. $\sum_1^1$ sentence) of second-order arithmetic with set variables is one of the form $\forall X_1 \cdots \forall X_n \varphi$ (resp. $\exists X_1 \cdots \exists X_n \varphi$), where φ is a formula of second-order arithmetic with set variables that has no quantification over set variables. A set A of natural numbers is in $\prod_1^1$ (resp. $\sum_1^1$) exactly if there is a $\prod_1^1$ sentence (resp. $\sum_1^1$ sentence) $\psi(x)$ with one free number variable x and no free set variables such that $a \in A$ iff $\psi(a)$ is true. $\prod_1^1$-hardness and $\prod_1^1$-completeness are defined in the obvious way (the reduction is via one–one recursive functions). It is well known that $\prod_1^1$-complete sets are not recursively enumerable (cf. [Rog]). In particular, it follows from the fact that the validity problem for both $CKL_{(m)}$ and $CKB_{(m)}$, $m \geq 2$, is $\prod_1^1$-complete that there can be no complete (recursive) axiomatization for these languages.

For all the $\prod_1^1$ lower bound proofs, we use the following result, due to Harel, Pnueli, and Stavi [HPS]. We say that a Turing machine $\mathbf{A}$ is *recurrent* if, when started on the empty tape, it has an infinite computation that reenters its start state infinitely often. Let $\mathbf{A}_0, \mathbf{A}_1, \ldots$ be a recursive enumeration of the nondeterministic Turing machines with one tape, infinite to the right.

PROPOSITION 3.1. *The set* $\{n \mid \mathbf{A}_n$ *is recurrent*$\}$ *is* $\sum_1^1$*-complete.*

We now state and prove our first $\prod_1^1$ lower bound result, which focuses on synchronous systems. We then show how the result can be extended to classes of systems that are not necessarily synchronous. The matching upper bound results are relatively straightforward; the proof can be found in part II of this paper.

THEOREM 3.2. *The validity problem for* $CKL_{(m)}$ *and* $CKB_{(m)}$, $m \geq 2$, *is* $\prod_1^1$*-hard with respect to the following classes of interpreted systems:* $\mathscr{C}_{(\mathrm{nf,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$, $\mathscr{C}_{(\mathrm{nl,sync})}$, *and* $\mathscr{C}_{(\mathrm{nf,nl,sync})}$.

Proof. The idea is to show how to encode the computation of an arbitrary Turing machine in a $CKL_{(2)}$ formula. Roughly speaking, we show that given a one-tape, infinite to the right, nondeterministic Turing machine $\mathbf{A}$, we can construct a $CKL_{(2)}$ formula $\varphi_{\mathbf{A}}$ such that for an interpreted system $M \in \mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,sync,uis})}$ $\mathscr{C}_{(\mathrm{nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,nl,sync})}$) we have $(M, r, 0) \models \varphi_{\mathbf{A}}$ iff, for each n, the "nth level" of M (i.e., the points of M of the form (r', n)) encodes a possible situation after n steps of a computation of $\mathbf{A}$ when started on a blank tape, in a sense to be made clear below.

In order to understand the idea (and the difficulties!) of our construction, it is instructive to recall the proof that two-dimensional temporal logic is $\prod_1^1$-hard [HPV, Har]. In two-dimensional temporal logic, the structures are two-dimensional grids, infinite to the right and downwards. Thus, a point in a structure is just a pair (i, j) of natural numbers. There are four temporal operators: $\bigcirc_{\mathrm{r}}$, $\Box_{\mathrm{r}}$, $\bigcirc_{\mathrm{d}}$, and $\Box_{\mathrm{d}}$. The formula $\bigcirc_{\mathrm{r}} \varphi$ says that φ is true one step to the right, while $\Box_{\mathrm{r}} \varphi$ says φ is true everywhere to the right. Similarly, $\bigcirc_{\mathrm{d}} \varphi$ says that φ is true one step down, while $\Box_{\mathrm{d}} \varphi$ says that φ is true everywhere below the given point. Thus, for example, $(M, i, j) \models \bigcirc_{\mathrm{r}} \varphi$ if $(M, i+1, j) \models \varphi$ and $(M, i, j) \models \Box_{\mathrm{d}} \varphi$ if $(M, i, j') \models \varphi$ for all

$j' \geq j$. It is easy to encode the computation of Turing machines in this structure. Every row represents an ID (instantaneous description); consecutive rows represent consecutive ID's of the computation. Using $\Box_d$ we can easily say the start state appears infinitely often in the computation.

In $CKL_{(2)}$, we can use $\bigcirc$ and $\Box$ to play the same role as $\bigcirc_d$ and $\Box_d$. It might seem that we could use K_1 to play the role of $\bigcirc_r$ and then C could play the role of $\Box_r$. This will not quite work. The problem is that, since $\sim_1$ is an equivalence relation, K_1 has the property that $K_1 p \Rightarrow K_1 K_1 p$. Intuitively, you can not get anywhere by taking many K_1 "steps" that you could not already get to by taking one K_1 step. We solve this problem by using the modal operators K_1 and K_2 together with a special primitive proposition p_Δ (which is used to mark the fact that a change has taken place) to play the role of $\bigcirc_r$. We replace a formula of the form $\bigcirc_r \varphi$ by one of the form $K_1(\sim p_\Delta \Rightarrow K_2(p_\Delta \Rightarrow \varphi))$. Thus, taking a K_1 step to a point where $\sim p_\Delta$ holds, followed by a K_2 step to a point where p_Δ holds, corresponds to taking a $\bigcirc_r$ step. (This is why we need at least two processors to get our $\prod_1^1$ result.) We then use C to play the role of $\Box_r$, as suggested above.

Another difference between systems and structures of two-dimensional temporal logic is that the latter have the "grid" property. For every pair (i, j), $(i+1, j)$ of consecutive nodes at the "jth level," there is a corresponding pair $(i, j+1)$, $(i+1, j+1)$ of consecutive nodes at the $(j+1)$th level. This property is crucial to being able to encode the fact that consecutive ID's "match up right." No forgetting and no learning each give us half of the grid property in synchronous systems. With no forgetting we have $(r, n+1) \sim_i (r', n+1)$ implies $(r, n) \sim_i (r', n)$, while with no learning we have $(r, n) \sim_i (r', n)$ implies $(r, n+1) \sim_i (r', n+1)$. Showing that either half of the grid property suffices to encode the computation of a Turing machine makes up the heart of our proof. (This is exactly why we cannot straightforwardly apply the techniques of [HPV, Har], as is done, for example, in [RS].)

We proceed as follows. For the remainder of this proof, we fix a Turing machine A. Suppose A has state space S and uses tape alphabet Γ. We use the special symbol # to denote the left-hand end of the tape, and b to denote a blank cell of the tape. We assume that $\#, b \notin \Gamma$. Let CD (for *cell descriptor*) be $\Gamma \cup \{b, \#\} \cup (\Gamma \times S)$. Thus, an ID is always of the form $\#xbbb...$, where x is a finite string of symbols in CD. Corresponding to every symbol $c \in$ CD we have a primitive proposition p_c. If $c = \langle \gamma, s \rangle \in \Gamma \times S$ then the primitive proposition p_c is meant to denote a cell of the tape which has symbol γ and is currently being read by the head, while A is in state s. Finally, as mentioned above, we use a special primitive proposition p_Δ to mark the change to a new cell.

φ_A will consist of the conjunction of a number of formulas, which we now describe. Let φ_1 be the formula:

$$\Box C\left(\bigvee_{c \in \mathrm{CD}} \left(p_c \wedge \sim\left(\bigvee_{d \in \mathrm{CD}, d \neq c} p_d\right)\right)\right).$$

Thus, if $(M, r, n_0) \models \varphi_1$, then for all $n \geq n_0$, exactly one cell descriptor holds of any

point reachable from (r, n). (Our comments here and below hold whether M is in $\mathscr{C}_{(\mathrm{nf,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$, $\mathscr{C}_{(\mathrm{nl,sync})}$, or $\mathscr{C}_{(\mathrm{nf,nl,sync})}$.)

Let φ_2 be the formula

$$\Box C((p_A \Rightarrow \Box p_A) \wedge (\sim p_A \Rightarrow \Box \sim p_A)).$$

If $(M, r, n_0) \models \varphi_2$, then the truth value of p_A is constant below any point reachable from (r, n) for $n \geq n_0$.

Next, let φ_3 be the formula

$$\Box C(\sim K_1 p_A \wedge \sim K_2 \sim p_A).$$

If $(M, r, n_0) \models \varphi_3$, then for any (r', n) reachable from (r, n), $n \geq n_0$, there are points (r_1, n) and (r_2, n) such that $(r', n) \sim_1 (r_1, n)$, $(r', n) \sim_2 (r_2, n)$, $(M, r_1, n) \models \sim p_A$, and $(M, r_2, n) \models p_A$.

Given an interpreted system M, a run r in M, and $k \geq 0$, we will say a *level k alternating sequence of runs starting with* (r, n_0) is a sequence $r_0, r_1, \ldots$ of runs such that (a) $r = r_0$, (b) $(r_{2j}, n_0 + k) \sim_1 (r_{2j+1}, n_0 + k)$, (c) $(r_{2j+1}, n_0 + k) \sim_2 (r_{2j+2}, n_0 + k)$, (d) $(M, r_{2j}, n_0 + k) \models p_A$, (e) $(M, r_{2j+1}, n_0 + k) \models \sim p_A$.

If $(M, r, n_0) \models \varphi_3$, then for all $k > 0$ there will be an infinite level k alternating sequence of runs starting with (r, n_0). Moreover, if $(M, r, n_0) \models \varphi_2$ and M is in $\mathscr{C}_{(\mathrm{nf,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$, or $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, then the half of the grid property that holds with no forgetting guarantees that if $r_0, r_1, \ldots$ is a level $k+1$ alternating sequence of runs starting with (r, n_0), then it is also a level k alternating sequence of runs. (This is true since no forgetting implies that for all j, if $(r_j, k+1) \sim_i (r_{j+1}, k+1)$ then $(r_j, k) \sim_i (r_{j+1}, k)$.) Similarly, if M is in $\mathscr{C}_{(\mathrm{nl,sync})}$ or $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, then if $r_0, r_1, \ldots$ is a level k alternating sequence of runs starting with (r, n_0), then it is also a level $k+1$ alternating sequence of runs.

We intend to encode ID's (instantaneous descriptions) of the Turing machine **A** along these alternating sequences of runs. The kth ID of some computation of **A** will be encoded at the points of a level k alternating sequence of runs that satisfy p_A. More formally, what we are aiming for is to find a formula ψ such that if $(M, r, n_0) \models \psi$ then the following property holds:

> There is a computation **comp** of **A** started on the empty string such that for all $k \geq 0$, there exists a level k alternating sequence of runs $r_0, r_1, \ldots$ starting with (r, n_0), such that for all $c \in \mathrm{CD}$, we have $(M, r_{2j}, n_0 + k) \models p_c$ iff c is in the jth cell after the kth step of **comp**. (*)

The situation that we are trying to capture in (*) is shown in Fig. 4, where c_{jk} denotes the contents of the jth cell after the kth step of **comp**. The lack of the full grid property will make it somewhat more difficult to find such a formula ψ, but we now show it can be done.

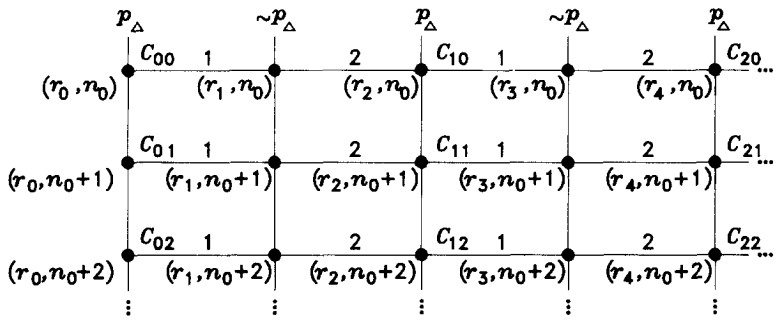

FIG. 4. Encoding a computation by (*).

Suppose s_0 is the initial state of **A**. Let φ_4 be the formula

$$p_\# \wedge p_\Delta \wedge K_1(\sim p_\Delta \Rightarrow K_2(p_\Delta \Rightarrow p_{\langle b, s_0 \rangle} \wedge K_1(\sim p_\Delta \Rightarrow K_2(p_\Delta \Rightarrow p_b))))$$
$$\wedge C(p_\Delta \wedge p_b \Rightarrow K_1(\sim p_\Delta \Rightarrow K_2(p_\Delta \Rightarrow p_b))).$$

φ_4 guarantees that the computation "starts right," with a blank tape and **A** in state s_0. (Note that the last conjunct forces blanks everywhere past the second cell.)

We now have to make sure that consecutive ID's in the computation match up right. It is well known that we can characterize a Turing machine by giving a function which, given three consecutive cells in an ID, describes the set of possible corresponding three cells in the next ID. Thus, given the Turing machine **A** and $i, j, k \in \mathrm{CD}$ with there is a function N such that

$$N(i, j, k) = \{\langle c, d, e \rangle \mid \text{if } i, j, k \text{ describes three consecutive cells in a given ID then } \langle c, d, e \rangle \text{ is a possible description of the corresponding cells in the next ID}\}.$$

Let φ_5 be the formula

$$\Box C\Bigg(\bigwedge_{i,j,k \in \mathrm{CD}} \Big((p_\Delta \wedge p_i \wedge K_1(\sim p_\Delta \Rightarrow K_2(p_\Delta \Rightarrow p_j \wedge K_1(\sim p_\Delta \Rightarrow K_2(p_\Delta \Rightarrow p_k)))))$$
$$\Rightarrow \bigvee_{\langle c,d,e \rangle \in N(i,j,k)} \bigcirc(p_c \wedge K_1(\sim p_\Delta \Rightarrow K_2(p_\Delta \Rightarrow p_d \wedge K_1(\sim p_\Delta \Rightarrow K_2(p_\Delta \Rightarrow p_e)))))\Big)\Bigg).$$

Let ψ be the conjunction of φ_1 through φ_5.

