Synthesis of Distributed Systems from Knowledge-Based Specifications[1]

UNSW-CSE-TR-0504

Ron van der Meyden
School of Computer Science and Engineering,
University of New South Wales &
National ICT Australia
meyden@nicta.com.au

Thomas Wilke
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
wilke@ti.informatik.uni-kiel.de

February 9, 2005

[1]Work supported by a grant from the Australian Research Council. National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

Abstract

We consider the problem of synthesizing protocols in a distributed setting satisfying specifications phrased in the logic of linear time and knowledge. In general, synthesis in distributed settings is undecidable already for linear-time temporal logic specifications, but there exist special cases in which synthesis from linear-time temporal logic specifications is known to be decidable. On the basis of these results and a result on the decidability of synthesis of temporal and knowledge specifications in systems with a single agent, van der Meyden and Vardi [CONCUR 96] conjectured that synthesis of temporal and knowledge specifications would be decidable in two classes of environments: hierarchical environments, in which each agent in a linear sequence observes at least as much as the preceding agents, and broadcast environments, in which all communication is constrained to be by synchronous broadcast. We show that this conjecture is true in the case of broadcast environments, but false in the case of even a very simple type of hierarchical environment, where only two agents are involved, one of which observes every aspect of the system state and one of which observes nothing of it. Nevertheless, synthesis from linear-time logic specifications is decidable in hierarchical environments. Moreover, for specifications that are positive in the knowledge modalities, the synthesis problem can be reduced to the same problem for the logic of linear time. We use these facts to conclude the decidability in hierarchical systems of a property closely related to nondeducibility on strategies, a notion that has been studied in computer security.

Contents

1 Introduction

In program synthesis, one starts with a specification of a system and attempts to derive a program that implements this specification. This problem is particularly challenging in the context of *open systems*, which are required to respond appropriately to a sequence of inputs provided by an environment that is not under the full control of the program to be synthesized. A specification of an open system is said to be *realizable* if there exists a protocol with the property that the specification is satisfied, whatever the behaviour of the environment. A problem that has received significant attention is the synthesis of open systems from temporal logic specifications [Ant95, AE98, AE01, AAE04, AVW03, EC82, FF90, MW84, KV97, PR89a,

PR89b, Var95, WTD91, Wal04]. It has been shown that, under certain circumstances, automata-theoretic ideas can be applied to automate this synthesis process.

Often, designers of concurrent systems reason informally but explicitly not just about time, but also about the uncertainty that systems components have about the global state of the system. One finds statements such as "if process X knows that the transaction will be aborted, it should rollback its local contribution and terminate immediately." Such assertions can be made formal in the logic of knowledge [FHMV95]. A variety of distributed protocols have been studied using the logic of knowledge, and it has been argued that such an approach leads to a more perspicuous presentation of the design, and to implementations in which components are optimal in their use of information—see [FHMV95] for many citations.

The logic of knowledge also provides expressive capabilities useful for the specification of information flow in security protocols [HO03, MS04]. For example, the Dining Cryptographers protocol [Cha88] provides a mechanism for a sender to communicate a message anonymously. This can be specified in the logic of knowledge by the requirement that all parties come to know a fact p, but all agents except the sender should not come to know the identity of the sender [MS04].

A common assumption in both types of applications of the logic of knowledge is that agents have *perfect recall* of their observations, i.e., that they keep a complete record of all events they have observed, and determine what they know using this complete record. This assumption is of particular relevance both when optimal use of information acquired is a design concern, and when we wish to determine the capabilities of the most powerful possible adversary in a security analysis.

Realizability of specifications in the logic of knowledge and linear time under the assumption of perfect recall has been studied by van der Meyden and Vardi [MV98]. They showed that the realizability problem is decidable for such specifications in the context of an open system involving a single agent.

In general, realizability for specifications in the logic of knowledge and time is undecidable when there is more than one agent, because the problem is already undecidable even for two-agent systems and specifications involving only linear-time temporal operators, by results of Pnueli and Rosner [PR90]. However, a number of cases have been identified for temporal specifications where multi-agent temporal specifications are decidable. Pnueli and Rosner identify a class of architectures for process communication as yielding a decidable case for synthesis from linear-time temporal logic specification. One particular example in this class is pipelines, in which communication is

constrained to occur only along a chain of processes. The characterization of the decidable cases has recently been refined [Mad01].

These positive results for realizability from temporal specifications led van der Meyden and Vardi to conjecture that similar results could be found for specifications in the logic of knowledge and time. In particular, they proposed that *hierarchical systems* and *broadcast* systems might be cases where realizability of specifications in the logic of knowledge and linear time could be found to be decidable. Hierarchical systems are systems in which agents can be linearly ordered in such a way that each agent in the sequence observes (hence knows) at least as much as the preceding agents. An example of an hierarchical system is a system of three agents with security clearances to read unclassified, secret and top-secret documents, respectively (where a security clearance implies a capability to read documents at or below the security level.)[1]

Broadcast systems are systems in which agents maintain a private state, information about which they can communicate to other agents, but only by a broadcast to all other agents, which is synchronous in the sense that all agents receive a broadcast at the same time. It has been shown in other contexts that these assumptions lead to lowered complexity of a variety of problems in the logic of knowledge [EvdMS03, Mey96], so the conjecture that they might make realizability decidable is reasonable.

We provide in this paper a complete resolution of van der Meyden and Vardi's conjectures. In the case of broadcast systems, we show that the conjecture is true. However, the conjecture concerning hierarchical systems is false: realizability for specifications in the logic of knowledge and linear time is undecidable in hierarchical systems with two or more agents. On the positive side, however, we identify a special class of formulas: those in which the knowledge operators have only positive occurrences, and show that for such formulas the realizability problem can be reduced to a problem of realizability of specifications in linear-time temporal logic. This result enables known cases of decidable realizability problems for linear-time temporal logic to be transferred to give decidable cases of realizability for the logic of knowledge and linear time. In particular, we show that realizability of linear-time temporal logic formulas is decidable in hierarchical systems, so the reduction yields the decidability of realizability of positive specifications in the logic of linear time and knowledge in hierarchical systems. As an application of this result, we establish the decidability of a property closely related to the

[1]This definition differs from the definition of hierarchical system shown by Pnueli and Rosner to yield a decidable class of architectures. It resembles their pipelines, but has a faster flow of information since visibility is transitive.

notion of "nondeducibility on strategies" [Wit90] from the computer security literature.

2 Basic definitions and main result

In this section we lay out the definition of the synthesis problem we study, provide an example that illustrates how it may express the type of information flow property that has been studied in the computer security literature, and state the main results of the paper.

2.1 The logic of linear time and knowledge

We fix a finite set *Prop* of propositional variables and a finite number n of agents, which are simply numbered 1 through n.

The syntax of the logic of linear time and knowledge is defined as follows. It is built from the elements of *Prop* using boolean connectives, the usual temporal operators such as X, G, U, ..., and the unary operators K_i for $i \in [n]$. For each $i \in [n]$, the operator L_i is also allowed; it is the dual of K_i, that is, it is an abbreviation for $\neg\mathsf{K}_i\neg$. For later use, we also mention that we use R to denote the dual of U, that is, $\varphi\,\mathsf{R}\,\psi$ stands for $\neg(\neg\varphi\,\mathsf{U}\,\neg\psi)$.

An *interpreted system* is a tuple

$$\mathcal{I} = (\mathcal{R}, \pi, \{\sim_i\}_{i\in[n]}) \tag{1}$$

where

- $\mathcal{R}$ is a set of so-called *runs*,
- $\pi\colon \mathcal{R} \times \mathbf{N} \to 2^{Prop}$ is a function which assigns to each *point* (r, m) of a run the propositions that hold true in it, and
- $\{\sim_i\}_{i\in[n]}$ is a family of *indistinguishability relations* on the points of all runs.

Each indistinguishability relation $\sim_i$ is required to be an equivalence relation; the relation $\sim_i$ relates the points that are indistinguishable by agent i.