LEMMA 3.3. *If* $M \in \mathscr{C}_{(\mathrm{nf,sync})}$ *(resp.* $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$, $\mathscr{C}_{(\mathrm{nl,sync})}$*) and* $(M, r, n_0) \models \psi$, *then* (*) *holds.*

Proof. We first consider the case where we have no learning, so that M is in $\mathscr{C}_{(\mathrm{nl,sync})}$ or $\mathscr{C}_{(\mathrm{nf,nl,sync})}$. By φ_3, there is some level 0 alternating sequence of runs

starting with (r, n_0), say $r_0, r_1, \ldots$. As observed above, by φ_2 together with the assumption of no learning, $r_0, r_1, \ldots$ is actually a level k alternating sequence of runs starting with (r, n_0) for all k. By φ_1 and φ_4, at level 0 this sequence encodes the initial ID, where the tape is blank and **A** starts in state s_0. Using φ_1 and φ_5, it is easy to show by induction on k that there is a computation **comp** of **A** such that at level k, this sequence encodes the kth step of **comp**.

If M is in $\mathscr{C}_{(\mathrm{nf,sync})}$ or $\mathscr{C}_{(\mathrm{nf,sync,uis})}$ we have to work a little harder. In fact, we first prove the following version of (*):

> For all level k alternating sequences of runs $r_0, r_1, \ldots$ starting with (r, n_0), there exists a computation **comp** of **A** started on the empty string such that for all $l \leqslant k$ and all $c \in \mathrm{CD}$, we have $(M, r_{2j}, n_0 + l) \models p_c$ iff c is in the jth cell after the lth step of **comp**. (**)

(**) is proved by induction on k. The case $k = 0$ follows immediately from φ_1 and φ_4. For the general case, suppose $r_0, r_1, \ldots$ is a level $k+1$ alternating sequence of runs starting with (r, n_0). From the assumption of no forgetting, it follows using φ_2 that $r_0, r_1, \ldots$ is also a level k alternating sequence of runs. From the induction hypothesis, it follows that this latter sequence of points encodes the kth step of some computation of **A**. Now from φ_1 and φ_5, it follows that $(r_0, n_0 + k + 1)$, $(r_1, n_0 + k + 1), \ldots$ does indeed encode the $(k+1)$th step of some computation of **A**. This completes the proof of (**).

By φ_3 it follows that for all k, there is some level k alternating sequence starting with (r, n_0). We can now construct a tree whose nodes at depth k consist of all kth steps of computations encoded by level k alternating sequences in M starting with (r, n_0). We put an edge between a depth k node and a depth $k+1$ node exactly if there is a computation **comp** such that these nodes encode the kth and $(k+1)$th steps of **comp**. It is easy to see that the tree so constructed is finitely branching and, by (**) and the fact that we have level k alternating sequences for all k, it has arbitrarily long paths. By König's lemma, there must be an infinitely long path in the tree. (*) now follows. ▮

We are almost done. We just need one more formula to say that **A** is recurrent; i.e., that there is some computation **comp** where **A** reenters the start state s_0 infinitely often. Note that it would be consistent with ψ that several computations of **A** were being simultaneously encoded by different level k alternating sequences of runs. Since we want to make sure that there is one particular computation **comp** where **A** does reenter the start state infinitely often, we require that infinitely often *all* the computations being encoded by M are in the start state. This is the job of φ_6:

$$\Box \Diamond \left(\sim C \sim \left(\bigvee_{c \in \Gamma \times \{s_0\}} p_c \right) \wedge C \sim \left(\bigvee_{c \in \Gamma \times \{s_i \mid i \neq 0\}} p_c \right) \right).$$

(Recall that $\Diamond$ is the dual of $\Box$, so that $\Diamond \psi$ is an abbreviation for $\sim \Box \sim \psi$.) Let $\varphi_{\mathbf{A}}$ be the conjunction of φ_1 through φ_6.

LEMMA 3.4. *The formula $\varphi_{\mathbf{A}}$ is satisfiable in an interpreted system $M \in \mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$, $\mathscr{C}_{(\mathrm{nl,sync})}$) iff* **A**, *when started on a blank tape, admits an infinite computation which reenters its start state infinitely often.*

Proof. Suppose $(M, r, n) \models \varphi_{\mathbf{A}}$ for some interpreted system $M \in \mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$ $\mathscr{C}_{(\mathrm{nf,sync,uis})}$, $\mathscr{C}_{(\mathrm{nl,sync})}$). By Lemma 3.3, it follows that (*) holds. Thus, M encodes a computation of **A** in the sense of (*). Moreover, φ_6 guarantees that in this infinite computation, **A** enters the start state infinitely often.

Conversely, suppose that **A** is recurrent. We first construct an interpreted system $M_A = (R_A, \pi_A) \in \mathscr{C}_{(\mathrm{nf,nl,sync})}$ for two processors and a run r_0 of M such that $(M, r_0, 1) \models \varphi_{\mathbf{A}}$. Since $\mathscr{C}_{(\mathrm{nf,nl,sync})} = \mathscr{C}_{(\mathrm{nf,sync})} \cap \mathscr{C}_{(\mathrm{nl,sync})}$, this also gives us the desired result for $\mathscr{C}_{(\mathrm{nf,sync})}$ and $\mathscr{C}_{(\mathrm{nl,sync})}$.

We take $R_{\mathbf{A}}$ to consist of the runs $r_0, r_1, r_2, \ldots$. Let the processor's local states be of the form (j, n) (j and n integers), and define $r_j(n) = \langle (\lfloor j/2 \rfloor, n), (\lceil j/2 \rceil, n) \rangle$. Thus, we have $(r_{2j}, n) \sim_1 (r_{2j+1}, n)$ and $(r_{2j+1}, n) \sim_2 (r_{2j+2}, n)$. Note that this definition guarantees that the processors do not forget and do not learn. Given this definition, we can view the system as a two-dimensional grid, with edges labelled 1 alternating with edges labelled 2. Let **comp** be a computation where **A** reenters the start state infinitely often. We define $\pi_{\mathbf{A}}$ so that this computation is encoded by the interpreted system $M_{\mathbf{A}}$ in the sense of (*). In particular, we define $\pi_{\mathbf{A}}$ so that:

1. For $c \in \mathrm{CD}$, we have that p_c is true at the point $(r_{2j}, n+1)$ iff c is in the jth cell after the nth step of the computation of **comp**,

2. p_{Δ} is true at points of the form (r_{2j}, n) and false at points of the form (r_{2j+1}, n). It is now easy to check that $(M_{\mathbf{A}}, r_0, 1) \models \varphi_{\mathbf{A}}$.

In order to deal with $\mathscr{C}_{(\mathrm{nf,sync,uis})}$, we slightly modify the system $R_{\mathbf{A}}$ so that there is a unique initial state. For all $r_j \in R_{\mathbf{A}}$, we take $r_j(0) = \langle 0, 0 \rangle$. This guarantees that there is a unique initial state, and we still have that $(M_{\mathbf{A}}, r_0, 1) \models \varphi_{\mathbf{A}}$. (Note that by adding this unique initial state, we have lost the property of no learning, so that this trick would not work for $\mathscr{C}_{(\mathrm{nf,nl,sync,uis})}$.) ∎

The $\prod_1^1$ lower bound on validity for the language $CKL_{(m)}$, $m \geq 2$, now follows. We briefly sketch the modifications required to deal with $CKB_{(m)}$. We first replace all occurrences of $\bigcirc$ (resp. $\Box$) in φ_i, $i = 1, \ldots, 5$, by $\forall\bigcirc$ (resp. $\forall\Box$). Call the resulting formulas φ_i^b, and let ψ_b be the conjunction of these formulas. We leave it to the reader to check that the analog of Lemma 3.3 holds for ψ_b. Let φ_6^b be $\forall\Box\forall\Diamond(\sim C \sim (\bigvee_{c \in \Gamma \times \{s_0\}} p_c) \wedge C \sim (\bigvee_{c \in \Gamma \times \{s_i \mid i \neq 0\}} p_c))$, and let $\varphi_{\mathbf{A}}^b$ be the conjunction of φ_1^b through φ_6^b. Then the analog of Lemma 3.4 holds for $\varphi_{\mathbf{A}}^b$. (However, note that we need $\forall\Diamond$ in φ_6^b rather than $\exists\Diamond$ in order to guarantee recurrence if we allow learning.) ∎

Remark. It may seem that in order to be able to encode the computation of arbitrary Turing machines in the language, we need an infinite number of primitive propositions (since the number of primitive propositions used to encode the computation of **A** in the proof above is greater than the number of states in **A**).

However, it is easy to see that we can encode the computation of arbitrary Turing machines using only two primitive propositions and a slightly more sophisticated encoding. There are also some situations where we may want to restrict the interpreted system so that the truth value of primitive propositions is stable along all runs; i.e., for all primitive propositions p, the formula $\Box p \vee \Box \sim p$ is valid. This is one of the assumptions considered, for example, in [FHV2]. Ron Fagin has pointed out that even with this assumption, our $\prod_1^1$ lower bound still holds, although we seem to require at least three agents in this case. The idea is to replace each use of a primitive proposition p in our encoding by the formula $K_1 \sim K_3 p$. We omit details here. ∎

For the classes $\mathscr{C}_{(\mathrm{nf})}$, $\mathscr{C}_{(\mathrm{nf,nl})}$, and $\mathscr{C}_{(\mathrm{nf,uis})}$, where we no longer assume time is synchronous but do still assume that we have no forgetting, it is easy to modify the previous proof to again get a $\prod_1^1$ lower bound. In fact, we can do this in a uniform way; we construct a formula that essentially *forces* synchrony. More formally, we have

PROPOSITION 3.5. *For all formulas φ in $CKL_{(m)}$ (resp. $CKB_{(m)}$), there is a formula $sync_\varphi$ such that φ is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$) if and only if $\varphi \wedge sync_\varphi$ is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl})}$, $\mathscr{C}_{(\mathrm{nf,uis})}$).*

Proof. Let φ be a formula in $CKL_{(m)}$. Let *tick* be a new primitive proposition (not appearing in φ), and let $sync_\varphi$ be the formula:

$$C\,\Box\,((tick \Rightarrow C\,tick) \wedge (\sim tick \Rightarrow C \sim tick))$$
$$\wedge\, C\,\Box\,((tick \Rightarrow \bigcirc \sim tick) \wedge (\sim tick \Rightarrow \bigcirc tick)).$$

Thus, $sync_\varphi$ says that the truth value of *tick* is always common knowledge, and that its truth value changes at consecutive points along any run (we can think of a change in the value of *tick* as denoting one tick of a global clock). Note that the only dependence of $sync_\varphi$ on φ is in the choice of the primitive proposition *tick*.

Now suppose that φ is satisfied in some interpreted system M in $\mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$). Since *tick* does not appear in φ, we can assume without loss of generality that *tick* is true at all points in M of the form $(r, 2n)$ and false at all other points. Clearly $\varphi \wedge \mathrm{sync}_\varphi$ is then satisfied in M. This gives us one direction of the result.

For the converse, suppose $M = (R, \pi)$ is in $\mathscr{C}_{(\mathrm{nf})}$ and $(M, r_0, n_0) \models \varphi \wedge sync_\varphi$ for some point (r_0, n_0) in M. We now show that $sync_\varphi$ essentially forces the system to be weakly synchronous. For suppose not. Then there must be two points (r, n) and (r, n') with $n' > n$ such that $(r, n) \sim_i (r, n')$, for some agent i. Because we are assuming no forgetting, by Lemma 2.2 it follows that $(r, n) \sim_i (r, n'')$ for all n'' with $n \leqslant n'' \leqslant n'$. (Note the key use of no forgetting here.) Since *tick* changes its truth value at every step, in particular, it must be the case that *tick* has different truth values at (r, n) and $(r, n+1)$. But we have just shown that (r, n) and $(r, n+1)$ are

indistinguishable, which contradicts the assumption that the truth value of *tick* is common knowledge. Thus, the system (or at least that part of it below points reachable from (r_0, n_0)) is weakly synchronous. It is also not hard to show that in the weakly synchronous part of M, if $(r, n) \sim_i (r', n')$ and $n > 0$, then $(r, n-1) \sim_i (r', n'-1)$. Thus, we have the situation shown in Fig. 5. We can now convert the weakly synchronous system to a synchronous system by chopping off the prefixes of runs.

Formally, we proceed as follows. Let R' consist of all runs reachable from (r_0, n_0) and, for all $r \in R'$, let (r, n_r) be the first point in r reachable from (r_0, n_0). Let f be a function on the runs in R' defined by $f(r)(n) = r(n_r + n)$. Thus, $f(r)$ is the result of chopping off the prefix of r before n_r and relabelling the points so that we start with 0. Let $M' = (f(R'), \pi')$, where $\pi'(f(r), n) = \pi(r, n_r + n)$ for $r \in R'$. An easy induction now shows that for $r \in R'$ and all formulas ψ, we have $(M', f(r), n) \models \psi$ iff $(M, r, n + n_r) \models \psi$. We next show that for $r, r' \in R'$ and $k \geq 0$, if $(r, n_r + k) \sim_i (r', n'_r + k')$, then $k = k'$. We prove the result by induction on k. It suffices to show that $k \leq k'$; equality follows by the symmetry of $\sim_i$. For the case $k = 0$ the result is immediate by the definition of $n_{r'}$. Suppose $k > 0$. By the assumption of no forgetting, it follows that $r(n_r + k - 1) \sim_i r'(n_{r'} + k'')$ for some $k'' \leq k'$. By the induction hypothesis, it follows that $k - 1 \leq k''$. Now we must only show that in fact $k'' < k'$ and we will get $k \leq k'$, as desired. Note that if $k'' = k'$ (the only other possibility), then by the transitivity of $\sim_i$ we get $r(n_r + k - 1) \sim_i r(n_r + k)$. Observe that since the formula $sync_\varphi$ is prefixed by $C\Box$, it follows from the construction that $(M, r, n_r + k - 1) \models sync_\varphi$. Suppose, without loss of generality, that $(M, r, n_r + k - 1) \models tick$. Since $sync_\varphi$ forces the truth value of *tick* to alternate along a run, it follows that $(M, r, n_r + k - 1) \models tick$. But $sync_\varphi$ also forces the truth value of *tick* to be common knowledge, which contradicts the observation that $r(n_r + k - 1) \sim_i r(n_r + k)$. This proves the desired result.

It now easily follows that $(f(r), k) \sim_i (f(r'), k')$ implies $k = k'$. Moreover, we get that processors do not forget in M', since if $(f(r), n) \sim_i (f(r'), n)$ and $k \leq n$, then by construction we have $(r, n_r + n) \sim_i (r', n_{r'} + n)$. Since processors do not forget in M, we must have some k' such that $(r, n_r + k) \sim_i (r', n_{r'} + k')$. By the previous observation we have $k = k'$, so that $(f(r), k) \sim_i (f(r'), k)$. Thus $M \in \mathscr{C}_{(\text{nf, sync})}$.

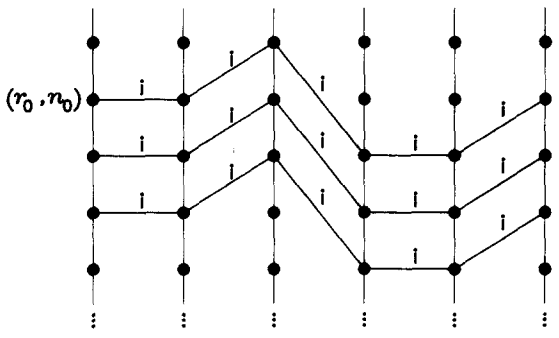

FIG. 5. An interpreted system M where $(M, r_0, n_0) \models sync_\varphi$.

A similar proof works in the case of $M \in \mathscr{C}_{(\mathrm{nf,nl})}$. For the case $M \in \mathscr{C}_{(\mathrm{nf,uis})}$, we proceed as above, and then add an initial state as in the proof of Lemma 3.4. Finally, if φ is in the language $CKB_{(m)}$, we construct $sync_{\varphi}$ using $\forall\Box$ and $\forall\bigcirc$ rather than $\Box$ and $\bigcirc$. ▮

COROLLARY 3.6. *The validity problem for* $CKL_{(m)}$ *and* $CKB_{(m)}$, $m \geq 2$, *is* $\prod_1^1$*-hard with respect to the following classes of interpreted systems:* $\mathscr{C}_{(\mathrm{nf})}$, $\mathscr{C}_{(\mathrm{nf,nl})}$, *and* $\mathscr{C}_{(\mathrm{nf,uis})}$.

Proof. Consider the formula $\varphi_{\mathrm{A}} \wedge \mathrm{sync}_{\varphi_{\mathrm{A}}}$. By Proposition 3.5, this formula is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl})}$, $\mathscr{C}_{(\mathrm{nf,uis})}$) iff φ_{A} is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$), and by the proof of Theorem 3.2, this holds iff A is recurrent. This observation gives us the lower bound. ▮

In the formula $\varphi_{\mathrm{A}} \wedge sync_{\varphi_{\mathrm{A}}}$ above, we make heavy use of the nexttime operator ($\bigcirc$). As Lamport has argued [La1], it is somewhat unreasonable to have a nexttime operator in the language if we are considering asynchronous systems. In fact, this use of the nexttime operator is unnecessary. The reader is referred to the Appendix for details.

In the previous proof where we eliminated the assumption of synchrony, we made heavy use of our assumption of no forgetting. The proof does not go through if we only assume no learning. Nevertheless, as we now show, we can still make use of these ideas in the presence of no learning.

Given an interpreted system M, we say a point (r, n) in M is *i-repeating* if there exist infinitely many $n' > n$ such that $(r, n) \sim_i (r, n')$.

LEMMA 3.7. *Let* $M = (R, \pi)$ *be an interpreted system in* $\mathscr{C}_{(\mathrm{nl})}$ *(resp.* $\mathscr{C}_{(\mathrm{nl,uis})}$*) and let* $r, r' \in R$.

1. *If* $n_0 < n_1 < n_2$, $(r, n_0) \sim_i (r, n_2)$, *and it is not the case that* $(r, n_1) \sim_i (r, n_2)$, *then both* (r, n_0) *and* (r, n_1) *are i-repeating.*

2. *If* $(r, n) \sim_i (r, n')$, $n \leq n'$, *and* (r, n') *is not i-repeating, then* $(r, n) \sim_i (r, n'')$ *for all* n'' *with* $n \leq n'' \leq n'$.

3. *If* (r, n) *is i-repeating and* $n' > n$, *then* (r, n') *is i-repeating.*

4. *If* $(r, n) \sim_i (r', n')$, *then* (r, n) *is i-repeating iff* (r', n') *is i-repeating.*

Proof. For part 1, we extend n_0, n_1, n_2 to a sequence $\langle n_j \rangle$, $j \geq 0$, such that $n_j < n_{j+1}$, $(r, n_{2k}) \sim_i (r, n_0)$, and $(r, n_{2k+1}) \sim_i (r, n_1)$. The existence of such a sequence is almost immediate from the assumption of no learning. For example, since $(r, n_0) \sim_i (r, n_2)$ and $n_1 > n_0$ by assumption, there must exist $n_3 \geq n_2$ such that $(r, n_1) \sim_i (r, n_3)$. We cannot have $n_3 = n_2$ since it is not the case that $(r, n_2) \sim_i (r, n_1)$. The rest of the construction continues in the same way. The existence of such a sequence shows that (r, n_0) and (r, n_1) are both i-repeating.

For part 2, note that if $n < n'' < n'$ and it is not the case that $(r, n) \sim_i (r, n'')$, then by part 1, (r, n') is i-repeating, contradicting our original assumption.

For part 3, observe the result is immediate if $(r, n) \sim_i (r, n')$. If not, since (r, n) is i-repeating, there must exist $n'' > n'$ such that $(r, n) \sim_i (r, n'')$. The result now follows from part 1.

For part 4, suppose that (r, n) is i-repeating. We now show that (r', n') is i-repeating. Suppose not. Then there is some $k' > n'$ such that for all $n'' \geq k'$, it is not the case that $(r', n') \sim_i (r', n'')$. By the assumption of no learning, there is some $k \geq n$ such that $(r, k) \sim_i (r', k')$. Since (r, n) is i-repeating, it must be the case that for some $l > k$, we have $(r, l) \sim_i (r, n)$. By the assumption of no learning again, we must have some $l' \geq k'$ such that $(r, l) \sim_i (r', l')$. By the transitivity of $\sim_i$, it follows that $(r', l') \sim_i (r', n')$. But this contradicts our choice of k. Thus, (r', n') is i-repeating. Part 4 now follows by the symmetry of $\sim_i$. ∎

Note that, among other things, this lemma tells us that in the non-i-repeating part of a run, we essentially have the property described in Lemma 2.2. This was the main property we needed to force synchrony in Proposition 3.5. We cannot quite force synchrony in systems with no learning, but we can come close enough to get the $\prod_1^1$ lower bound, as the following result shows.

THEOREM 3.8. *The validity problem for $CKL_{(m)}$ and $CKB_{(m)}$, $m \geq 2$, is $\prod_1^1$-hard with respect to $\mathscr{C}_{(\mathrm{nl})}$.*

Proof. We slightly modify the construction of $\varphi_{\mathbf{A}}$ given in Theorem 3.2 to again get a formula $\varphi_{\mathbf{A}}^{\mathrm{nl}}$ that encodes the computation of the TM **A** in the nonrepeating part of runs. We proceed as follows.

Let q be a new primitive proposition (not appearing in $\varphi_{\mathbf{A}}$) and let *nonrep* be an abbreviation $q \wedge \Diamond \Box \sim q$. It is easy to see that $K_i(nonrep)$ is true at the point (r, n) only if (r, n) is not i-repeating.

Essentialy the same argument used in Proposition 3.5 shows that the formula $sync_{\varphi_{\mathbf{A}}}$ constructed in that proof also forces weak synchrony in the nonrepeating part of any interpreted system in $\mathscr{C}_{(\mathrm{nl})}$. Formally, if $M \in \mathscr{C}_{(\mathrm{nl})}$, $(r, n) \sim_i (r, n')$, (r, n') is not an i-repeating point, and $(M, r, n) \models sync_{\varphi_{\mathbf{A}}}$, then $n = n'$. To see this, suppose by way of contradiction that $n' > n$. Note that by part 2 of Lemma 3.7, we must have $(r, n) \sim_i (r, n+1)$. Since $sync_{\varphi_{\mathbf{A}}}$ forces the truth value of *tick* at (r, n) to be common knowledge and to change at every step, this leads us to a contradiction, just as in the proof of Proposition 3.5. The situation is now essentially that described in Fig. 5.

At this point there are still two problems to be dealt with before we can run through the proof of Theorem 3.2. The first is that we cannot now delete the prefix of each run as we did in the proof of Proposition 3.5 in order to get a synchronous system. The problem is that we may well have $(r, n) \sim_1 (r', n')$ and $(r, n) \sim_2 (r', n'')$, with $n' \neq n''$. (As the proof of Proposition 3.5 shows, this can not happen if we make the assumption of no forgetting; it can happen with no learning.) It turns out that weak synchrony is enough for our purposes; we just need to appropriately modify the statement (*) from the proof of Theorem 3.2.

A more serious problem is that we cannot use the formula *nonrep* to force all the points in a run to be non-i-repeating. The reason is that $\Box K_i(nonrep)$ is unsatisfiable (in fact, $\Box(nonrep)$ is unsatisfiable). We deal with this problem by offsetting the computation by one run at each step. More precisely, we construct $\varphi_{\mathbf{A}}^{\mathrm{nl}}$ such that if $M \in \mathscr{C}_{(\mathrm{nl})}$ and $(M, r_0, n_0) \models \varphi_{\mathbf{A}}^{\mathrm{nl}}$, then the following variant of property (*) holds. (The reader should compare the conditions below to those defining a level k alternating sequence of runs.)

There is a computation **comp** of **A** started on the empty string and a sequence

(r_0, n_0), (r_1, n_1), (r_2, n_2), ... of points such that for all $k \geqslant 0$ we have $(r_{2j}, n_{2j}+k) \sim_1 (r_{2j+1}, n_{2j+1}+k)$, $(r_{2j+1}, n_{2j+1}+k) \sim_2 (r_{2j+2}, n_{2j+2}+k)$, $(M, r_{2j}, n_{2j}+k) \models p_{\Delta}$, $(M, r_{2j+1}, n_{2j+1}+k) \models \sim p_{\Delta}$, and for all $c \in \mathrm{CD}$, we have $(M, r_{2(j+k)}, n_{2(j+k)}+k) \models p_c$ iff c is in the jth cell after the kth step of **comp**. (+)

The situation that we are trying to capture in (+) is shown in Fig. 6, where c_{jk} denotes the contents of the jth cell after the kth step of **comp**.

We construct $\varphi_{\mathbf{A}}^{\mathrm{nl}}$ as follows. Let φ_0 be the formula:

$$\Box C(sync_{\varphi_{\mathbf{A}}}) \wedge \Box C((p_{\#} \wedge p_{\Delta} \Rightarrow E(nonrep)) \wedge (p_{\Delta} \wedge E(nonrep) \Rightarrow K_1(\sim p_{\Delta} \Rightarrow K_2(p_{\Delta} \Rightarrow E(nonrep))))).$$

As our comments above indicate, the first conjunct of φ_0 forces weak synchrony in the nonrepeating part. The second conjunct of φ_0 says that along any sequence encoding an ID, all relevant points are nonrepeating.

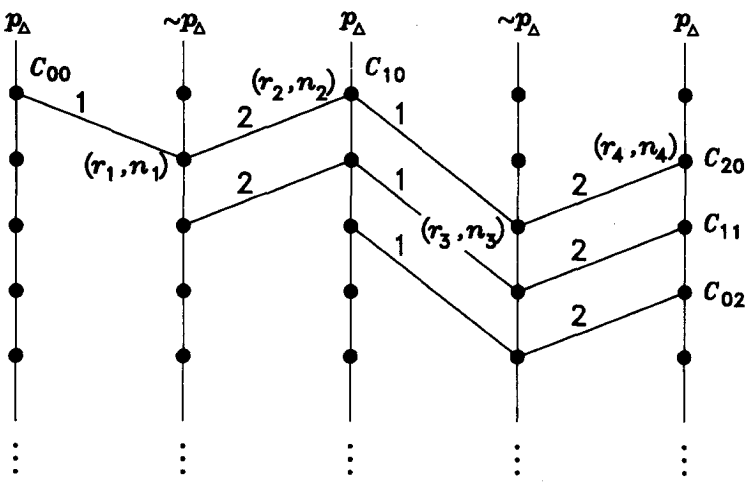

FIG. 6. Encoding a computation by (+).

Let φ'_5 be the variant of φ_5 that offsets the computation by one step:

$$\Box C \bigwedge_{i,j,k \in CD} (p_A \wedge p_i \wedge K_1(\sim p_A \Rightarrow K_2(p_A \Rightarrow p_j \wedge K_1(\sim p_A \Rightarrow K_2(p_A \Rightarrow p_k))))$$

$$\Rightarrow \bigvee_{\langle c,d,e \rangle \in N(i,j,k)} \bigcirc K_1(\sim p_A \Rightarrow K_2(p_A \Rightarrow p_c \wedge K_1(\sim p_A$$

$$\Rightarrow K_2(p_A \Rightarrow p_d \wedge K_1(\sim p_A \Rightarrow K_2(p_A \Rightarrow p_e)))))))).$$

Let φ_i, $i = 1, 2, 3, 4, 6$ be as in the proof of Theorem 3.2, and let $\varphi_{\mathbf{A}}^{\mathrm{nl}}$ be the conjunction of φ_0, φ'_5, and φ_i, $i = 1, 2, 3, 4, 6$. Now a similar argument to that used in the proof of Theorem 3.2 shows that $\varphi_{\mathbf{A}}^{\mathrm{nl}}$ is satisfiable with respect to $\mathscr{C}_{(\mathrm{nl})}$ iff **A** is recurrent. The result now follows. The standard modifications now enable us to deal with $CKB_{(m)}$ as well. ∎

As we remarked in the Introduction, the combination of no learning and unique initial state leads us to some anomalous situations with regard to the complexity of the validity problem. As we pointed out in the proof of Lemma 3.4, the straightforward trick used to add a unique initial state to a system does not work if we require no learning (and are dealing with more than one processor). Indeed, if we consider the classes $\mathscr{C}_{(\mathrm{nf,nl,sync,uis})}$ and $\mathscr{C}_{(\mathrm{nl,sync,uis})}$, it is easy to show that we get no more expressive power with many agents and common knowledge than we do with just one agent. More formally, it is easy to show

PROPOSITION 3.9. 1. $\mathscr{C}_{(\mathrm{nf,nl,sync,uis})} = \mathscr{C}_{(\mathrm{nl,sync,uis})}$.