Given a point (r, m) of an interpreted system $\mathcal{I}$, we define what it means for a formula φ in the logic of linear time and knowledge to hold at this point, denoted $\mathcal{I}, (r, m) \models \varphi$:

- $\mathcal{I}, (r, m) \models p$ if $p \in \pi((r, m))$,
- $\mathcal{I}, (r, m) \models \mathsf{X}\psi$ if $\mathcal{I}, (r, m + 1) \models \psi$,

- $\mathcal{I}, (r, m) \models \psi \,\mathsf{U}\, \chi$ if there exists $m' \geq m$ such that $\mathcal{I}, (r, l) \models \psi$ for all l with $m \leq l < m'$ and $\mathcal{I}, (r, m') \models \chi$,
- $\mathcal{I}, (r, m) \models \mathsf{K}_i \psi$ if $\mathcal{I}, (r', m') \models \psi$ for all (r', m') with $(r, m) \sim_i (r', m')$.

The boolean connectives are dealt with as usual.

We write $\mathcal{I}, r \models \varphi$ if $\mathcal{I}, (r, 0) \models \varphi$ and $\mathcal{I} \models \varphi$ if $\mathcal{I}, r \models \varphi$ for all runs r of $\mathcal{I}$.

2.2 Systems with perfect recall in finite-state environments

A *signature* of size n is a family $\{ACT_i\}_{i \in \{e,1,\ldots,n\}}$ where each set ACT_i is a finite, non-empty set of actions for the *environment* e or *agent* $i \in [n]$. The set of *joint actions* of such a signature is defined by $ACT = ACT_e \times ACT_1 \times \cdots \times ACT_n$. When $\boldsymbol{a}$ denotes a joint action, we write $\boldsymbol{a}_i$ for the action of agent i in $\boldsymbol{a}$.

An *environment* over a signature as just described is a tuple

$$E = (S, I, P_e, \tau, \{O_i\}_{i \in [n]}, Prop, \pi) \tag{2}$$

where

- S is a finite set of *states,*
- $I \subseteq S$ is the set of *initial states,*
- $P_e \colon S \to 2^{ACT_e}$ is the *protocol of the environment,* which says which actions can be performed by the environment in a given state,
- $\tau \colon ACT \to (S \to S)$ is the *transition function,* which, for every joint action $\boldsymbol{a}$ specifies a transition function $\tau(\boldsymbol{a})$,
- $\{O_i\}_{i \in [n]}$ is a family of *observation functions* $O_i \colon S \to \mathcal{O}$ for some set $\mathcal{O}$ of *observations,*
- P is a finite set of propositions, and
- $\pi_e \colon S \to 2^{Prop}$ is an *interpretation function* which assigns to each state the propositions that hold in it.

We require $P_e(s) \neq \emptyset$ for each $s \in S$. We also note that $\tau(\boldsymbol{a})(s)$ needs only be defined if $\boldsymbol{a}_e \in P_e(s)$.

A *run* of such an environment is an infinite sequence $s_0, s_1, s_2, \ldots$ such that $s_0 \in I$ and such that for all m there exists $\boldsymbol{a} \in ACT$ with $\boldsymbol{a}_e \in P_e(s_m)$

and $\tau(\boldsymbol{a})(s_m) = s_{m+1}$. When r denotes such a run and (r, m) is a point, we set $r(m) = s_m$.

To obtain an interpreted system, we set $\pi(r, m) = \pi_e(r(m))$ for every point (r, m). Further, we let $O_i(r, m) = O_i(r(0))O_i(r(1)) \dots O_i(r(m))$ for every agent i and call $O_i(r, m)$ the *local state* of agent i at point (r, m). Using this notation, we define $\sim_i$ by $(r, m) \sim_i (r', m')$ iff $O_i(r, m) = O_i(r', m')$. (Note that this implies $m = m'$.) This indistinguishability relation is called *synchronous perfect recall.* The resulting interpreted system is denoted $\mathcal{I}(E)$.

2.3 Hierarchical and broadcast environments

Our results focus on two classes of environments, hierarchical environments and broadcast environments, which were conjectured by van der Meyden and Vardi to yield decidable cases of synthesis from knowledge-based specifications in a multi-agent setting.

Hierarchical environments [EvdMS03] are those in which for all states s and agents $i \in [n-1]$, we have that $O_i(s) = O_i(t)$ implies $O_{i+1}(s) = O_{i+1}(t)$. Intuitively, this means that each agent in the sequence observes not more than the preceding agent: if agent i is not able to distinguish states s and t by observation, then neither is agent $i + 1$. Clearly the same property holds for the indistinguishability relations on points derived using the assumption of perfect recall. The name derives from the fact that the equivalence classes of these relations form a hierarchically nested collection of sets.

Broadcast environments [Mey96] model situations in which agents may maintain private information, but where the only means by which this information can be communicated is by synchronous simultaneous broadcast to all agents. This sort of situation arises in systems in which components communicate by means of a shared bus, and protocols that operate in a sequence of synchronised *rounds* with broadcasts between rounds may also be modelled in this way. Broadcast environments are able to represent a variety of games of incomplete information, including Battleships and Bridge.

Our definition of broadcast environment in this paper will be slightly more general than that in [Mey96], where the definition was tailored so as to allow derivation of a result concerning non-deterministic knowledge based programs. This motivated the restriction that all actions having an effect on the shared state of the system be immediately visible to all agents. In the present context, we deal only with deterministic protocols, and we can relax this restriction so as to allow unobserved effects on the shared state. We still require that all agents always make the same observation of the shared state.

Formally, we define a broadcast environment to be an environment $E = (S, I, P_e, \tau, \{O_i\}_{i\in[n]}, \mathit{Prop}, \pi)$ of a specific structure we now describe. The

states of the environment consist of a private state for each of the agents plus a state for the remainder of the system. For each agent $i \in [n]$, we assume that there exists a set S_i of *instantaneous private states*, intuitively representing the state of the private objects maintained by the agent. The private states S_i will be observable and modifiable by agent i only. We also assume a set of *shared*, or *common* states S_0, representing the remainder of the system, which is under the shared control of the agents. The set of all states is taken to be the cartesian product $S = S_0 \times S_1 \times \ldots \times S_n$. We define *agent i's private state* at a state $s = (s_0, s_1, \ldots, s_n)$, denoted $\boldsymbol{p}_i(s)$, to be the private state s_i. We allow the set I of initial states to be any nonempty subset of S.

In a broadcast environment, agents are able to observe their own private state, as well as some aspects of the shared state. However, we assume that all agents make the *same* observation of the shared state. Formally, we suppose that there exists a "common" observation function O_c mapping each shared state in S_0 to some observation. It is convenient to extend this mapping to states in S by $O_c((s_0, s_1, \ldots, s_n)) = O_c(s_0)$. Using this, agent i's observation function is defined by $O_i(s) = (O_c(s), \boldsymbol{p}_i(s))$ for $s \in S$.

Actions are assumed to affect the private and shared states independently, with an agent's private state being affected only by its own action. For each agent $i \in [n]$, we assume that there exists an action interpretation function τ_i such that for each action $a_i \in ACT_i$, the function $\tau_i(a_i)\colon S_i \to S_i$ is a private state transition function, representing the effect of the internal action on the agent's private states. For the shared state, we assume that there exists a function $\tau_0 : ACT \to (S_0 \to S_0)$ that captures the dependency of the environment's state transitions on the joint action performed by the agents. The state transition function for the environment is then defined by

$$\tau(\boldsymbol{a})((s_0, s_1, \ldots, s_n)) = (\tau_0(\boldsymbol{a})(s_0), \tau_1(\boldsymbol{a}_1)(s_1), \ldots, \tau_n(\boldsymbol{a}_n)(s_n)).$$

The protocol P_e of the environment may depend only upon the shared state. Formally, we assume that there exists a function $f\colon S_0 \to 2^{ACT_e}$ such that $P_e((s_0, s_1, \ldots, s_n)) = f(s_0)$. We allow the finite set of propositions *Prop* to describe any property of the states S, captured semantically by the valuation $\pi\colon S \to 2^{Prop}$.

2.4 Protocols and realizability

Assume we are given an environment E as above. A *protocol* for agent i is a function $P_i\colon \mathcal{O}^+ \to ACT_i$. A *joint protocol* is a family $\boldsymbol{P} = \{P_i\}_{i\in[n]}$ where each P_i is a protocol for agent i. Given such a protocol and a run r in the environment, we say r is *consistent* with the protocol if for every m there exists

$a \in ACT_e$ such that $r(m+1) = \tau(a, P_1(O_1(r,m)), \ldots, P_n(O_n(r,m)))(r(m))$. The interpreted system which is obtained from $\mathcal{I}(E)$ by restricting its runs to runs consistent with $\boldsymbol{P}$ is denoted $\mathcal{I}(E, \boldsymbol{P})$.

We say a formula φ is *realizable* in an environment E if there exists a joint protocol $\boldsymbol{P}$ such that $\mathcal{I}(E, \boldsymbol{P}) \models \varphi$.

2.5 A security example

Realizability of specifications in the logic of linear time and knowledge may be used to express a type of information flow property similar to those studied in the computer security literature. Consider a system with two agents High and Low, subject to a security policy that permits High to observe any information belonging to Low, but does not permit any information known only to High to flow to Low. If the system has been designed in an insecure fashion, and contains a "covert channel" that enables unintended information flow, High and Low may be able to collude to ensure that Low comes to know some secret belonging to High. (Concretely, such collusion may come about if Low has managed to place a Trojan Horse program at High.) We show how to formulate a version of this question as a realizability problem.

Let E be an environment (with agents H (High) and L (Low)) describing the possible states of the system we wish to analyse for unintended information flows. We may capture the assumption that information is permitted to flow from Low to High by defining states the observation functions by $O_H(s) = (O_L(s), P_H(s))$, where $O_L(s)$ is Low's observation in s and $P_H(s)$ is additional private information observable to High but not to Low. Note that this makes the environment hierarchical with respect to the ordering H, L on the agents. Suppose that p is a proposition whose value depends only on $P_H(s)$, and is moreover unaffected by the agents' actions. Then we may phrase the question "Can High and Low collude to reliably pass the information p from High to Low?" as the problem of whether the formula $\mathsf{F}(\mathsf{K}_L(p) \vee \mathsf{K}_L(\neg p))$ is realizable.

This question is closely related to, but somewhat stronger than, the notion of "(non)deducibility on strategies" of Wittbold and Johnson [Wit90]. It can be shown that deducibility on strategies corresponds to the formula $\mathsf{F}(\mathsf{K}_L(p) \vee \mathsf{K}_L(\neg p))$ being true on *some* run, rather than *all* runs, as required by our definition of realizability, with Low acting passively rather than having a choice of protocol. Realizability of the branching time formula $\mathsf{EF}(\mathsf{K}_L(p) \vee \mathsf{K}_L(\neg p))$, (where the path quantifier $E\varphi$ means that φ is true on some computation path) would correspond more directly to deducibility on strategies. Nevertheless, realizability of $\mathsf{F}(\mathsf{K}_L(p) \vee \mathsf{K}_L(\neg p))$ does seem to correspond to an interesting and intuitive security notion, which we call *strong*

deducibility on strategies in the sequel.

2.6 Main results

We now state the main results of the paper. As discussed above, van der Meyden and Vardi [MV98] showed that realizability of specifications in the logic of linear time and knowledge is decidable in the case of a single agent, but noted that a direct generalization of this result to multi-agent systems does not hold because of the undecidability [PR90] of realizability for linear-time logic specifications (without knowledge operators). They conjectured that the restriction to hierarchical and broadcast systems might provide decidable cases of realizability for specifications in the logic of linear time and knowledge.

The definition of hierarchical systems resembles, but differs from in some key respects, the class of pipelines shown by Pnueli and Rosner to yield a decidable case of realizability for linear-time logic specifications in multi-agent systems. It is therefore reasonable to first ask if realizability of linear-time logic specifications is decidable in hierarchical systems. We first show that this is indeed the case.

Theorem 1. *The synthesis problem for distributed systems with respect to specifications in the logic of linear time is decidable in hierarchical systems.*

The proof, which closely follows arguments of Pnueli and Rosner [PR90] for a similar result on pipeline architectures, is given in Section 4.1. However, this result does not generalize to the richer language of linear time and knowledge, even with respect to a quite specific class of hierarchical systems. Say that agent i is *omniscient* if for all states s, we have $O_i(s) = s$, i.e., the complete state is observable to the agent. Say that agent i is *blind* if for all states s, we have $O_i(s) = \bot$, for some fixed value $\bot$. Clearly, a system with only omniscient and blind agents is hierarchical. The proof of the following result is given in Section 4.2.

Theorem 2. *The synthesis problem for distributed systems with respect to specifications in the logic of linear time and knowledge is undecidable in a system with two agents, the first being omniscient, the second being blind. It remains undecidable for the case where the protocol of the blind agent is fixed (so only the protocol for the omniscient agent needs to be synthesized) and the specification does not contain the omniscient agent's knowledge operator (but does contain the blind agent's knowledge operator).*

For broadcast systems, however, it turns out that van der Meyden and Vardi's conjecture is true. The following result, proved in Section 3, is by a reduction to the case of a single agent.

Theorem 3. *The synthesis problem for specifications in the logic of linear time and knowledge is decidable in broadcast environments.*

Moreover, we can obtain some additional decidable cases in multi-agent systems by restricting the syntax of specifications. We say a formula φ in the language of linear time and knowledge is *positive* if every occurrence of any knowledge modality K_i is positive, that is, under an even number of negations. More precisely, a formula is positive if it can be built from p and $\neg p$ for $p \in P$, using $\vee$, $\wedge$, U, R, X, and K_i for $i \in [n]$.

Theorem 4. *For positive specifications φ in the logic of linear time and knowledge, and environments E, there exists an effective construction of a formula φ' and an environment E' such that φ is realizable in E iff φ' is realizable in E'. If E is hierarchical then so is E'.*

This reduction enables results concerning decidable cases of realizability for linear-time logic specifications to be lifted to a certain class of specifications involving knowledge operators. We provide the proof in Section 5.

Putting together Theorem 1 and Theorem 4, and noting that $\mathsf{F}(\mathsf{K}_L(p) \vee \mathsf{K}_L(\neg p))$ is a positive formula, we obtain the following result concerning the notion from computer security of Section 2.5.

Theorem 5. *Strong deducibility on strategies is decidable in hierarchical environments.*

3 Broadcast environments

In this section we prove Theorem 3, by means of a reduction to the synthesis problem for specifications in the logic of linear time and knowledge in single agent environments, which is decidable by the results of [MV98].

The following notion is useful for the proof that the reduction works. Define an *isomorphism of interpreted systems* $\mathcal{I} = (\mathcal{R}, \{\sim_i\}_{i\in[n]}, \pi)$ and $\mathcal{I}' = (\mathcal{R}', \{\sim'_i\}_{i\in[n]}, \pi')$ to be a bijection f between the runs of $\mathcal{I}$ and the runs of $\mathcal{I}'$, such that

1. $\pi(r,k) = \pi'(f(r),k)$ for all $r \in \mathcal{R}$, and $k \in \mathbf{N}$ and
2. $(r,n) \sim_i (r',n')$ iff $(f(r),n) \sim'_i (f(r'),n')$, for all runs $r, r \in \mathcal{R}$, $k, k' \in \mathbf{N}$ and agents i.

The following can be shown by a straightforward induction on the construction of the formula.

Lemma 1. *If f is an isomorphism of interpreted systems $\mathcal{I}$ and $\mathcal{I}'$ then for all formulas φ of the logic of linear time and knowledge, runs r of $\mathcal{I}$ and times k, we have $\mathcal{I}, (r, k) \models \varphi$ iff $\mathcal{I}, (f(r), k) \models \varphi$.*

For the proof of Theorem 3, we break the reduction into two stages. Intuitively, since we synthesize a deterministic protocol for each agent and transitions of agents' private state depend only on their choices of action, we can always derive an agent's private state from its initial private state and the sequence of observations it has made of the shared state. This is formalised in the following construction.

Given a broadcast environment $E = (S, I, P_e, \tau, \{O_i\}_{i\in[n]}, \mathit{Prop}, \pi)$, define the environment $E' = (S', I', P'_e, \tau', \{O'_i\}_{i\in[n]}, \mathit{Prop}, \pi')$ over the same signature and with the same set of propositions, as follows. The set of states of E' is given by adding to the states of E a bit that records whether the state is initial: $S' = \{0, 1\} \times S$, and the initial states of E' are exactly the initial states of E, but with the bit set to 0 to indicate that the state is initial, i.e., $I' = \{0\} \times I$. The transition function is accordingly defined so that $\tau'(\boldsymbol{a})((x, s)) = (1, \tau(\boldsymbol{a})(s))$ for all states (x, s) of E', encoding that a state obtained immediately after a transition cannot be initial. The interpretation of propositions ignores the new bit: $\pi'((x, s)) = \pi(s)$ for all states $s \in S$ and $x \in \{0, 1\}$. Similarly, for the protocol of the environment, $P'_e((x, s)) = P_e(s)$, ignoring the new bit.