2. *Any formula φ in $CKL_{(m)}$ (resp. $CKB_{(m)}$) is equivalent in interpreted systems in $\mathscr{C}_{(\mathrm{nl,sync,uis})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync,uis})}$) to the formula φ' that results by replacing all occurrences of K_i, $i \geqslant 2$, E, and C by K_1.*

Proof. Clearly we have $\mathscr{C}_{(\mathrm{nf,nl,sync,uis})} \subseteq \mathscr{C}_{(\mathrm{nl,sync,uis})}$. For the opposite inclusion, consider an interpreted system $M = (R, \pi) \in \mathscr{C}_{(\mathrm{nl,sync,uis})}$. The assumption of unique initial state guarantees that for all r, $r' \in R$ and all processors i, we have that $(r, 0) \sim_i (r', 0)$. Since the system is synchronous and there is no learning, it is easy to show by induction on n that we have $(r, n) \sim_i (r', n)$ for all n. It immediately follows that processors do not forget. Moreover, an easy induction on the structure of φ proves the second part. ∎

Thus, the lower bound for the validity problem with respect to $\mathscr{C}_{(\mathrm{nl,sync,uis})}$ and $\mathscr{C}_{(\mathrm{nf,nl,sync,uis})}$ for the language $CKL_{(m)}$ (resp. $CKB_{(m)}$) is the same as the lower bound for $KL_{(1)}$ (resp. $KB_{(1)}$), namely EXPSPACE. (We prove this lower bound in the next section; the matching upper bound is proved in the sequel to this paper.)

On the other hand, if we drop the assumption of synchronous time, the situation changes drastically. Ladner and Reif [LR] prove that for interpreted systems in $\mathscr{C}_{(\mathrm{nf,nl,uis})}$, even the language $KB_{(2)}$ is undecidable. In particular, they show that the computation of a nondeterministic Turing machine can be encoded by a $KB_{(2)}$ for-

mula. They do their encoding by laying out the computation of the TM *vertically* down one run, rather than using many runs as we do in the proof of Theorem 3.2. (We remark that we also encode computations vertically in the proofs of our non-elementary bounds in the next section.) It is straightforward to add a conjunct to their formula so that we actually encode the recurrence problem. Other trivial modifications allow us to do the encoding by a $KL_{(2)}$ formula. We omit the details here. As a consequence we get:

THEOREM 3.10. *The validity problem for* $KL_{(m)}$ *and* $KB_{(m)}$, $m \geqslant 2$, *with respect to* $\mathscr{C}_{(\mathrm{nf,nl,uis})}$ *is* $\prod_1^1$*-hard.*

COROLLAR 3.11. *The validity problem for* $CKL_{(m)}$ *and* $CKB_{(m)}$, $m \geqslant 2$, *with respect to* $\mathscr{C}_{(\mathrm{nf,nl,uis})}$ *is* $\prod_1^1$*-hard.*

By combining the ideas of Theorem 3.8 and those in Ladner and Reif's proof, we can also prove a tight bound for validity with respect to $\mathscr{C}_{(\mathrm{nl,uis})}$. At first glance, it might seem that we could just force the run along which the computation of the TM is encoded to be nonrepeating, and then proceed as before to encode the non-recurrence problem and get a $\prod_1^1$ lower bound. However, as we pointed out in the proof of Theorem 3.8, if we consider the formula $K_i(nonrep)$ that we used to force a point to be non-*i*-repeating, $\Box K_i(nonrep)$ is unsatisfiable. Thus, we cannot use this formula to force a whole run to be non-*i*-repeating, but only a finite prefix of that run. This problem is not just a consequence of the way we defined the formula non-rep. As we show in the sequel to this paper, a formula is satisfiable with respect to $\mathscr{C}_{(\mathrm{nl})}$ (resp. $\mathscr{C}_{(\mathrm{nl,uis})}$) iff it is satisfiable with respect to an interpreted system in $\mathscr{C}_{(\mathrm{nl})}$ (resp. $\mathscr{C}_{(\mathrm{nl,uis})}$) in which only a finite prefix of every run is non-*i*-repeating, for each agent *i*. The fact that only a finite prefix of a run is non-*i*-repeating suffices to allow us to encode the halting problem in a formula, using the techniques of Ladner and Reif. As a consequence we get:

THEOREM 3.12. *The validity problem for* $KL_{(m)}$ *and* $KB_{(m)}$, $m \geqslant 2$, *with respect to* $\mathscr{C}_{(nl,uis)}$ *is co-r.e.-hard.*

COROLLAY 3.13. *The validity problem for* $CKL_{(m)}$ *and* $CKB_{(m)}$, $m \geqslant 2$, *with respect to* $\mathscr{C}_{(\mathrm{nl,uis})}$ *is co-r.e.-hard.*

We remark that the observations made above also allow us to prove that these bounds are tight, and we do so in the sequel to this paper.

Finally, we consider the situation for classes of interpreted systems where we do not assume either no forgetting or no learning. Here the validity problem becomes much easier.

THEOREM 3.14. *The validity problem for* $CKL_{(m)}$ *and* $CKB_{(m)}$, $m \geqslant 2$, *with respect to* $\mathscr{C}$, $\mathscr{C}_{(\mathrm{sync})}$, $\mathscr{C}_{(\mathrm{sync,uis})}$, *and* $\mathscr{C}_{(\mathrm{uis})}$ *is EXPTIME-hard.*

Proof. The result follows immediately from the fact that even without temporal operators, the logic with at least two agents and common knowledge is EXPTIME-hard [HM2]. ∎

We remark that Fischer and Immerman independently proved the EXPTIME result for $CKB_{(m)}$ with respect to $\mathscr{C}$ [FI2].

4. Lower Bounds for $KL_{(m)}$ and $KB_{(m)}$

In order to prove the complexity results discussed in the Introduction for classes of interpreted systems with no learning or no forgetting, we first show how we can use formulas to encode "yardsticks" of the type used by Stockmeyer. Again we start with synchronous systems. Our yardsticks are going to be of length $f(k, n)$, where $f(0, n) = n$, and $f(k+1, n) = f(k, n)\, 2^{f(k,n)}$. Note that $f(k, n) \geqslant \mathrm{ex}(k, n)$ for all $k \geqslant 0$ and $n \geqslant 1$. (Also note that there is some constant c such that $f(k, n) \leqslant \mathrm{ex}(k, cn)$ for all $k \geqslant 0$ and $n \geqslant 1$.) For each $k \geqslant 1$ and $n > 1$ we construct a formula $\varphi_{k,n}$ that forces a proposition p_k to act as a yardstick of length $f(k, n)$, in the sense that it is true exactly $f(k, n)$ steps apart. The situation is shown in Fig. 7. More precisely,

Lemma 4.1. *For all k and n, there is formula $\varphi_{k,n}$ of $KL_{(2)}$ with $|\varphi_{k,n}| = O(k+n)$ and $\mathrm{ad}(\varphi_{k,n}) = k-1$, such that for all interpreted systems $M \in \mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,nl,sync})}$), we have*

> if $(M, r, n_0) \models \varphi_{k,n}$, then there is some N with $0 \leqslant N < f(k, n)$ such that for $n \geqslant n_0$, $(M, r, n') \models p_k$ iff $n' = n_0 + N + jf(k, n)$ for some $j \geqslant 0$. (†)

Moreover, $\varphi_{k,n}$ is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,nl,sync})}$).

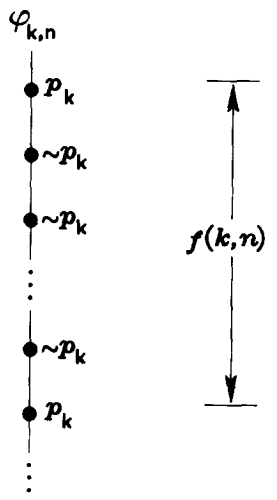

Fig. 7. Encoding yardsticks with $\varphi_{k,n}$ and p_k.

Proof. We construct $\varphi_{k,n}$ by induction on k. In the course of our construction we will also need formulas $\varphi'_{k,n}$ which are just like $\varphi_{k,n}$ except that all occurrences of K_1 are replaced by K_2 and vice versa. By symmetry, it will be easy to check that $\varphi'_{k,n}$ satisfies (†).

We now describe the construction of $\varphi_{1,n}$. The idea is to partition every run into segments of length n, and to view each such segment as an n-bit binary counter. We then force consecutive counters to differ by 1 modulo 2^n. Thus, the counter will take on consecutively all the values from 0 to $2^n - 1$. The formula p_1 will be true exactly when the counter is at 0, and thus will be true every $f(1, n) = n2^n$ steps. We proceed as follows.

We first define some abbreviations. Let $\bigcirc^1\psi$ and $\bigcirc^{\leq 1}\psi$ both the abbreviations for the formula $\bigcirc\psi$, let $\bigcirc^k\psi$ denote $\bigcirc\bigcirc^{k-1}\psi$ for $k > 1$, and let $\bigcirc^{\leq k}\psi$ denote $\bigcirc(\psi \wedge \bigcirc^{\leq k-1}\psi)$. It is easy to see that $\bigcirc^{\leq k}\psi$ is equivalent to $\bigcirc^1\psi \wedge \cdots \wedge \bigcirc^k\psi$; however, note that as we have defined it, the formula $\bigcirc^{\leq n}p$ has length $O(n)$ (since p is a primitive proposition), whereas the length of $\bigcirc p \wedge \cdots \wedge \bigcirc^n p$ is $O(n^2)$.

We use p_0 to mark the beginning of a counter. Let α_1 be the formula

$$\Box((p_0 \Leftrightarrow \bigcirc^n p_0) \wedge (p_0 \Rightarrow \bigcirc^{\leq n-1} \sim p_0)) \wedge \Diamond p_0.$$

Intuitively, α_1 says that the distance between the points where p_0 holds is precisely n, that p_0 is followed by $n-1$ occurrences of $\sim p_0$, and that p_0 holds at some point of the run.

We use the proposition b_0 to encode the bits of the counter, where b_0 encodes a 1, and $\sim b_0$ encodes a 0. Thus, $p_0 \wedge b_0$ followed by $n-1$ occurrences of $\sim b_0$ encodes the number 2^{n-1}. We now want to force consecutive counters to differ by 1 modulo 2^n. Recall that if $\mathbf{c} = c_{n-1} \cdots c_0$ and $\mathbf{d} = d_{n-1} \cdots d_0$ are two n-bits binary numbers, then $\mathbf{d}$ is the successor of $\mathbf{c}$ modulo 2^n precisely when the following holds: $c_i = d_i$ iff $c_j = 0$ for some $0 \leq j < i$; i.e., the ith bits in $\mathbf{c}$ and $\mathbf{d}$ are the same iff some bit c_j in $\mathbf{c}$ with $j < i$ is 0.

Let α_2 be the formula

$$\Box(\bigcirc(b_0 \, U \, p_0) \Rightarrow (b_0 \Rightarrow \bigcirc^n \sim b_0) \wedge (\sim b_0 \Rightarrow \bigcirc^n b_0))$$
$$\wedge \Box(\sim\bigcirc(b_0 \, U \, p_0) \Rightarrow (b_0 \Rightarrow \bigcirc^n b_0) \wedge (\sim b_0 \Rightarrow \bigcirc^n \sim b_0)).$$

Intuitively, the first conjunct of α_2 says that if, for some i, all the bits c_j, $j < i$, in a counter are 1, then the jth bit of the next counter is different. The second conjunct says that if some bit c_j, $j < i$, is 0, then ith bit of the next counter is the same.

Finally, we want to force p_1 to be true exactly if the counter is at 0. Let α_3 be the formula

$$\Box(p_1 \Leftrightarrow (p_0 \wedge \sim b_0 \wedge \bigcirc(\sim b_0 \, U \, p_0))).$$

It is easy to see if we now define $\varphi_{1,n}$ to be the conjunction of α_1, α_2, and α_3, then it has the required property (†).

The construction of $\varphi_{k+1,n}$ proceeds along very similar lines. We partition runs into segments of length $f(k, n)$, using p_k to mark the beginning of each segment. Again, we view each of these segments as an $f(k, n)$-bit binary counter, using the proposition b_{k+1} to encode the bits, and force consecutive counters to differ by 1 modulo $2^{f(k,n)}$. The formula p_{k+1} will be true exactly when the counter is at 0, and thus will be true every $f(k+1, n) = f(k, n)\, 2^{f(k,n)}$ steps. The only thing that prevents the definition of $\varphi_{k+1,n}$ from being identical to that of $\varphi_{1,n}$ is that we now cannot mark off segments of length $f(k, n)$ using $\bigcirc^{f(k,n)}$ in the same way we did in, for example, α_2, since the resulting formula would then be much too big. Our construction uses nested K_i's and the fact that we are in an interpreted system with no forgetting and/or no learning to measure this distance in a more succinct way. The idea is to distribute yardsticks across runs accessible by the $\sim_1$ relation in such a way that at every step, there is some accessible run whose counter is at 0. This is the job of the formula β_1, defined as

$$\Box(K_1(q_k \Rightarrow \varphi'_{k,n}) \wedge \sim K_1 \sim (p_k \wedge q_k) \wedge K_1(\Diamond q_k \Rightarrow q_k)).$$

Recall that $\varphi'_{k,n}$ is the result of reversing the roles of K_1 and K_2 in $\varphi_{k,n}$. We can think of the proposition q_k in the formula β_1 as denoting a run that encodes a counter. Thus, the first conjunct of β_1 says that those runs by accessible by the $\sim_1$ relation where q_k is true encode counters; the second conjunct says that there is always some run accessible by the $\sim_1$ relation where the counter is 0; and the third conjunct says that if q_k is true at any point in the future, then it is true at the present. By the induction hypothesis, the alternation depth of β_1 is k. As we shall see, all the other conjuncts that make up $\varphi_{k+1,n}$ will have alternation depth 0 or 1.

Let β_2 be the formula

$$\Box((b_{k+1} \Rightarrow K_1 b_{k+1}) \wedge (\sim b_{k+1} \Rightarrow K_1 \sim b_{k+1})).$$

Thus, β_2 forces b_{k+1} and $\sim b_{k+1}$ to be uniform across all accessible runs at any given level.