The crux of the construction is that we treat observations in initial and subsequent states differently. In E, observations are given by $O_i(s) = (O_c(s), \boldsymbol{p}_i(s))$, where O_c is the observation function on shared states. In E', we take $O'_i((0, s)) = (O_c(s), \boldsymbol{p}_i(s))$ as in E, but take $O_i((1, s)) = O_c(s)$. That is, we suppress all but the initial observation of the private state.

The fact that we have defined protocols to be deterministic (and that the semantics for knowledge assumes implicitly that the protocol being executed is common knowledge) plays a critical role in the following result.

Lemma 2. *Let φ be a formula of the logic of linear time and knowledge. Then φ is realizable in E iff φ is realizable in E'.*

Proof. Let $\mathcal{O}_c$ and $\mathcal{O}_i$ be the ranges of O_c and O_i, respectively. If $\sigma = (o^c_0, s_0)(o^c_1, s_1) \dots (o^c_k, s_k)$ is a sequence in $\mathcal{O}_i^*$, define $\sigma\downarrow$ to be the sequence $(o^c_0, s_0)o^c_1 \dots o^c_k$ in $\mathcal{O}_i\mathcal{O}_c^*$.

Conversely, suppose $\boldsymbol{P}$ is a joint protocol for environment E. Given a sequence $\sigma = (o^c_0, s_0)o^c_1 \dots o^c_k$ in $\mathcal{O}_i\mathcal{O}_c^*$, define the sequence $\sigma \uparrow \boldsymbol{P}_i = (o^c_0, s_0)(o^c_1, s_1) \dots (o^c_k, s_k)$ inductively by

$$s_{j+1} = \tau_i[\boldsymbol{P}_i((o^c_0, s_0) \dots (o^c_j, s_j))](s_j).$$

Note that if r is a run of $\boldsymbol{P}$ in E then for all $k \in \mathbf{N}$ we have $(O_i(r,k)\downarrow)\uparrow\boldsymbol{P}_i = O_i(r,k)$.

Suppose that the joint protocol $\boldsymbol{P}$ realizes φ in E. Define the joint protocol $\boldsymbol{P}'$ for E' by $\boldsymbol{P}'_i(\sigma) = \boldsymbol{P}_i(\sigma\uparrow\boldsymbol{P}_i)$ for each agent i and $\sigma \in \mathcal{O}_i\mathcal{O}_c^*$. We claim that $\boldsymbol{P}'$ realizes φ in E'. For, given a run $r = s_0, s_1, s_2, \ldots$ of $\boldsymbol{P}$ in E define $f(r) = (0, s_0), (1, s_1), (1, s_2), \ldots$. This can be seen to be a run of $\boldsymbol{P}'$ in E'. In fact, using the conclusion of the previous paragraph we can easily see that the function f is an isomorphism of $\mathcal{I}(E, \boldsymbol{P})$ and $\mathcal{I}(E', \boldsymbol{P}')$. The claim then follows using Lemma 1.

Conversely, suppose that the joint protocol $\boldsymbol{P}'$ realizes φ in E'. Define the joint protocol $\boldsymbol{P}$ for E by $\boldsymbol{P}_i(\sigma) = \boldsymbol{P}'_i(\sigma\downarrow)$. The function f from the previous paragraph can again be shown to be an isomorphism, and it follows that $\boldsymbol{P}$ realizes φ in E. □

The transformation from E to E' shows that (with respect to each deterministic joint protocol) an agent's knowledge is completely determined by a sequence of the form $O_i(s_0)O_c(s_1)\ldots O_c(s_k)$, where $s_0, \ldots, s_k$ is a sequence of states of E. This suggests that it may be possible to treat O_c as the observation function of a single agent, and reduce the synthesis problem to a problem of synthesis for this agent.

One obstacle to this is that the remaining initial observations $O_i(s)$, which contain a private state for agent i, mean that the choice of action of distinct agents still depends on more than the sequence of common observations. This can be handled by means of a further transformation of E' into an environment with a modified signature.

Let $I_i = \{s_i \mid (s_0, s_1, \ldots, s_n) \in I\}$ be the set of possible initial private states of agent i in E. We define an environment

$$E^c = (S^c, I^c, P_e^c, \tau^c, O_c^c, Prop^c, \pi^c)$$

with a single agent (that we call c) and signature such that the environment has the same set of actions ACT_e as in E, and agent c has actions $(I_1 \to ACT_1) \times \ldots \times (I_n \to ACT_n)$.

We take the states of E^c to be the set $S^c = I \times S'$. The set of initial states is given by $I^c = \{(s, (0, s)) \mid s \in I\}$. Intuitively, the first component in these states keeps a memory of the initial state of E at the start of the run, the second component is the "current state" of E'. This is reflected in the definition of the transition functions. A joint action $\boldsymbol{a}^c$ of E^c is comprised of an action a_e of the environment and an action $(\alpha_1, \ldots, \alpha_n)$ of agent c, where each α_i is a function from I_i to ACT_i. This function may be applied to the private state $\boldsymbol{p}_i(s)$ of agent i in a state s of E, so that $\boldsymbol{a}^c(s) =$

$(a_e, \alpha_1(\boldsymbol{p}_1(s)), \ldots, \alpha_n(\boldsymbol{p}_n(s)))$ is a joint action of E, hence of E'. Thus, we may define

$$\tau^c(\boldsymbol{a}^c)((s,t)) = (s, \tau'(\boldsymbol{a}^c(s))(t)).$$

The protocol of the environment in E^c we take to be $P^c_e((s,t)) = P'_e(t)$. We enrich the set of propositions, defining

$$Prop^c = Prop \cup \{p_{i,x} \mid i \in [n],\ x \in I_i\}.$$

Propositions $p \in Prop$ are interpreted according to the "current state", i.e., $p \in \pi^c((s,t))$ iff $p \in \pi'(t)$. The new propositions refer to the initial private states: $p_{i,x} \in \pi^c((s,t))$ iff $x = \boldsymbol{p}_i(s)$, i.e., x is the private state of agent i in s. Finally, the observations of the single agent c are given by $O^c_c((s,(x,t))) = O_c(t)$.

Given a formula φ of the logic of linear time and knowledge, define φ^c to be the formula obtained, recursively, by replacing each subformula of the form $K_i\psi$ by

$$\bigwedge_{x \in I_i} (p_{i,x} \rightarrow K_c(p_{i,x} \rightarrow \psi^c)).$$

Then we can prove the following.

Lemma 3. *If φ is a formula of the logic of linear time and knowledge, then φ is realizable in E' iff φ^c is realizable in E^c.*

Proof. First note that we can convert a system $\mathcal{I}(E^c, P^c)$ to a system $\mathcal{I}_n(E^c, P^c)$ for n agents by defining the equivalence relations $\sim^c_i$ on points in the usual way, using the functions O^c_i on points, defined by $O^c_i(r,k) = \boldsymbol{p}^c_i(r(0)) \cdot O^c_c(r,k)$, where $\boldsymbol{p}^c_i((s,t)) = \boldsymbol{p}_i(s)$. Let the function f from sequences of states of E^c to sequences of states of E be defined by $f((s_0,t_0)(s_1,t_1)\ldots) = t_0, t_1, \ldots$. Given a protocol P^c for c in E^c, we can define a joint protocol $\boldsymbol{P'}$ for E', by

$$\boldsymbol{P'}_i((o^c_0, s^i_0)o^c_1 \ldots o^c_k) = (P^c(o^c_0 o^c_1 \ldots o^c_k))_i(s^i_0).$$

It can then be shown that f is an isomorphism between $\mathcal{I}_n(E^c, P^c)$ and $\mathcal{I}(E', \boldsymbol{P'})$. Conversely, given a joint protocol $\boldsymbol{P'}$ for E', define the protocol P^c for E^c by $P^c(o^c_0 \ldots o^c_k) = (\alpha_1, \ldots, \alpha_n)$ where $\alpha_i \colon I_i \rightarrow ACT_i$ is given by $\alpha_i(s^i) = \boldsymbol{P'}_i((o^c_0, s^i)o^c_1 \ldots o^c_k)$. Again, f is an isomorphism between $\mathcal{I}_n(E^c, P^c)$ and $\mathcal{I}(E', \boldsymbol{P'})$. In either case, it follows using Lemma 1 that $\mathcal{I}_n(E^c, P^c), (r,k) \models \varphi$ iff $\mathcal{I}(E', \boldsymbol{P'}), (f(r), k) \models \varphi$. By an induction on the construction of the formula ψ, we can show that $\mathcal{I}_n(E^c, P^c), (r,k) \models K_i\psi$ iff $\mathcal{I}(E^c, P^c), (r,k) \models \bigwedge_{x \in I_i}(p_{i,x} \rightarrow K_c(p_{i,x} \rightarrow \psi^c))$, which yields the result. □

Combining these two lemmas, the transformation from E to E' to E^c reduces the realizability problem for the multi-agent broadcast environment E to a problem of realizability in a single agent environment E^c, that is decidable by results of van der Meyden and Vardi [Mey96]. We note that this result plays essentially on the determinism of the evolution of the private states. If we were to add a source of non-determinism, e.g., independent inputs to the agents, then the realizability problem would be undecidable already for linear-time temporal logic formulas. This can be seen by noting that including in the formula a constraint that all broadcasts are trivial reduces this to the case of completely independent agents shown to be undecidable by Pnueli and Rosner [PR90].