We are now in a position to give an analog to α_2. Let β_3 be the formula

$$\begin{aligned}
&\Box(\bigcirc(b_{k+1}\, U\, p_k) \Rightarrow (b_{k+1} \Rightarrow K_1(p_k \wedge q_k \Rightarrow \bigcirc(\sim p_k\, U\, (p_k \wedge \sim b_{k+1}))))\\
&\qquad\qquad \wedge (\sim b_{k+1} \Rightarrow K_1(p_k \wedge q_k \Rightarrow \bigcirc(\sim p_k\, U\, (p_k \wedge b_{k+1})))))\\
&\Box(\sim\bigcirc(b_{k+1}\, U\, p_k) \Rightarrow (b_{k+1} \Rightarrow K_1(p_k \wedge q_k \Rightarrow \bigcirc(\sim p_k\, U\, (p_k \wedge b_{k+1}))))\\
&\qquad\qquad \wedge (\sim b_{k+1} \Rightarrow K_1(p_k \wedge q_k \Rightarrow \bigcirc(\sim p_k\, U\, (p_k \wedge \sim b_{k+1}))))).
\end{aligned}$$

Intuitively, the formula $K_1(p_k \wedge q_k \Rightarrow \bigcirc(\sim p_k\, U\, (p_k \wedge \sim b_{k+1})))$ in the first conjunct of β_3 plays the same role as $\bigcirc^n \sim b_0$ in the first conjunct of α_2. To see this, suppose $M \in \mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,nl,sync})}$). Moreover, suppose $(M, r, n_0) \models \beta_1 \wedge \beta_2 \wedge \beta_3$ and, for some $n' \geq n_0$, we have $(M, r, n') \models b_{k+1} \wedge \bigcirc(b_{k+1}\, U\, p_k)$. We want to show that $(M, r, n' + f(k, n)) \models \sim b_{k+1}$. The argument now splits into two cases, depending on whether processors do not learn or do not forget.

If processors do not learn (i.e., M is in $\mathscr{C}_{(\mathrm{nl,sync})}$ or $\mathscr{C}_{(\mathrm{nf,nl,sync})}$), then by β_1, we know that there exists a run r' such that $(r, n') \sim_1 (r', n')$ and $(M, r', n') \models p_k \wedge q_k \wedge \varphi'_{k,n}$. Since processors do not learn, it must also be the case that $(r, n'+f(k,n)) \sim_1 (r', n'+f(k,n))$. By induction hypothesis, we know that $(M, r', n'+f(k,n)) \models p_k$ and that $(M, r', n'+i) \models \sim p_k$ for $0<i<f(k,n)$. By β_3, we have $(M, r', n'+f(k,n)) \models \sim b_{k+1}$. Finally, by β_2, we also have $(M, r, n'+f(k,n)) \models \sim b_{k+1}$, as desired.

If processors do not forget (i.e., $M \in \mathscr{C}_{(\mathrm{nf,sync})}$), then we essentially run through the same arguments as above, backwards. This time, by β_1, we know that there is another run r' such that $(r, n'+f(k,n)) \sim_1 (r', n'+f(k,n))$ and $(M, r', n'+f(k,n)) \models p_k \wedge q_k$. Since processors do not forget, it must also be the case that $(r, n') \sim_1 (r', n')$. By β_1 again, we have that $(M, r', n') \models q_k$ (by the $\Diamond q_k \Rightarrow q_k$ clause) and hence that $(M, r', n') \models \varphi'_{k,n}$. By the induction hypothesis, we know that $(M, r', n') \models p_k$ and $(M, r', n'+i) \models \sim p_k$ for $0<i<f(k,n)$. By β_3, $(M, r', n'+f(k,n)) \models \sim b_{k+1}$. Finally, by β_2, we also have $(M, r, n'+f(k,n)) \models \sim b_{k+1}$, as desired. Similar proofs work for all the other conjuncts.

Finally, let β_4 be

$$\Box(p_{k+1} \Leftrightarrow (p_k \wedge \sim b_{k+1} \wedge \bigcirc(\sim b_{k+1} \, U \, p_k))).$$

Thus, β_4 forces p_{k+1} to be true exactly when the counter is 0.

Let $\varphi_{k+1,n}$ be $\beta_1 \wedge \beta_2 \wedge \beta_3 \wedge \beta_4$. It is now easy to check that $\varphi_{k+1,n}$ satisfies (†), that $\mathrm{ad}(\varphi_{k+1,n}) = k$, and that $|\varphi_{k+1,n}| = O(k+1+n)$.

Next we must show that $\varphi_{k,n}$ is satisfiable. We proceed by induction on k, but we use a stronger induction hypothesis: For each $i \geq 0$ there is an interpreted system M in $\mathscr{C}_{(\mathrm{nf,nl,sync})}$ such that for some run r of M we have that $(M, r, 0) \models \varphi_{k,n}$ and $(M, r, i) \models p_k$. Since $\mathscr{C}_{(\mathrm{nf,nl,sync})} = \mathscr{C}_{(\mathrm{nf,sync})} \cap \mathscr{C}_{(\mathrm{nl,sync})}$, the result also holds for the other classes. The proof is by induction on k.

Fix $i \geq 0$. For $k=1$, M consists of a single run r. We let $(M, r, j) \models p_0$ iff $j = i \pmod n$. Thus, the points where p_0 is true partition r into segments of length n starting at $\mathrm{rem}(i, n)$, where $\mathrm{rem}(i, n)$ is the remainder of the division of i by n. Now we set the truth value of b_0 in such a way that the segments encode an n-bit counter and the segment between i and $i+n-1$ encodes 0. Finally, we let $(M, r, j) \models p_1$ iff $j = i \pmod{f(1,n)}$. Thus, $(M, r, 0) \models \varphi_{1,n}$ and $(M, r, i) \models p_1$.

Suppose we have proved the claim for $\varphi_{k,n}$. By symmetry the claim also holds for $\varphi'_{k,n}$. For $0 \leq l < f(k,n)$, let M_l be an interpreted system with run set R_l, such that for some $r_l \in R_l$ we have that $(M_l, r_l, 0) \models \varphi'_{k,n}$ and $(M_l, r_l, l) \models p_k$. Note that q_k does not occur in $\varphi'_{k,n}$, so we can assume that its truth values in M_l is undefined. We let $(M_l, r_l, j) \models q_k$ for all $j \geq 0$ and $(M_l, r, j) \models \sim q_k$ for $r \neq r_l$ and $j \geq 0$. We construct $M = (R, \pi)$ by taking R to be the union of the R_l's (which we take to be disjoint). We assume that the processor's local states in each of the R_l's are distinct except that processor 1's local state is the same in $r_{l_1}(j)$ and $r_{l_2}(j)$ for $j \geq 0$ and $0 \leq l_1, l_2 < f(k,n)$. Thus, the equivalence relation $\sim_2$ in M is just the union of the equivalence relations in M_l, while the equivalence relation $\sim_1$ in M is the union of

the equivalence relations $\sim_1$ in M_l augmented by the pairs $\langle (r_{l_1}, j), (r_{l_2}, j) \rangle$ for $j \geq 0$ and $0 \leq l_1, l_2 < f(k, n)$. Since in $\varphi'_{k,n}$ every knowledge subformula is governed by a K_2, it follows that $(M, r_l, 0) \models \varphi'_{k,n}$ for all runs r_l in M. Moreover, since every conjunct of $\varphi'_{k,n}$ is prefixed with $\Box$, it actually follows that $(M, r_l, n') \models \varphi'_{k,n}$ for all $n' \geq 0$. Finally, since q_k holds only on the r_l's, it is easy to check that our construction guarantees that $(M, r_l, 0) \models \beta_1$ for all the r_l's.

We now define truth values for b_{k+1} and p_{k+1}. Choose some l with $l = i \pmod{f(k, n)}$. For this choice of l we have $(M, r_l, j) \models p_k$ iff $j = i \pmod{f(k, n)}$. By the induction hypothesis, the points where p_k is true partition r_l into segments of length $f(k, n)$ starting at $\text{rem}(i, f(k, n))$. We set the truth value of b_{k+1} in the run r_l in such a way that these segments encode an $f(k, n)$-bit counter and the segment between i and $i + f(k, n)$ encodes 0. For other runs r in M we let $(M, r, j) \models b_{k+1}$ iff $(M, r_l, j) \models b_{k+1}$. Thus, we have $(M, r_l, 0) \models \beta_2 \wedge \beta_3$. Finally, we let $(M, r, j) \models p_{k+1}$ iff $j = i \pmod{f(k+1, n)}$. Thus, $(M, r_l, 0) \models \varphi_{k+1,n}$ and $(M, r_l, i) \models p_{k+1}$. This completes the proof. ∎

THEOREM 4.2. *Any algorithm that decides whether the $KL_{(m)}$ (resp. $KB_{(m)}$), $m \geq 2$, formula φ is valid must have the following complexity for infinitely many formulas φ with respect to the following classes of interpreted systems.*

1. *$\mathscr{C}_{(\text{nf, nl, sync})}$ and $\mathscr{C}_{(\text{nl, sync})}$: space $\text{ex}(\text{ad}(\varphi), c\,|\varphi|)$ for some constant $c > 0$.*
2. *$\mathscr{C}_{(\text{nf, sync})}$ and $\mathscr{C}_{(\text{nf, sync, uis})}$: time $\text{ex}(\text{ad}(\varphi) + 1, c\,|\varphi|)$ for some constant $c > 0$,*

Proof. As in the proof of Theorem 3.2, the idea is to encode the computations of a Turing machine. For part 1, given a Turing machine **A** that uses space at most $\text{ex}(k, |x|)$ on input x, we show how to uniformly construct a family of formulas $\gamma_{\mathbf{A},x}$ in $KL_{(2)}$ such that (1) $\text{ad}(\gamma_{\mathbf{A},x}) = k$, (2) for some constant $c > 0$, we have $|\gamma_{\mathbf{A},x}| = c\,|x|$, and (3) $\varphi_{\mathbf{A}}$ is satisfiable with respect to $\mathscr{C}_{(\text{nf, nl, sync})}$ (resp. $\mathscr{C}_{(\text{nl, sync})}$) iff **A** accepts x. We then show how to modify the construction to deal with part 2.

Again we assume that **A** has state space S and tape alphabet Γ, and we let CD be the set of cell descriptors as before. We assume without loss of generality that there is a unique accepting state q_a, so that **A** accepts x iff it reaches state q_a at some point in its computation. Fix an input x with $|x| = n$, so that $x = x_1 \cdots x_n$. We know that on input x no more than space $\text{ex}(k, n)$ is used in the computation, so we take all ID's to have length $f(k, n) \geq \text{ex}(k, n)$ (padding with blanks if necessary), separated by #. The crucial difference between this proof and that of Theorem 3.2 is that now we encode the computation of **A** *vertically* along the run, using the yardstick $\varphi_{k,n}$.

The first conjunct of $\gamma_{\mathbf{A},x}$ is the analog of φ_1 in Theorem 3.2. Let γ_1 be

$$\Box\left(\bigvee_{c \in CD} K_1\left(p_c \wedge \sim\left(\bigvee_{d \in CD, d \neq c} p_d\right)\right)\right).$$

If $(M, r, n') \models \gamma_1$, then at most one cell descriptor holds at every point in runs reachable by $\sim_1$ from (r, n') and cell descriptors are uniform across $\sim_1$.

Since we are using $p_\#$ to delimit ID's, we want $p_\#$ to be true at the beginning of the run and at all later points exactly $f(k, n)$ apart. It is here that we need to use our yardsticks. Let γ_2 be

$$p_\# \wedge \Box K_1(q_k \Rightarrow \varphi'_{k,n}) \wedge \sim K_1 \sim (p_k \wedge q_k) \wedge K_1(\Diamond q_k \Rightarrow q_k) \wedge \Box(p_\# \Rightarrow K_1(p_k \wedge q_k \Rightarrow \bigcirc((\sim p_k \wedge \sim p_\#) U (p_k \wedge p_\#)))).$$

Note that the second line in this formula is exactly the formula β_1, and has the job of distributing yardsticks. The third line essentially uses these yardsticks in much the same way as they are used in the formula β_3 to guarantee that the distance between the points, where $p_\#$ is true is $f(k, n)$. Thus, if $M \in \mathscr{C}_{(\mathrm{nl,sync})}$ and $(M, r, n') \models \gamma_1 \wedge \gamma_2$, then for all r' such that $(r, n') \sim_1 (r', n')$ we have that $(M, r', j) \models p_\#$ iff $j = if(k, n)$ for some $i \geq 0$.

We now describe the formulas that force run segments to encode successive ID's of **A**. Since we are encoding the computation of **A** on input x, the first ID is of the form $\#\langle x_1, s_0\rangle x_2 \cdots x_n b^{f(k,n)-n}$ (recall that b is the blank symbol and s_0 is the start state). Let γ_3 be

$$p_\# \wedge \bigcirc(p_{\langle x_1, s_0\rangle} \wedge \bigcirc(p_{x_2} \wedge \bigcirc(\cdots \wedge \bigcirc(p_{x_n} \wedge \bigcirc(p_b U p_\#)) \cdots))).$$

γ_3 forces the first segment to encode the starting ID.

To enforce correct transitions we again use the fact that it suffices to examine triples of cells descriptors, and encode this information in the function N. (Recall that N is the function from the proof of Theorem 3.2 that, given three consecutive cells in an ID, describes the set of possible corresponding three cells in the next ID.) Since all ID's are of length $f(k, n)$, in order to compare corresponding triples of cell descriptors, we have to compare cells that are $f(k, n)$ cells apart. Let γ_4 be

$$\Box\Big(\bigwedge_{i,j,l \in CD} (p_i \wedge \bigcirc p_j \wedge \bigcirc\bigcirc p_l) \Rightarrow \bigvee_{\langle c,d,e\rangle \in N(i,j,l)} K_1(p_k \wedge q_k \Rightarrow \bigcirc(\sim p_k U (p_k \wedge p_c \wedge \bigcirc p_d \wedge \bigcirc\bigcirc p_e)))\Big).$$

γ_4 guarantees that the transitions behave according to N and that they are correct. This follows from the fact that the cell descriptors are uniform over accessible runs and from the properties of p_k.

Finally, we must say that we have an accepting computation. Recall that **A** accepts x iff it eventually goes into state s_a. Let γ_5 be

$$\Diamond\Big(\bigvee_{c \in \Gamma \times \{s_a\}} p_c\Big).$$

Let $\gamma_{\mathbf{A},x}$ be $\bigwedge_{1 \leq i \leq 5} \gamma_i$. Note that $\mathrm{ad}(\gamma) = k$. It is also easy to check that $\gamma_{\mathbf{A},x}$ is satisfied by an interpreted system in $\mathscr{C}_{(\mathrm{nf,nl,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nl,sync})}$) iff **A** accepts x. If **A**

accepts x, consider an interpreted system satisfying $\varphi_{k+1,n}$, as constructed in the proof of Lemma 4.1. As the construction shows, we can actually assume that $(M, r, 0) \models \varphi_{k+1,n}$ for some run r of M. We can now easily lay out the computation of M along run r in such a way as to satisfy $\gamma_{\mathbf{A},x}$. More precisely, we define the truth values of the primitive propositions p_c, $c \in \mathrm{CD}$, so that we can divide each run r' with $(r', 0) \sim_1 (r, 0)$ into a sequence of segments of length $f(k, n)$ encoding the ID's of an accepting run of $\mathbf{A}$ on input x. Since γ_1 requires that the truth values of p_c, $c \in \mathrm{CD}$, are uniform across all $\sim_1$ accessible runs and there is no learning, we must lay out the same computation along all runs. (Note that the fact that $(M, r, 0) \models \varphi_{k+1,n}$ guarantees that β_2, which is part of γ_2, is also satisfied.)