4 Hierarchical environments

In this section we first show that for our knowledge-based definition of hierarchical environments, the synthesis problem for linear-time temporal formulas is decidable. Then we show that the synthesis problem is undecidable for the logic of linear time and knowledge in hierarchical environments.

4.1 Temporal formulas

In this section we establish the decidability of realizability of linear-time logic formulas in hierarchical environments. We recall that our definition of hierarchical environment differs from the Pnueli–Rosner [PR90] definitions of "hierarchical architecture" and their special case of "pipeline architecture", though it somewhat resembles the latter. However, our proof uses ideas similar to those they use to show decidability of realizability of temporal formulas in pipeline architectures.

We prove two lemmas from which the claim follows by induction on the number of agents. Recall that a protocol for agent i with observation space O_i is a function in $O_i^* \to ACT_i$, which can be viewed as an infinite tree. We say it is a (O_i, ACT_i)-tree. In general, a (M, N)-tree is a tree with M-branching and N-node labels, that is, it is a function in $M^* \to N$.

Since the environment is hierarchical, for each pair of agents $i < j$, there exists a mapping $h_{i,j} : O_i \to O_j$ such that for all states s of the environment we have $O_j(s) = h_{i,j}(O_j(s))$. These mappings have the property that $h_{j,k} \circ h_{i,j} = h_{i,k}$.

The induction base can be proved using the following lemma.

Lemma 4. *Given an environment E and a temporal specification φ, we can construct a finite tree automaton A for $(O_1, ACT_1 \times \cdots \times ACT_n)$-trees that*

accepts a tree t iff t realizes φ in the modified environment E' where agent 1 takes over the rôles of agents 2 through n (and plays his own part).

The induction step can be proved using the following lemma.

Lemma 5. *Given an environment E with just two agents and a tree automaton A for $(O_1, ACT_1 \times ACT_2)$-trees, we can construct a finite tree automaton A' for (O_2, ACT_2)-trees such that A' accepts a tree t_2 iff there exists a protocol t_1 for agent 1 such that the joint protocol $(t_1, t_2 \circ h_{1,2})$ (viewed as a protocol for agent 1) is accepted by A.*

Let's see that these two lemmas let us prove the general claim. To this end, let us assume we are given a hierarchical environment E with n agents. Let A be the automaton promised by Lemma 4. For $i = 1, \ldots, n$ we show that there exists a finite tree automaton A_i for $(O_i, ACT_i \times \cdots \times ACT_n)$-trees that accepts a tree t iff there exist protocols $t_j \colon O_j^* \to ACT_j$ for $j = 1, \ldots, i-1$ such that A accepts the tree $(t_1, \ldots, t_{i-1} \circ h_{1,i-1}, t \circ h_{1,i})$. The proof is by an induction on i. Considering $i = n$, and applying Lemma 4, we have that A_n is empty iff there exists protocols $t_1, \ldots, t_n$ such that A accepts $(t_1, \ldots, t_n \circ h_{1,n})$, which is the case iff φ is realizable in E.

The induction base is trivial, for it follows immediately from Lemma 4. For the induction step, assume the claim holds for $i < n$ and let A_i be the automaton which we already know exists. This automaton can be viewed as a finite tree automaton for $(O_i, ACT_i \times (ACT_{i+1} \times \cdots \times ACT_n))$-trees. Applying Lemma 5 yields a finite tree automaton for $(O_{i+1}, ACT_{i+1} \times \cdots \times ACT_n)$-trees which has the desired properties.

So let's turn to the proofs of the two lemmas, where we use some facts from automata and game theory. In particular, we will use the forgetful determinacy theorem, which we explain in what follows. For further background on these tools, the reader is referred to [Tho90, Tho97, GTW02].

A game is a triple $(P, P_0, P_1, E, c, W, p_I)$ where P is a set of positions, $\{P_0, P_1\}$ is a partition of P into the positions of the two players, Zero and One, $E \subseteq P \times P$ is the set of moves, $c \colon P \to A$ is a function into a finite alphabet, which is extended to a function $c \colon P^\omega \to A^\omega$ in the canonical way, $W \subseteq A^\omega$ is an ω-regular set, the winning set for Player 0, and p_I is the initial position. A play is a maximal path π in (P, E) starting in p_I. It is winning for Player 0 if $c(\pi)$ belongs to W or it is finite and ends in a position from P_1 (early loss, Player 1 can't move any more). A strategy for Player 0 is a partial function $\sigma \colon P^* P_0 \to P$ such that whenever $\sigma(\pi p)$ is defined, then $(p, \sigma(\pi p)) \in E$. A play π conforms with a strategy σ if for all $i < |\pi|$ with $\pi(i) \in P_0$, then $\pi(i+1) = \sigma(\pi[0, i])$. A strategy σ is winning for Player 0 (is a winning strategy for Player 0) if every play that conforms with σ is winning for Player 0. A

memory-based strategy is determined by a set M, an initial memory m_0, an update-function $f\colon M \times P \to M$, and a function $\sigma_0\colon P_0 \times M \to P$. The induced strategy σ is determined as follows. First, for every finite path through (P, E), the corresponding memory $m(\pi)$ is defined inductively by $m(\epsilon) = m_0$ and $m(\pi p) = f(m(\pi), p)$. Then, $\sigma(\pi) = \sigma_0(p, m(\pi))$. Now, the forgetful determinacy theorem states that if there exists a winning strategy for Player 0, then there exists a memory-based strategy where $|M|$ is smaller than a natural number $N(W)$ only depending on W (rather than on P or E). Moreover, if W is given by a Muller-automaton or a monadic second-order formula or something similar, then an upper bound $B(W)$ for $N(W)$ can be computed effectively. That is, one can assume that $M = [B(W)]$ and $m_0 = 1$.

Proof of Lemma 4. We use the fact that monadic second-order logic is equivalent to tree automata and observe that the following can be phrased in that logic:

> For every path of a $(O_1, ACT_1 \times \cdots \times ACT_n)$-tree, for each labeling of the path with actions of the environment, and for each labeling of the path with states from S, if these two labelings are consistent (the joint actions yield the states), then the given specification φ holds true on the path.

This yields the desired result.

Proof of Lemma 5. We use the idea sketched in [Ros92]. Without loss of generality we assume that A is a non-deterministic tree automaton with set of transitions Δ and states Q. A transition of A is a triple $(q, (\boldsymbol{a}_1, \boldsymbol{a}_2), s)$ where $q \in Q$, $(\boldsymbol{a}_1, \boldsymbol{a}_2) \in ACT_1 \times ACT_2$ and $s\colon O_1 \to Q$ determines the successor states. We write simply h for the function $h_{1,2}\colon O_1 \to O_2$.

Let t_2 be a (O_2, ACT_2)-tree. We construct an infinite game between two players, called *Challenger* (Player 1) and *Responder* (Player 0) which we claim has the property that Responder wins iff there exists a protocol for agent 1 as described in the lemma. This game will then be used to describe how A' can be constructed.

We first describe the positions and the moves of the game. There are three different types of positions. First, there are positions that are strings over O_2. The other positions connect these as follows. Suppose we are given a position $u \in O_2^*$. Then it is Responder's move. He gets to choose a transition $\delta = (q, (\boldsymbol{a}_1, \boldsymbol{a}_2), s)$ such that $t_2(u) = \boldsymbol{a}_2$. After that, it is Challenger's move. She gets to choose a direction $o_1 \in O_1$. Then the game deterministically moves

to $uh(o_1)$. Thus, intermediate nodes of the game are of the form (u, δ) and (u, δ, o_1). More formally, the set of positions of Responder is $P_R = O_2^*$, the set of positions of Challenger is $P_C = O_2^* \times \Delta$, and the set $P_D = O_2^* \times \Delta \times O_1$ is the set of deterministic positions. For simplicity, we assume that all positions from P_D belong to Challenger.