Conversely, it is easy to see that if $M \in \mathscr{C}_{(\mathrm{nf,nl,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nl,sync})}$) and $(M, r, n) \models \varphi_{\mathbf{A},x}$, then, by γ_2, we can partition run r into segments of length $f(k, n)$ starting at (r, n) so that each segment encodes an ID. By γ_3 and γ_4, this sequence of ID's is actually a computation of $\mathbf{A}$ on input x, and by γ_5, it is an accepting computation. The lower bound for $\mathscr{C}_{(\mathrm{nf,nl,sync})}$ and $\mathscr{C}_{(\mathrm{nl,sync})}$ in the case of $KL_{(m)}$, $m \geqslant 2$, immediately follows.

To get the result for $KB_{(m)}$, again we replace occurrences of $\Box$ by $\forall\Box$ and $\bigcirc$ by $\forall\bigcirc$, and the occurrence of $\Diamond$ in γ_5 by $\forall\Diamond$. We leave it to the reader to check that this has the desired effect. (We remark that since we are working in systems with no learning, replacing the $\Diamond$ in γ_5 by $\exists\Diamond$ also works, as the interested reader can verify.)

To get the lower bound in the case of $\mathscr{C}_{(\mathrm{nf,sync})}$ and $\mathscr{C}_{(\mathrm{nf,sync,uis})}$ for the language $KL_{(m)}$, $m \geqslant 2$, we encode alternating Turing machines (ATMs) [CKS]. The difference between this case and the previous case is that the possibility of learning allows us to simultaneously encode different runs of the ATM, in a sense we make precise. We first review the necessary definitions.

In an ATM, there is a subset $U \subseteq S$ of *universal* states. The states in $S - U$ are existential states. There is an *accepting* state s_a. At each configuration the machines has two possible moves. Thus, every ID has two successors. It is convenient to assume that the machine starts in a universal state and alternates at every step, that is, if the ID β is a successor of the ID α, then the state of α is existential iff the state of β is universal. An ATM $\mathbf{A}$ accepts input x if there is an infinite tree, called an *accepting computation tree* of $\mathbf{A}$ on x, labeled by ID's such that

1. the root of the tree is labeled by the initial ID of $\mathbf{A}$ on x,
2. if a node u is labeled with an existential ID α, then u has one child that is labeled by a successor of α,
3. if a node u is labeled by a universal ID α, then u has two children labeled by the two successors of α, and
4. every infinite path through the tree is eventually labeled with an accepting ID.

It is known that a language L is accepted by a deterministic Turing machine within time $O(2^{S(n)})$, where $S(n) \geqslant \log(n)$, iff L is accepted by an alternating Turing machine in space $O(S(n))$ [CKS]. Thus, $\mathrm{ASPACE}(\mathrm{ex}(k, n)) = \mathrm{TIME}(\mathrm{ex}(k+1, n))$.

As before, given an ATM **A** which runs in (alternating) space $\mathrm{ex}(k, n)$ and an input x, we construct a formula $\delta_{\mathbf{A},x}$ which is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf},\mathrm{sync})}$ iff **A** accepts x.

The idea now is to encode different possible computations (i.e., paths in the computation tree) along different runs. Thus, we have to force the properties we required last time to hold on all $\sim_1$ accessible runs, not just one. We accomplish this by prefixing relevant formulas by K_1. Thus, we take δ_1, δ_2, δ_3, and δ_4 to be as $K_1\gamma_1$, $K_1\gamma_2$, $K_1\gamma_3$, and $K_1\gamma_5$, respectively. Besides adding the K_1, we have to slightly modify γ_4 to take into account the alternation. Since an ID has two possible successors, we now have two transition functions rather than one; call them N_1 and N_2. Let δ_5 be

$$K_1\Box \bigwedge_{i,j,l\in CD} \Big((p_i \wedge \bigcirc p_j \wedge \bigcirc\bigcirc p_l) \Rightarrow \bigvee_{1\le q\le 2,\, \langle c,d,e\rangle \in N_q(i,j,l)} K_1(p_k \wedge q_k \Rightarrow \bigcirc(\sim p_k\, U\, (p_k \wedge p_c \wedge \bigcirc p_d \wedge \bigcirc\bigcirc p_e))) \Big).$$

δ_5 guarantees that all transitions behave according to either N_1 or N_2.

We now need a formula that guarantees that when we are in a universal state, both transitions occur. In order to do this, we introduce a new primitive proposition p_u to indicate whether or not we are in a universal state. Let δ_6 be the formula

$$\begin{aligned} & p_u \wedge K_1\Box(p_u \Rightarrow K_1 p_u) \\ & \wedge K_1\Box(\bigcirc \sim p_{\#} \Rightarrow (p_u \Leftrightarrow \bigcirc p_u)) \\ & \wedge K_1\Box(\bigcirc p_{\#} \Rightarrow (p_u \Leftrightarrow \bigcirc \sim p_u)). \end{aligned}$$

The first conjunct of δ_6 says the initial state is universal, the second conjunct says that p_u is uniform over $\sim_1$ accessible runs, the third conjunct says that the truth value of p_u is constant throughout an ID, while the fourth conjunct says that it changes between consecutive IDs.

In order to capture the alternation in **A**, we encode different paths in the computation tree of **A** along different runs in the system. We could not do this when we had no learning, since we insist (in formula γ_1 and δ_1) that the values of p_c for $c \in \mathrm{CD}$ are uniform across $\sim_1$ accessible runs. But once we have learning, we can have the equivalence relations refine, allowing us to encode different paths. Let δ_7 be

$$K_1\Box \bigwedge_{i,j,l\in CD} \Big((p_u \wedge p_i \wedge \bigcirc p_j \wedge \bigcirc\bigcirc p_l) \Rightarrow \bigwedge_{1\le q\le 2,\, \langle c,d,e\rangle \in N_q(i,j,l)} \sim K_1 \sim (p_k \wedge q_k \Rightarrow \bigcirc(\sim p_k\, U\, (p_k \wedge p_c \wedge \bigcirc p_d \wedge \bigcirc\bigcirc p_e))) \Big).$$

Thus, δ_7 says that when we are in a universal state, there are accessible runs that encode both possible transitions. Note that the formula $\delta_1 \wedge \delta_7$ is not satisfiable with respect to $\mathscr{C}_{(\mathrm{nf,nl,sync})}$.

Let $\delta_{\mathbf{A},x}$ be the conjunction of δ_1 through δ_7. A similar argument to that above can now be used to show that $\delta_{\mathbf{A},x}$ is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf,sync})}$ iff **A** accepts input x. We encode different paths in the computation tree, which are $\sim_1$ equivalent as long as the computations agree. A slight modification, along the lines sketched in Lemma 3.4, suffices to deal with $\mathscr{C}_{(\mathrm{nf,sync,uis})}$. We simply add a new first state to every run in which the global states is the same. We leave details to the reader. Finally, to deal with $KB_{(m)}$, we again replace $\Box$, $\bigcirc$, and $\Diamond$ by $\forall\Box$, $\forall\bigcirc$, and $\forall\Diamond$. ∎

Observe that the formula $\varphi_{1,n}$ (and thus $\varphi'_{1,n}$) does not have any occurrences of K_1 or K_2. Thus, the formula needed to encode TMs (resp. ATMs) that run in space $\mathrm{ex}(1, |x|)$ involves only K_1, and not K_2. This gives us the following immediate corollary to the preceding theorem.

COROLLARY 4.3. *Any algorithm that decides whether the* $KL_{(1)}$ *(resp.* $KB_{(1)}$*) formula* φ *is valid must have the following complexity for infinitely many formulas* φ *with respect to the following classes of interpreted systems*:

1. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nl,sync})}$, $\mathscr{C}_{(\mathrm{nl,sync,uis})}$, $\mathscr{C}_{(\mathrm{nf,nl,sync,uis})}$: *space* $\mathrm{ex}(1, c\,|\varphi|)$ *for some constant* $c > 0$ *(i.e., exponential space)*

2. $\mathscr{C}_{(\mathrm{nf,sync})}$ *and* $\mathscr{C}_{(\mathrm{nf,sync,uis})}$: *time* $\mathrm{ex}(2, c\,|\varphi|)$ *for some constant* $c > 0$ *(i.e., double-exponential time).*

Proof. The only cases that are not immediate from the previous proof are $\mathscr{C}_{(\mathrm{nl,sync,uis})}$ and $\mathscr{C}_{(\mathrm{nf,nl,sync,uis})}$ (which are actually the same case, as observed in Proposition 3.9). This case follows from the observation that the standard trick for dealing with a unique initial state—that of adding a new initial state to each run—now works. (It does not work if we have many agents since then adding the initial state imposes extra constraints on the system.) We leave details to the reader. ∎

Remarks. 1. Sistla and German [SG] have independently proved EXPSPACE-completeness for their logic IPTL, which is essentially the same as $KL_{(1)}$ interpreted over interpreted systems in $\mathscr{C}_{(\mathrm{nl,nf,sync})}$.

2. We can avoid the use of the until operator in this proof, using only the $\bigcirc$ and $\Box$ operators, by appropriately encoding a formula of the form $\varphi\, U\, \psi$. The idea is to introduce a new primitive proposition q to encode where $\varphi\, U\, \psi$ is true, and adding the conjuncts $\Box(q \Rightarrow \Diamond\psi)$ and $\Box(q \Leftrightarrow \psi \vee (\varphi \wedge \bigcirc q))$. We leave it to the reader to check that this works.

Again, in the presence of no forgetting, it is easy to drop the assumption that time is synchronous.

PROPOSITION 4.4. *For any formula φ in $KL_{(m)}$ (resp. $KB_{(m)}$), $m \geqslant 2$, there is a formula $sync_\varphi$ such that* $\mathrm{ad}(sync_\varphi) = \mathrm{ad}(\varphi)$, $|sync_\varphi| \leqslant c\,|\varphi|$ *for some constant $c > 0$, and φ is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$) iff $\varphi \wedge sync_\varphi$ is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl})}$, $\mathscr{C}_{(\mathrm{nf,uis})}$).*

Proof. The details are essentially the same as those of Proposition 3.5, except that now we cannot use C, and we must be careful in terms of the length and the alternation depth of $sync_\varphi$. The idea is to replace C by the appropriate number of alternation of K_i's. Given φ, let *tick* be a primitive proposition not appearing in φ. We define $sync_\psi$ for subformulas of ψ by induction on the structure of ψ as

- For a primitive proposition p, let $sync_p = true$
- $sync_{\psi \wedge \psi'} = sync_{\psi U \psi'} = sync_\psi \wedge sync_{\psi'}$
- $sync_{\sim\psi} = sync_{\bigcirc\psi} = sync_\psi$
- $sync_{K_i\psi} = K_i \Box((tick \Rightarrow K_i\, tick) \wedge (\sim tick \Rightarrow K_i tick) \wedge (tick \Rightarrow \bigcirc \sim tick) \wedge (\sim tick \Rightarrow \bigcirc tick) \wedge sync_\psi)$.

The proof that this works is similar to that of Proposition 3.5. Of course, $sync_\varphi$ does not force the whole system to be synchronous, but only those runs relevant to our argument. But this clearly suffices for our purposes. We leave details to the reader. ∎

As an immediate corollary we get

COROLLARY 4.5. *Any algorithm for checking if the $KL_{(m)}$ (resp. $KB_{(m)}$) formula φ is valid must have the following complexity for infinitely many formulas φ with respect to the following classes of interpreted systems:*

1. $\mathscr{C}_{(\mathrm{nf},nl)}$: *space* $\mathrm{ex}(\mathrm{ad}(\varphi), c\,|\varphi|)$ *for some constant $c > 0$ (and* $\mathrm{ex}(1, c\,|\varphi|)$ *if $m = 1$)*

2. $\mathscr{C}_{(\mathrm{nf})}$, $\mathscr{C}_{(\mathrm{nf,uis})}$: *time* $\mathrm{ex}(\mathrm{ad}(\varphi)+1, c\,|\varphi|)$ *for some constant $c > 0$ (and* $\mathrm{ex}(2, c\,|\varphi|)$ *if $m = 1$).*

We can again avoid the use of the nexttime operator in this proof. See the Appendix for details.

We next consider the complexity of the validity problem for $\mathscr{C}_{(\mathrm{nl})}$ and $\mathscr{C}_{(\mathrm{nl,uis})}$. We can combine the ideas of Theorem 3.8 and Theorem 4.2 to get:

THEOREM 4.6. *Any algorithm that decides whether the $KL_{(m)}$ (resp. $KB_{(m)}$) formula φ is valid with respect to $\mathscr{C}_{(\mathrm{nl})}$ (resp. $\mathscr{C}_{(\mathrm{nl,uis})}$) must take space* $\mathrm{ex}(\mathrm{ad}(\varphi), c\,|\varphi|)$ *for some $c > 0$ and infinitely many formulas φ.*

Proof. We modify the construction of the yardsticks $\varphi_{k,n}$ by adding the clause $nonrep \Rightarrow K_1(nonrep)$ to the construction of $\varphi_{1,n}$. (Of course, this means the clause $nonrep \Rightarrow K_2(nonrep)$ is now added to the definition of $\varphi'_{1,n}$, and these clauses will be nested in the definition of $\varphi_{k,n}$ for $k > 1$). We also add the conjunct $sync_{\varphi_A}$ from

Proposition 4.4. This forces weak synchrony in the nonrepeating part of the runs, so that in the nonrepeating part, (the modified) $\varphi_{k,n}$ forces p_k to act like a yardstick.

Finally, we modify the definition of the formula γ_5 in the construction of φ_A of Theorem 4.2 to

$$(nonrep)\, U\Big(\bigvee_{c \in \Gamma \times \{s_\alpha\}} p_c\Big).$$

This forces the run to be nonrepeating until we reach an accepting state. Thus, in the part of the run that is relevant to the lower bound proof, the yardstick construction works right. This gives us the desired lower bound. ∎

As a corollary to the proof, we again get:

COROLLARY 4.7. *Any algorithm that decides whether the* $KL_{(1)}$ *(resp.* $KB_{(1)}$*) formula* φ *is valid with respect to* $\mathscr{C}_{(\mathrm{nl})}$ *(resp.* $\mathscr{C}_{(\mathrm{nl,uis})}$*) takes space* $\mathrm{ex}(1, c\,|\varphi|)$ *for some constant* $c > 0$ *and infinitely many formulas* φ *(i.e., exponential space).*

We remark that in the last section we already discussed the complexity of $KL_{(m)}$ and $KB_{(m)}$ in $\mathscr{C}_{(\mathrm{nl,uis})}$ and $\mathscr{C}_{(\mathrm{nf,nl,uis})}$ (recall that the results here depended on the proofs of Ladner and Reif), so all that remains is to discuss the cases where we do not assume no learning and no forgetting. Just as in the case of $CKL_{(m)}$ and $CKB_{(m)}$, the complexity goes down dramatically in these cases. Note that the next theorem is the only one in which there is a difference between branching time and linear time.

THEOREM 4.8. 1. *The validity problem for* $KL_{(m)}$, $m \geq 1$, *with respect to* $\mathscr{C}$, $\mathscr{C}_{(\mathrm{sync})}$, $\mathscr{C}_{(\mathrm{sync,uis})}$, *and* $\mathscr{C}_{(\mathrm{uis})}$ *is PSPACE-hard.*

2. *The validity problem for* $KB_{(m)}$, $m \geq 1$, *with respect to* $\mathscr{C}$, $\mathscr{C}_{(\mathrm{sync})}$, $\mathscr{C}_{(\mathrm{sync,uis})}$, *and* $\mathscr{C}_{(\mathrm{uis})}$ *is EXPTIME-hard.*

Proof. The result follows immediately from the fact that even without knowledge operators, linear time logic is PSPACE-hard [HR, SC] while branching time logic is EXPTIME-hard [EH1]. It is interesting that the difference here again comes between being able to encode deterministic TMs that run in polynomial space and alternating TMs that run in polynomial space. ∎

5. CONCLUSIONS

We have carefully examined the complexity of reasoning about knowledge and time, indicating how different constraints on the system and different choices of modalities may affect the complexity. We have completely characterized the complexity of the validity problem for a number of combinations of parameters of

interest. We have presented the general framework and lower bound proofs here; the upper bound proofs will appear in a sequel to this paper.

The most significant conclusion we can draw from these results is that by making the assumption that processors do not forget, we greatly complicate the process of reasoning about knowledge and time. While no forgetting is obviously an unrealistic assumption, it is an assumption that is often implicitly made in proofs and specifications. For example, we do not usually say explicitly that the processor still remembers the initial value of the variable x. However, this assumption is often unnecessary. In fact, in almost all cases where no forgetting is implicitly assumed, the assumption can in fact be dropped (only the values of a few variables, which take on only finitely many possible values, really need to be remembered). Our results suggest that, if possible, it may be worthwhile not to make the assumption of no forgetting. (This approach may have the added advantage of making clear exactly how much storage is needed for the algorithm.)

As already mentioned in [HM1], one use that can be made of such a formal model of knowledge and time is in specifying protocols. Once we can specify protocols, we can perhaps use techniques similar to those of [EC, MW] to synthesize a protocol from its specifications. If we use a logic like $CKL_{(m)}$ as a specification language, the observations above suggest that we should not assume no forgetting unless it is absolutely necessary; instead, we should explicitly encode into the specification formula the facts that are not forgotten.

Another line of research inspired in part by the considerations of this paper is that of describing what states of knowledge are attainable in distributed systems. We have given a somewhat abstract notion of model here and have not concerned ourselves, for example, with the mechanics of the transition between global states. If we consider concrete distributed systems, we may want to put restrictions on the possible transitions that can occur or on the possible initial states that arise. As is shown in [FHV2], such restrictions can have a critical impact on the properties of knowledge. Even if we project away time, and only consider formulas involving K_i's, the S5 axioms for knowledge discussed in the previous section, while still sound, may not be complete. Again assumptions such as no forgetting can make a big difference. We refer the interested reader to [FHV2] for more details.

While we have done a relatively exhaustive analysis of the possibilities here, there is perhaps one further complexity issue that might be investigated: that of taking the only temporal modalities to be $\Box$ and its dual $\Diamond$ in the linear time case (and $\forall\Box$, $\exists\Box$, and their duals in the branching time case). In this case we do not have either the nexttime operator or the until operator in the language. Sistla and Clarke have shown that the validity problem for linear time temporal logic with just $\Box$ and $\Diamond$ is NP-complete [SC]. (See [SZ] for a further discusion of the advantages of using just the $\Box$ and $\Diamond$ operators.) We conjecture that even with knowledge operators in the language, the complexity still becomes much simpler without $\bigcirc$ and U. However, since this gives us yet another ninety-six logics to consider, we leave the question for others to look into.

APPENDIX: REMOVING THE NEXTTIME OPERATOR

In this appendix we show how to avoid the use of the nexttime operator in our lower bound results. We first focus on the case with common knowledge and then show how to modify the proof to deal with $KL_{(m)}$ and $KB_{(m)}$. Formally, we prove the following result.

THEOREM A.1. *For all formulas φ in $CKL_{(m)}$ (resp. $CKB_{(m)}$), there is a formula φ_U not involving the nexttime operator such that φ is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$) if and only if φ_U is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl})}$, $\mathscr{C}_{(\mathrm{nf,uis})}$).*

Proof. Assume φ is in $CKL_{(m)}$ (the modifications for $CKB_{(m)}$ are straightforward and left to the reader). Intuitively, the idea is to use a new primitive proposition q (not appearing in φ) to mark off sections of a run where the truth values of all formulas are constant. These q-sections correspond to "points." Corresponding to a subformula of φ of the form $\bigcirc\psi$, we have the translated formula that says that ψ is true at the next q-section.

More formally, we proceed as follows. Given a formula ψ, let ψ_V be the formula that results by recursively replacing all subformulas in ψ of the form $\bigcirc\psi'$ by

$$(q \Rightarrow q\, U\, (\sim q \wedge \psi'_V)) \wedge (\sim q \Rightarrow \sim q\, U\, (q \wedge \psi'_V)).$$

This formula captures the idea that ψ'_V is true in the next q-section. Let $\psi \leqslant \varphi$ denote that ψ is a subformula of φ. Let φ_U be

$$\varphi_V \wedge C\Box(q \Rightarrow Cq)$$

$$\times \bigwedge_{\psi \leqslant \varphi} C\Box((q \wedge \psi_V \Rightarrow \psi_V\, U \sim q) \wedge (\sim q \wedge \psi_V \Rightarrow \psi_V\, U\, q))$$

$$\wedge C\Box(\Diamond q \wedge \Diamond \sim q).$$

Note that the second conjunct says that the truth value of q is common knowledge throughout the system (it is easy to see that $C\Box(q \Rightarrow Cq)$ implies $C\Box(\sim q \Rightarrow C \sim q)$), the third conjunct says that the truth value of ψ_V for all subformulas ψ of φ is constant until the truth value of q changes, and the fourth conjunct says that the truth value of q at any point does eventually change.

We now show that φ is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$) iff φ_U is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl})}$, $\mathscr{C}_{(\mathrm{nf,uis})}$).

Suppose that φ is satisfiable in some interpreted system M in $\mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$). Let M_U be identical to M except that the truth value of q changes at consecutive points along every run in M. Clearly $M \in \mathscr{C}_{(\mathrm{nf})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl})}$, $\mathscr{C}_{(\mathrm{nf,uis})}$). The last three conjuncts of φ_U are then trivially satisfied at every point in M_U. A straightforward induction on the structure of formulas shows that for all subformulas ψ of φ, we have $(M, r, n) \models \psi$ iff $(M_U, r, n) \models \psi_V$. Thus, φ_U is satisfiable in M_U.

For the converse, suppose that φ_U is satisfiable in some interpreted system $M=(R,\pi)$ in $\mathscr{C}_{(\mathrm{nf})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl})}$, $\mathscr{C}_{(\mathrm{nf,uis})}$). Suppose that $(M,r_0,n_0)\models\varphi_U$. Let R' consist of all the runs in R with points reachable from (r_0,n_0). For every run $r'\in R'$, let $(r',n'_{r'})$ be the first point in r' reachable from (r_0,n_0). We can now define the kth q-section of r' in a straightforward way. For example, if $(M,r',n_{r'})\models q$, then the first q-section consists of all the points $(r',n_{r'}),\ldots,(r',n_{r'}+n)$ such that $(M,r',n_{r'}+i)\models q$ for $0\leqslant i\leqslant n$, and $(M,r',n_{r'}+n+1)\models\sim q$. The second q-section consists of all the points $(r',n_{r'}+n+1),\ldots,(r',n_{r'}+n+k)$ such that $(M,r',n_{r'}+n+j)\models\sim q$ for $1\leqslant j\leqslant k$ and $(M,r',n_{r'}+n+k+1)\models q$. We omit a formal inductive definition here. The third conjunct of φ_U guarantees that for all subformulas ψ of φ, the truth values of $\psi_{r'}$ will be constant in every q-section of r'. The fourth conjunct guarantees that every run in R' has infinitely many q-sections. Using the second conjunct we can now show that the q-sections respect the $\sim_i$ equivalence relations in the following sense:

LEMMA A.2. *Suppose* $r_1, r_2\in R'$, $(r_1,n_1)\sim_i(r_2,n_2)$, *and* (r_1,n_1) *is in the* k*th* q*-section of* r_1, *then* (r_2,n_2) *is in the* k*th* q*-section of* r_2. *Moreover, if* (r_1,n_1) *is the first point in the* k*th* q*-section of* r_1, *then* $(r_1,n_1)\sim_i(r_2,n_3)$, *where* (r_2,n_3) *is the first point in the* k*th* q*-section of* r_2.

Proof. We proceed by induction on k. Without loss of generality we can assume that $(M,r_1,n_1)\models q$. The case $k=0$ breaks down into two subcases. First suppose that (r_1,n_1) is the first point in the 0th q-section of r_1. Let (r_2,n_3) be the first point in the 0th q-section of r_2. Clearly $n_3\leqslant n_2$, so, by our assumption of no forgetting, it follows that for some point $n_4\leqslant n_1$, we must have $(r_1,n_4)\sim_i(r_2,n_3)$. But, by assumption, (r_1,n_1) is the first point reachable in r_1 reachable from (r_0,n_0), so we must have $n_1=n_4$. Thus, $(r_1,n_1)\sim_i(r_2,n_3)$. By transitivity, we also have $(r_2,n_2)\sim_i(r_2,n_3)$, so, from the assumption that processors do not forget, we get $(r_2,n')\sim_i(r_2,n_3)$ for all n' with $n_3\leqslant n'\leqslant n_2$. Since the truth value of q is common knowledge, q must be true at (r_2,n_3), (r_2,n_2), and at all points in between. Thus, (r_2,n_3) and (r_2,n_2) must be in the same q-section. In particular, it follows that (r_2,n_2) is in the 0th q-section.

Note that from the observations in the previous paragraph it follows that no point in the kth q-section of r_2 can be indistinguishable by i from a point before the first point in the 0th q-section of r_1. (To see this, note that if not, we can use the assumption of no forgetting to show that the first point in the 0th q-section of r_2 is indistinguishable from a point previous to the first point in the first q-section of r_1. This would mean that a point previous to the first point in the first q-section of r_1 is reachable from (r_0,n_0), and this is a contradiction.) A similar remark holds when we reverse the roles of r_1 and r_2.

Now consider the case where (r_1,n_1) is not necessarily the first point in the 0th q-section. If (r_2,n_2) is not in the 0th q-section of r_2, there is a point (r_2,n_4) between the first point in the 0th q-section of r_2 and (r_2,n_2) such that $(M,r_2,n_4)\models\sim q$. By the assumption of no forgetting, there must be a point (r_1,n_5) with $n_5\leqslant n_1$ such

that $(r_1, n_5) \sim_i (r_2, n_4)$. By the observations of the previous paragraph, (r_1, n_5) must be in the 0th q-section of r_1. Thus, $(M, r_1, n_5) \models q$. But this contradicts the assumption that q is common knowledge. Thus, it must be the case that (r_2, n_2) is in the 0th q-section of r_2.

For $k > 0$ the proof is similar. Again, we first suppose (r_1, n_1) is the first point in the kth q-section of r_1. It must be the case that (r_2, n_2) is in the lth q-section for $l \geq k$ (if we had $l < k$ it would contradict the induction hypothesis). Thus, if (r_2, n_3) is the first point in the kth q-section of r_2 we must have that $n_3 \leq n_2$. By the assumption of no forgetting, it follows that there is some point (r_1, n_4) such that $(r_1, n_4) \sim_i (r_2, n_3)$ and $n_4 \leq n_1$. By the induction hypothesis, it must be the case that $n_4 = n_1$ (since a point cannot simultaneously be in two different q-sections). The same arguments as above can now be used to show that (r_2, n_3) and (r_2, n_2) are in the same q-section. The case where (r_1, n_1) is not the first point in the kth q-section is similar to the case $k = 0$. We leave details to the reader. ∎

We now build an interpreted system where φ is satisfied by collapsing all q-sections to single points. Given a run $r \in R'$, let $f(r)$ be the run such that $f(r)(n)$ is the global state at the first point in the nth q-section of r. Define π' on points in $f(R') \times \mathbb{N}$ so that $\pi'(f(r), n) = \pi(r, k)$, where (r, k) is the first point in the nth q-section of r. Let $M' = (f(R'), \pi')$. From Lemma A.2, it easily follows that $f(R')$ is a synchronous system where processors do not forget. Moreover, if processors do not learn in R, then they do no learn in $f(R)$ either. If R has a unique initial state, we can easily add one to $f(R)$ as described in the proof of Lemma 3.4.