The rest of the game is as follows. The initial position is ϵ. The coloring function c is given by $c(u) = \$$, where $\$$ is a dummy symbol, $c((u, \delta)) = \delta$ and $c((u, \delta, o_1)) = o_1$. The winning condition W consists of all infinite sequences over the alphabet $\{\$, \Delta, O_1\}$ of the form $\$\delta_0 o_1^0 \$\delta_1 o_1^1 \$\delta_2 o_1^2 \$\delta_3 \dots$ such that the following conditions are satisfied.

1. For every i, if $\delta_i = (q, (\boldsymbol{a}_1, \boldsymbol{a}_2), s)$ and $\delta_{i+1} = (q', (\boldsymbol{a}_1', \boldsymbol{a}_2'), s')$, then $s(o_1^i) = q'$.

2. Suppose $\delta_i = (q_i, \dots)$ for every i, then $q_0 q_1 q_2 \dots$ is accepting for A.

Let's first check that the above claim is true. A strategy for Responder is a function that maps every element of $(P_R P_C P_D)^* P_R$, which is of the form

$$\epsilon(\epsilon, \delta_0)(\epsilon, \delta_0, o_1^0) o_2^0 (o_2^0, \delta_1)(o_2^0, \delta_1, o_1^1) o_2^0 o_2^1 (o_2^0 o_2^1, \delta_2)(o_2^0 o_2^1, \delta_2, o_1^2) o_2^0 o_2^1 o_2^2 \dots o_2^0 o_2^1 \dots o_2^r \ ,$$

to a position of the form

$$(o_2^0 o_2^1 \dots o_2^r, \delta) \ . \tag{3}$$

Such a strategy can immediately be translated into a protocol tree t_1 for agent 1 and an accepting run of A on the tree obtained from joining t_1 and t_2. Conversely, a winning strategy for Responder can be constructed from a protocol for agent 1 and a suitable accepting run of A.

By the forgetful determinacy theorem, if there exists a winning strategy for Responder, then with $k = B(W)$, a winning strategy is induced by the set $[k]$, an initial memory 1, an update-function $f\colon [k] \times (P_R \cup P_C \cup P_D) \to [k]$, and a function $\sigma_0\colon P_R \times [k] \to P_C$.

Now, observe that f and σ_0 can be viewed as functions with domain O_2^* and a finite range:

- σ_0 can be viewed as a function $\sigma_0'\colon O_2^* \to \Delta^{[k]}$;
- f can be viewed as a function $f'\colon O_2^* \to [k]^{[k] \times ([1] \cup \Delta \cup (\Delta \times O_1))}$.

Now, observe that the following property of an (O_2, ACT_2)-tree can be expressed in monadic second-order logic:

There exists two labelings with functions $[k] \to \Delta$ and $[k] \times ([1] \cup \Delta \cup (\Delta \times O_1)) \to [k]$ such that the following holds true. The strategy σ, which is determined by the two labelings when interpreted as σ'_0 and f' as explained above is winning for Responder.

We can now use the same argument as above to conclude that the desired automaton exists.□

This completes the proof of Theorem 1.

4.2 Temporal *and* knowledge formulas

For the proof of Theorem 2, we need background on lossy counter machines, which is provided in the first subsection. The second subsection contains the actual undecidability proof.

4.2.1 Lossy counter machines

A *counter machine* is a tuple

$$L = (Q, k, q_I, \Delta) \tag{4}$$

where

- Q is a finite set of *states,*
- k is a natural number, the number of *registers* of the machine,
- $q_I \in Q$ is the *initial state* of L, and
- $\Delta \subseteq (Q \times \{0, \ldots, k-1\} \times Q) \cup (Q \times \{0, \ldots, k-1\} \times Q \times Q)$ is the set of *commands.*

A *configuration* of such a counter machine is a tuple $(q, r_0, \ldots, r_{k-1})$ where q is a state and $r_j \in \mathbf{N}$ for all $j < k$, representing the values of the registers.

The *lossy* semantics of such a machine is determined by two binary relations on configurations. First, the *machine relation,* denoted $\to^m$, is defined as follows. We have $(q, r_0, \ldots, r_{k-1}) \to^m (q', r'_0, \ldots, r'_{k-1})$ if

- $(q, j, q') \in \Delta$, $r'_j = r_j + 1$ and $r'_{j'} = r_{j'}$ for all $j' \neq j$, or
- $(q, j, q_1, q_2) \in \Delta$, $r_j > 0$, $q' = q_1$, $r'_j = r_j - 1$, and $r'_{j'} = r_{j'}$ for all $j' \neq j$, or
- $(q, j, q_1, q_2) \in \Delta$, $r_j = 0$, $q' = q_2$ and $r'_{j'} = r_{j'}$ for all j'.

Second, the *lossiness relation* is the relation denoted $\rightarrow^l$ and defined by $(q, r_0, \ldots, r_{k-1}) \rightarrow^l (q', r'_0, \ldots, r'_{k-1})$ if $q = q'$ and $r'_j \leq r_j$ for all j.

The *single-step relation* of the machine is defined by $\Rightarrow = \rightarrow^l \circ \rightarrow^m \circ \rightarrow^l$.

A lossy run of counter machine is a sequence $s_0 \Rightarrow s_1 \Rightarrow s_2 \Rightarrow \ldots$ of configurations. It is called an *infinite run* if the sequence is infinite.

We will use the following theorem.

Theorem 6 ([May03]). *The following problem is undecidable. Given a counter-machine L, is there a number n such that there exists an infinite lossy run of L starting with the configuration $(q_I, 0, \ldots, 0, n)$.*

We modify this slightly for technical reasons. A *complete counter machine with forbidden state* is a tuple

$$L = (Q, k, q_I, q_f, \Delta) \tag{5}$$

where

- (Q, k, q_I, Δ) is a counter machine as above,
- $q_f \in Q$ is the forbidden state, and
- Δ has the additional property that $(q, 0, q_f) \in \Delta$ for every q.

With this definition, we have the following as an immediate consequence of the above theorem.

Corollary 1. *The following problem is undecidable. Given a counter-machine L, is there a number n such that there exists an infinite lossy run of L starting with the configuration $(q_I, 0, \ldots, 0, n)$ and never going through state q_f.*

For later use, we will give an alternative definition of $\Rightarrow$. To this end, we need three other relations, denoted $\rightarrow^{al}$, $\rightarrow^{am}$, $\rightarrow^{aal}$. First, we let $(q, r_0, \ldots, r_{k-1}) \rightarrow^{al} (\gamma, r'_0, \ldots, r'_{k-1})$ if γ is a command starting with q and $r'_j \leq r_j$ for all $j < k$.

Second, we let $(\gamma, r_0, \ldots, r_{k-1}) \rightarrow^{am} (\overline{q'}, r'_0, \ldots, r'_{k-1})$ if

- $\gamma = (q, j, q')$, $0 < r'_j \leq r_j + 1$ and $r'_{j'} \leq r_{j'}$ for all $j' \neq j$,
- $\gamma = (q, j, q_1, q_2)$, $r_j > 0$, $q' = q_1$, $r'_j \leq r_j - 1$, and $r'_{j'} \leq r_{j'}$ for all $j' \neq j$, or
- $\gamma = (q, j, q_1, q_2)$, $r_j = 0$, $q' = q_2$, and $r'_{j'} \leq r_{j'}$ for all j'.

Here, $\overline{q}$ for a state $q \in Q$ denotes a new copy of q. The set of all these copies is denoted $\overline{Q}$.

And, third, we let $(\overline{q}, r_0, \ldots, r_{k-1}) \rightarrow^{aal} (q', r'_0, \ldots, r'_{k-1})$ if $(q, r_0, \ldots, r_{k-1}) \rightarrow^{l} (q', r'_0, \ldots, r'_{k-1})$.

Now we can state the following.

Remark 1. *For every counter machine, we have*

$$\Rightarrow \; = \rightarrow^{al} \circ \rightarrow^{am} \circ \rightarrow^{aal} \; . \tag{6}$$

There is another relation we will also need in the proof to follow. This relation is denoted $\rightarrow^{am'}$. It is different from $\rightarrow^{am}$ in that the second and the third clause above are simply replaced by

- $\gamma = (q, j, q_1, q_2)$, $q' \in \{q_1, q_2\}$ and $r'_{j'} \leq r_{j'}$ for all $j' < k$.

In particular, we have $\rightarrow^{am} \subseteq \rightarrow^{am'}$.

We say $s_0, s_1, s_2, \ldots$ is a *refined computation* of the given machine if $s_0 \rightarrow^{al} s_1 \rightarrow^{am} s_2 \rightarrow^{aal} s_3 \rightarrow^{al} s_4 \rightarrow^{am} s_5 \rightarrow^{aal} s_6 \rightarrow^{al} \ldots$. We say it is a *refined weak computation* if $s_0 \rightarrow^{al} s_1 \rightarrow^{am'} s_2 \rightarrow^{aal} s_3 \rightarrow^{al} s_4 \rightarrow^{am'} s_5 \rightarrow^{aal} s_6 \rightarrow^{al} \ldots$

The important observations here are the following.

Remark 2. *A refined weak computation $s_0, s_1, s_2, \ldots$ is a refined computation iff for every m with $s_{3m+1} = ((q, j, q_1, q_2), r_0, \ldots, r_{k-1})$ and $s_{3m+2} = (\overline{q'}, r'_0, \ldots, r'_{k-1})$ we have $q' = q_2$ if $r_j = 0$ and else $q' = q_1$ and $r'_j < r_j$.*

Remark 3. *There is an infinite computation starting with $(q_I, 0, \ldots, 0, n)$ for some n iff there is an infinite refined weak computation $s_0, s_1, s_2, \ldots$ starting with $(q_I, 0, \ldots, 0, n)$ and satisfying the above requirement.*

4.2.2 The reduction

In this section, we prove Theorem 2 by reducing the problem from Corollary 1 to the realizability problem. This will yield the desired result.

Let L be a complete k-counter machine with forbidden state as above. We construct an environment E with two agents and a formula φ in the language of linear time and knowledge such that there exists a joint protocol $\boldsymbol{P}$ realizing φ in E iff there exists a natural number n such that there exists an infinite run of L starting with $(q_I, 0, \ldots, 0, n)$ and avoiding q_f.

We call the two agents by the names A and B. (Formally, A may just stand for 1 and B for 0.) Agent A will observe "all" that happens, whereas B will be "blind:" it observes nothing. More precisely, for all states s of E, the observation functions for the agents are defined by $O_A(s) = s$ and $O_B(s) = \perp$ for some fixed element $\perp$.

Basic idea of reduction. We construct the environment E in such a way that for each joint protocol $\boldsymbol{P}$ we can view $\mathcal{I}(E, \boldsymbol{P})$ as a refined weak computation of L. In addition, we will construct φ in such a way that $\mathcal{I}(E, \boldsymbol{P}) \models \varphi$ iff this sequence is a refined computation and never goes through q_f.

To be able to view $\mathcal{I}(E, \boldsymbol{P})$ as a refined computation of L, we simply consider, for each m, the set of all states a run of $\mathcal{I}(E, \boldsymbol{P})$ can be in. This gives us a multi-set of states of E in a natural way, and with each such multiset we will associate an element of a refined computation.

Our environment operates in two phases. In the first phase, the environment uses nondeterminism to generate the initial configuration of the machine, that is, the number n register $k-1$ gets assigned in the initial configuration. In the second phase, the refined computation of L is simulated.

The important point is the following. There needs to be some "coordination" between the individual runs of $\mathcal{I}(E, \boldsymbol{P})$ so as to ensure that, for instance, the decision to switch from the first to the second phase is made at the same time for all runs. This is where agent B comes into the picture. Since B is blind, if B decides to perform an action corresponding to switch from the first to the second phase, it will take this decision at the same point in time of all runs. Similarly, if we want to check that the value of a register is 0, this can be checked by the knowledge operator for agent B, which can quantify over all runs, because B is blind. Agent A is used for "individual" actions, in particular, to model lossiness. (There is an alternate way to satisfy the requirement that the switch to the second phase is made simultaneously in all runs, which is to let A guess the switch point and add to the specification a formula that uses B's knowledge operator to check that the switch is made at the same time in all runs. This yields the refined version of the theorem stating that the problem remains undecidable for the case where, given a fixed protocol for B, we synthesize a protocol for the single agent A, and the specification talks only about the knowledge of B. We leave the details of this variant to the reader.)

The actual number of runs, hence distinguishable points, may in general be infinite, but since choice of action is based only on the prefixes, at each moment of time, there will be a finite number of equivalence classes of points for agent A. We use the number of these distinct equivalence classes to represent the values of the registers. To distinguish different counters we use different states, that is, depending on the current state of a point of an equivalence class, this class will be counted for a certain register or not.

The state set and the formula. The environment has the following state set:

$$S = \{\mathit{spawn}, \mathit{stable}, \mathit{trash}, c_0, \ldots, c_{k-1}, d_0, \ldots, d_{k-1}\} \cup Q \cup \overline{Q} \cup \Delta \quad (7)$$

where $\overline{Q}$ is a disjoint copy of Q. The state *spawn* is the initial state. The states *spawn* and *stable* are the states of the *first phase* whereas the other states are the states of the second phase.

We also set $\pi_e(s) = s$ for every $s \in S$, that is, we have a proposition for each state expressing that the system is in that state.

Let $\boldsymbol{P}$ be any joint protocol and let $\mathcal{I}(E, \boldsymbol{P}) = (\mathcal{R}, \pi, \{\sim_i\}_{i\in[n]})$. For each m, let $\mathcal{R}_m$ be the set of prefixes of elements from $\mathcal{R}$ of length m and let M_m be the multi-set of all end states of the elements from $\mathcal{R}_m$.

Now, assume a sequence $s_0, s_1, s_2, \ldots$ is a refined weak computation starting with $s_0 = (q_I, 0, \ldots, 0, n)$. Then there will be a joint protocol $\boldsymbol{P}$ such that the following holds.

First, for every $m \leq n$, we will have:

$$M_m(\mathit{spawn}) = 1 \ , \quad (8)$$
$$M_m(\mathit{stable}) = m \ , \quad (9)$$
$$M_m(s) = 0 \text{ for any other state.} \quad (10)$$

And for all $m \geq 0$ with $s_m = (\alpha, r_0, \ldots, r_{k-1})$, we will have:

$$M_{n+m}(\mathit{spawn}) = 0 \ , \quad (11)$$
$$M_{n+m}(\mathit{stable}) = 0 \ , \quad (12)$$
$$M_{n+m}(\alpha) = 1 \ , \quad (13)$$
$$M_{n+m}(\alpha) = 0 \text{ for any other } \alpha \in Q \cup \Delta \cup \overline{Q}, \quad (14)$$
$$M_{n+m}(c_j) = r_i \ . \quad (15)$$

That means, in particular, that the number of occurrences of state c_j corresponds exactly to the value of register j. In the above, we haven't said anything about the states d_j and the state *trash*. The latter is simply used as a dummy state if we want to discontinue a run. The former are used as indicator variables to have some control over the lossiness. In our construction, if we want to decrement a counter we could simply switch from c_i to *trash*. What we do is to transition from c_i to d_i and then to *trash*; as a consequence we can specify in a formula that a decrement for register i has occurred.

For the above claim, the converse will also hold true. That is, for every system induced by a joint protocol, there will be a sequence as above satisfying the specified conditions, with one exception. There will also be a system

where, for every m,

$$M_m(spawn) = 1 \ , \tag{16}$$

$$M_m(stable) = m \ , \tag{17}$$

$$M_m(s) = 0 \text{ for any other state.} \tag{18}$$

That is, the second phase will never be started in that system.

Since, by (12) and (13) the environment is constructed so that at each moment of time at most one element α of $Q \cup \Delta \cup \overline{Q}$ can occur, and the agent B has synchronous perfect recall but is blind, the formula $\mathsf{L}_B\alpha$ says that α is the unique such element occurring at the current time. Similarly, $\mathsf{L}_B c_i$ says that there is at least one prefix of length equal to the current time ending in state c_i.