We leave it to the reader to check that for all subformulas ψ of φ and all runs $r \in R'$, we have $(M', f(r), n) \models \psi$ iff $(M, r, k) \models \psi_V$, where (r, k) is the first point in the nth q-section of r. (Here we use the third conjunct of φ_U, which says that the truth value of ψ_U is constant over every q-section for all subformulas ψ of φ). Thus, $(M', f(r_0), 0) \models \varphi$. ∎

Using this result we can get the $\prod_1^1$ lower bound for $\mathscr{C}_{(\mathrm{nf})}$, $\mathscr{C}_{(\mathrm{nf,nl})}$, and $\mathscr{C}_{(\mathrm{nf,uis})}$ in the language $CKL_{(m)}$ (resp. $CKB_{(m)}$), $m \geq 2$, even without the nexttime operator in the language. (Indeed, we can also use the ideas of the proof to eliminate the nexttime operator in favor of the until operator even in synchronous systems, although we omit details here.) We remark that although we do not know how to replace occurrences of $\bigcirc$ by *until*, when we restrict attention to $\mathscr{C}_{(\mathrm{nl})}$, the ideas in the proof of Theorem A.1 do apply to the nonrepeating part, so we can replace the use of the nexttime operator in Theorem 3.8. Thus, the $\prod_1^1$ lower bound for $\mathscr{C}_{(\mathrm{nl})}$ also holds even if we do not have the nexttime operator in the language.

We next turn our attention to the languages $KL_{(m)}$ and $KB_{(m)}$.

THEOREM A.3. *For all formulas φ in $KL_{(m)}$ (resp. $KB_{(m)}$), there is a formula φ_u not involving the nexttime operator such that* $\mathrm{ad}(\varphi_u) = \mathrm{ad}(\varphi)$, $|\varphi_u| \leq c\,|\varphi|$ *for some constant $c > 0$, and φ is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf,sync})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl,sync})}$, $\mathscr{C}_{(\mathrm{nf,sync,uis})}$) if and only if φ_u is satisfiable with respect to $\mathscr{C}_{(\mathrm{nf})}$ (resp. $\mathscr{C}_{(\mathrm{nf,nl})}$, $\mathscr{C}_{(\mathrm{nf,uis})}$).*

Proof. The idea is to replace the use of C in φ_U by enough alternation K_i's to give us the result we need. We also have to be a little careful to make sure that $|\varphi_u| \leqslant c\,|\varphi|$ for some constant $c > 0$. In order to do this, we introduce a new primitive proposition q_ψ for each subformula ψ of φ. We then inductively define the formula ψ_t for each subformula ψ of φ; φ_u will be $q_\varphi \wedge \varphi_t$. Roughly speaking, ψ_t says that the truth value of q_ψ is constant along q-sections, and defines the truth value of q_ψ in terms of that of $q_{\psi'}$ for subformulas ψ' of ψ. Let $\text{const}(q_\psi)$ be an abbreviation for the formula

$$\Box((q \wedge q_\psi \Rightarrow q_\psi\, U \sim q) \wedge (\sim q \wedge q_\psi \Rightarrow \sim q_\psi\, U\, q)).$$

This formula essentially corresponds to the third conjunct in φ_U.

We now construct ψ_t inductively as follows:

- $p_t = \text{const}(q_p) \wedge \Box(\Diamond q \wedge \Diamond \sim q)$ for a primitive proposition p
- $(\sim\psi)_t = \text{const}(q_{\sim\psi}) \wedge \Box(q_{\sim\psi} \Leftrightarrow \sim q_\psi) \wedge \psi_t$
- $(\psi \wedge \psi')_t = \text{const}(q_{\psi \wedge \psi'}) \wedge \Box(q_{\psi \wedge \psi'} \Leftrightarrow q_\psi \wedge q_{\psi'}) \wedge \psi_t \wedge \psi'_t$
- $(\bigcirc\psi)_t = \text{const}(q_{\bigcirc\psi}) \wedge \Box(((q_{\bigcirc\psi} \wedge q) \Leftrightarrow q\, U\, (\sim q \wedge q_\psi)) \wedge ((q_{\bigcirc\psi} \wedge \sim q) \Leftrightarrow \sim q\, U\, (q \wedge q_\psi))) \wedge \psi_t$
- $(\psi\, U\, \psi')_t = \text{const}(q_{\psi U \psi'}) \wedge \Box(q_{\psi U \psi'} \Leftrightarrow q_\psi\, U\, q_{\psi'}) \wedge \psi_t \wedge \psi'_t$
- $(K_i\psi)_t = \text{const}(q_{K_i\psi}) \wedge \Box(q_{K_i\psi} \Leftrightarrow K_i q_\psi) \wedge K_i\psi_t \wedge K_i\Box(q \Leftrightarrow K_i q)$.

It is easy to see that $|\psi_t| \leqslant c\,|\psi|$ for some $c > 0$ and that $\text{ad}(\psi_t) = \text{ad}(\psi)$. Finally, we take $\varphi_u = q_\varphi \wedge \varphi_t$. We leave it to the reader to check that φ_u has all the required properties. The proof parallels that for φ_U. ∎

We remark that using these ideas we can also modify the Ladner and Reif proof that we used as the basis of our undecidability results for the classes $\mathscr{C}_{(\text{nl})}$ and $\mathscr{C}_{(\text{nl, uis})}$ and the languages $KL_{(m)}$, $KB_{(m)}$, $m \geqslant 2$, to eliminate the use of the nexttime operator. We omit the details here.

ACKNOWLEDGMENTS

We would like to thank Martin Abadi, Karl Abrahamson, Ron Fagin, and Ed Wimmers for their comments on a previous draft of this paper.

REFERENCES

[Ab] K. R. ABRAHAMSON, "Decidability and Expressiveness of Logics of Processes," Ph.D. thesis, University of Washington Technical Report 80-08-01, 1980.

[CM] M. CHANDY AND J. MISRA, How processes learn, *Distrib. Comput.* **1**, 1 (1986), 40–52.

[CKS] A. K. CHANDRA, D. C. KOZEN, AND L. J. STOCKMEYER, Alternation, *J. Assoc. Comput. Mach.* **28** (1981), 114–133.

[DM] C. DWORK AND Y. MOSES, Knowledge and common knowledge in a Byzantine environment I: Crash failures, *in* "Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference (J. Y. Halpern, Ed.), pp. 149–169, Kaufmann, Los Angeles, 1986.

[Em] E. A. EMERSON, Alternative semantics for temporal logics, *Theoret. Comput. Sci.* **26** (1983), 121–130.

[EC] E. A. EMERSON AND E. M. CLARKE, Using branching time temporal logic to synthesize synchronization skeletons, *Sci. Comput. Programming* **2** (1982), 241–266.

[EH1] E. A. EMERSON AND J. Y. HALPERN, Decision procedures and expressiveness in the temporal logic of branching time, *J. Comput. System Sci.* **30**, No. 1 (1985), 1–24.

[EH2] E. A. EMERSON AND J. Y. HALPERN, "Sometimes" and "not never" revisited: On branching vs. linear time, *J. Assoc. Comput. Math.* **33**, No. 1 (1986), 151–178.

[FHV1] R. FAGIN, J. Y. HALPERN, AND M. Y. VARDI, A model-theoretic analysis of knowledge, *in* "Proceedings, 25th Annual IEEE Symposium on Foundations of Computer Science, 1984," pp. 268–278.

[FHV2] R. FAGIN, J. Y. HALPERN, AND M. Y. VARDI, What can machines know? On the epistemic properties of machines, *in* "Proceedings, AAAI-86, 1986," pp. 428–434. (A revised appears as IBM Research Report RJ 6250, 1988.)

[FI1] M. J. FISCHER AND N. IMMERMAN, Foundations of knowledge for distributed systems, *in* "Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference" (J. Y. Halpern, Ed.), pp. 171–185, Kaufmann, Los Angeles, 1986.

[FI2] M. J. FISCHER AND N. IMMERMAN, Interpreting logics of knowledge in propositional dynamic logic with converse, *Inform. Process. Lett.* **25**, No. 3 (1987), 175–182.

[GPSS] D. GABBAY, A. PNUELI, S. SHELAH, AND J. STAVI, On the temporal analysis of fairness, *in* "Proceedings, 7th ACM Symposium on Principles of Programming Languages, 1980," pp. 163–173.

[Hal] J. Y. HALPERN, Using reasoning about knowledge to analyze distributed systems, *in* "Annual Review of Computer Science, Vol. 2," Annual Rev., Palo Alto, CA, 1987.

[HF] J. Y. HALPERN AND R. FAGIN, A formal model of knowledge, action, and communiction in distributed systems: Preliminary report, *in* "Proceedings, 4th ACM Symposium on Principles of Distributed Computing, 1985," pp. 224–236.

[HM1] J. Y. HALPERN AND Y. O. MOSES, Knowledge and common knowledge in a distributed environment, *in* "Proceedings, 3rd ACM Symposium on Principles of Distributed Computing, 1984," pp. 50–61; revised report, IBM Research Report RJ 4421, Jan. 1986.

[HM2] J. Y. HALPERN AND Y. O. MOSES, A guide to the modal logics of knowledge and belief: Preliminary report, *in* "Proceedings, 9th International Joint Conference on Artificial Intelligence, 1985," pp. 480–490.

[HR] J. Y. HALPERN AND J. H. REIF, The propositional dynamic logic of deterministic well-structured programs, *Theoret. Comput. Sci.* **27** (1983), 127–165.

[HV] J. Y. HALPERN AND M. Y. VARDI, The complexity of reasoning about knowledge and time: Extended abstract, *in* "Proceedings, 18th Annual ACM Symposium on Theory of Computing, 1986," pp. 304–315.

[Har] D. HAREL, Recurring dominoes: Making the highly undecidable highly understandable, *in* "Proceedings, Conference on Foundations of Computing Theory" (M. Karpanski, Ed.), Lecture Note in Computer Science Vol. 158, pp. 177–194, Springer-Verlag, New York/Berlin, 1983.

[HPS] D. HAREL, A. PNUELI, AND J. STAVI, Propositional dynamic logic of nonregular programs, *J. Comput. System Sci.* **26**, No. 2 (1983), 222–243.

[HPV] D. HAREL, A. PNUELI, AND M. Y. VARDI, Two-dimensional temporal logic and PDL with intersection, unpublished.

[LR] R. LADNER AND J. H. REIF, The logic of distributed protocols, *in* "Theoretical Aspects of Reasoning About Knowledge: Proceedings, 1986 Conference" (J. Y. Halpern, Ed.), pp. 207–221, Kaufmann, Los Angeles, 1986.

[La1] L. LAMPORT, What good is temporal logic?, *in* "Information Processing 83" (R. E. A. Mason, Ed.), New York, Elsevier, pp. 657–668, 1983.

[La2] L. LAMPORT, "Sometime" is sometimes "not never": On the temporal logic of programs, *in* "Proceedings, 7th ACM Symposium on Principles of Programming Languages, 1980," pp. 174–185.

[Le1] D. J. LEHMANN, Knowledge, common knowledge, and related puzzles, *in* "Proceedings, 3rd ACM Symposium on Principles of Distributed Computing, 1984," pp. 62–67.

[Le2] D. J. LEHMANN, Talk given at the "3rd ACM Symposium on Principles of Distributed Computing," August 1984.

[MW] Z. MANNA AND P. L. WOLPER, Synthesis of communicating processes from temporal logic specifications, *ACM Trans. Programming Lang. Systems* **6**, No. 1 (1984), 68–93.

[Mo] R. C. MOORE, Reasoning about knowledge and action, *in* "Formal Theories of the Commonsense World" (J. Hobbs and R. C. Moore, Eds.), Ablex, Norwood, NJ, 1985.

[MDH] Y. O. MOSES, D. DOLEV, AND J. Y. HALPERN, Cheating husbands and other stories: A case study of knowledge, action, and communication, *Distrib. Comput.* **1**, No. 3 (1986), 167–176.

[MT] Y. MOSES AND M. TUTTLE, Programming simultaneous actions using common knowledge, *in* "Proceedings, 27th Annual IEEE Symposium on Foundations of Computer Science, 1986," pp. 208–221.

[PR] R. PARIKH AND R. RAMANUJAM, Distributed processing and the logic of knowledge, *in* "Proceedings, Workshop on Logics of Programs" (R. Parikh, Ed.), pp. 256–268, Lecture Notes in Computer Science Vol. 193, Springer-Verlag, New York/Berlin, 1985.

[PSL] M. PEASE, R. SHOSTAK, AND L. LAMPORT, Reaching agreement in the presence of faults, *J. Assoc. Comput. Mach.* **27**, No. 2 (1980), 228–234.

[Pn] A. PNUELI, Linear and branching structures in the semantics and logics of reactive systems, *in* "Proceedings, 12th International Colloquium on Automata, Languages and Programming," Lecture Notes in Computer Science Vol. 194, Springer-Verlag, pp. 15–32, New York/Berlin, 1985.

[RS] J. REIF AND A. P. SISTLA, A multiprocess network logic with temporal and spatial modalities, *J. Comput. System Sci.* **30**, No. 1 (1985), 41–53.

[Rog] H. ROGERS, JR., "Theory of Recursive Functions and Effective Computability," McGraw–Hill, New York, 1967.

[Ros] S. ROSENSCHEIN, Formal theories of knowledge in AI and robotics, *New Generation Comput.* **3** (1985), 345–357.

[RK] S. ROSENSCHEIN AND L. KAELBLING, The synthesis of digital machines with provable epistemic properties, *in* "Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference" (J. Y. Halpern, Ed.), pp. 83–97, Kaufmann, Los Angeles, 1986.

[Sa] M. SATO, A study of Kripke-style methods of some modal logics by Gentzen's sequential method, *Publ. Res. Inst. Math. Sci.* **13**, No. 2 (1977).

[SC] A. P. SISTLA AND E. M. CLARKE, The complexity of propositional linear temporal logics, *J. Assoc. Comput. Mach.* **32**, 3 (1985), 733–749.

[SG] A. P. SISTLA AND S. M. GERMAN, Reasoning with many processes, *in* "Proceedings, Second IEEE Symposium on Logic in Computer Science, 1987," pp. 138–153.

[SZ] A. P. SISTLA AND L. D. ZUCK, On the eventually operator in temporal logic, *in* "Proceedings, Second IEEE Symposium on Logic in Computer Science, 1987," pp. 153–166.

[St] L. J. STOCKMEYER, "The Complexity of Decision Problems in Automata Theory and Logic," Ph.D. dissertation, Technical Report MAC MIT TR-133, MIT, 1974.

[VS] M. Y VARDI AND L. J. STOCKMEYER, Improved upper and lower bounds for modal logics of programs, *in* "Proceedings, 17th ACM Symposium of Computing, 1985," pp. 240–251.

Printed in Belgium