Taking all this into account, in order to make the reduction work, we will only have to choose our formula φ to be the conjunction of the following formulas:

$$\mathsf{FL}_B q_I \ , \tag{19}$$

which rules out the last system, which never gets to the second phase;

$$\bigwedge_{(q,j,q_1,q_2)\in\Delta} \mathsf{G}((q,j,q_1,q_2) \rightarrow ((\mathsf{K}_B\neg c_i \rightarrow \mathsf{XL}_B\overline{q_2}) \wedge (\mathsf{L}_B c_j \rightarrow \mathsf{X}(\mathsf{L}_B\overline{q_1} \wedge \mathsf{L}_B d_j))) \ , \tag{20}$$

which takes care of the condition from Remark 2, and

$$\bigwedge \mathsf{GK}_B\neg q_f \ , \tag{21}$$

which makes sure we get a run avoiding q_f. Note that this formula refers only to agent B's knowledge: we do not need the knowledge modality for agent A.

What remains to be specified are the actions and the transitions. This can be dealt with easily.

Actions and transitions. We have

$$ACT_A = \{\textsc{Keep}, \textsc{Zero}\} \cup \{\langle\gamma\rangle \mid \gamma \in \Delta\} \ , \tag{22}$$

$$ACT_B = \{\textsc{Skip}, \textsc{Start}\} \ . \tag{23}$$

For the environment, we have an action $\langle s\rangle$ for every state s except for *spawn* and states of the form (q,i,q'). For these states, the environment has two actions each, $\langle spawn\rangle_b$ with $b < 2$ and $\textsc{Inc}_b$ with $b < 2$, respectively. All actions of the environment are enabled in the respective states.

Let $\boldsymbol{a}$ be a joint action. We describe how it operates on the states.

State *spawn*. First case, $\boldsymbol{a}_e = \langle spawn \rangle_0$. If $\boldsymbol{a}_B = \textsc{Skip}$, then $\tau(\boldsymbol{a})(spawn) = spawn$, and if $\boldsymbol{a}_B = \textsc{Start}$, then $\tau(\boldsymbol{a})(spawn) = q_I$.

Second case, $\boldsymbol{a}_e = \langle spawn \rangle_1$. If $\boldsymbol{a}_B = \textsc{Skip}$, then $\tau(\boldsymbol{a})(spawn) = spawn$, and if $\boldsymbol{a}_B = \textsc{Start}$, then $\tau(\boldsymbol{a})(spawn) = trash$.

This makes sure that *spawn* splits in *spawn* and *stable* unless agent B performs Start. And B performing Start results in *spawn* transitioning to q_I, the initial state.

State *stable*. If $\boldsymbol{a}_B = \textsc{Skip}$, then $\tau(\boldsymbol{a})(stable) = stable$, else $\tau(\boldsymbol{a})(stable) = c_{k-1}$.

This makes sure all states *stable* are transformed into the initial counter value for register $k-1$.

State c_i. If $\boldsymbol{a}_A = \textsc{Keep}$, then $\tau(\boldsymbol{a})(c_i) = c_i$, else $\tau(\boldsymbol{a})(c_i) = d_i$.

If agent A performs anything other than Keep then the counter is decremented.

State d_i. $\tau(\boldsymbol{a})(d_i) = trash$.

State *trash*. $\tau(\boldsymbol{a})(trash) = trash$.

State q. If $\boldsymbol{a}_A$ is of the form $\langle \gamma \rangle$ for some $\gamma \in \Delta$ and γ starts with q, then $\tau(\boldsymbol{a})(q) = \gamma$ and else $\tau(\boldsymbol{a})(q) = (q, 0, q_f)$.

Agent A can choose which transition he wants to take; actions that don't correspond to an enabled transition result in choosing the dummy transition $(q, 0, q_f)$.

State (q, j, q'). If $\boldsymbol{a}_e = \textsc{Inc}_0$, then $\tau(\boldsymbol{a})((q, j, q')) = \overline{q'}$ and else $\tau(\boldsymbol{a})((q, j, q')) = c_j$.

The non-determinism for the environment makes sure we actually increase the counter (while there can also be lossy transitions).

State (q, j, q', q''). If $\boldsymbol{a}_A = \textsc{Zero}$, then $\tau(\boldsymbol{a})((q, j, q_1, q_2)) = \overline{q_2}$ and else $\tau(\boldsymbol{a})((q, j, q_1, q_2)) = \overline{q_1}$.

Agent A gets to guess whether the counter is zero or not; his guess will be verified by the formula φ (see above).

5 A reduction for positive formulas

We now define the reduction promised in Theorem 4, and prove its correctness.

5.1 The reduction

Let

$$E = (S, I, P_e, \tau, \{O_i\}_{i\in[n]}, P, \pi_e) \tag{24}$$

be an environment for n agents and φ a positive formula. We construct an environment E' and a formula φ' in the language of linear time (without knowledge operators) such that the following statements are equivalent:

(A) φ is realizable in E.

(B) φ' is realizable in E'.

Moreover, we will be able to derive a protocol realizing φ in E from a protocol realizing φ' in E' in a straightforward way (just by forgetting).

The idea is that in the new system in each state each agent has to say which of his knowledge subformulas he thinks are true by choosing a corresponding action. That the agent's choices are indeed correct will then be verified by modifying φ appropriately.

To describe E' we need some more notation. For every $i \in [n]$ we let Φ_i be the set of subformulas of φ of the form $\mathsf{K}_i\psi$ and Φ^* the union of all these sets.

We first describe the sets of actions for E', which are denoted by ACT'_e, ACT'_1, ..., ACT'_n. We set $ACT'_e = ACT_e$ and $ACT'_i = ACT_i \times 2^{\Phi_i}$ for every $i \in [n]$.

The system E' is given by

$$E' = (S \times 2^{\Phi^*}, I', P'_e, \tau', \{O'_i\}_{i\in[n]}, P', \pi'_e) \tag{25}$$

where the individual components are defined by

$$I' = \{(s_I, \emptyset) \mid s_I \in I\} \ , \tag{26}$$

$$P' = P \cup \{p_\psi \mid \psi \in \Phi^*\} \ , \tag{27}$$

$$O'_i((s, \Psi)) = O_i(s) \ , \tag{28}$$

$$\pi'_e((s, \Psi)) = \pi_e(s) \cup \{p_\psi \mid \psi \in \Psi \ , \tag{29}$$

and

$$\tau'(a_e,(a_1,\Psi_1),\ldots,(a_n,\Psi_n)) = (\tau(a,a_1,\ldots,a_n),\Psi_1\cup\cdots\cup\Psi_n)\ . \qquad (30)$$

To obtain the new specification φ', we proceed as follows. First, we define the *flattened variant* of a formula ψ, denoted $\overline{\psi}$. The formula $\overline{\psi}$ is the temporal formula obtained from ψ by substituting every maximal knowledge subformula ψ' by $\mathsf{X}p_{\psi'}$. For example, if ψ is $\mathsf{K}_1\mathsf{K}_2p \rightarrow \mathsf{G}\mathsf{K}_1q$ then $\overline{\psi}$ is $(\mathsf{X}p_{\mathsf{K}_1\mathsf{K}_2p}) \rightarrow \mathsf{G}\mathsf{X}p_{\mathsf{K}_1q}$.

Now, the new specification simply says that whenever agent i claims $\mathsf{K}_i\psi$ is true it is, in fact, true and uses flattened variants of the knowledge formulas:

$$\varphi' = \overline{\varphi} \wedge \bigwedge_{i\in[n],\mathsf{K}_i\psi\in\Phi_i} \mathsf{G}(\mathsf{X}p_{\mathsf{K}_i\psi} \rightarrow \overline{\psi})\ . \qquad (31)$$

We will also write χ for the big conjunction on the right-hand side.

The use of the X-operator is due to the fact that the claims about valid subformulas an agent makes by carrying out an action are only available in the environment in the next state.

5.2 The correctness proof

The correctness of the above construction is proved in two steps.

5.2.1 From E to E'

We first show that (A) from above implies (B). To this end, assume we are given a joint protocol $\boldsymbol{P}$ that realizes φ in E. If u is a sequence then we write $|u|$ for the length of u minus 1, so that $|O_i((r,n))| = n$.

Let $u \in O^+$ and $i \in [n]$. If there is no run r in $\mathcal{I}(E,\boldsymbol{P})$ such that $O_i((r,|u|)) = u$, then we set $P'_i(u)$ to an arbitrary action (because it won't matter). If there is a run r in $\mathcal{I}(E,\boldsymbol{P})$ such that $O_i((r,|u|)) = u$, let r be such a run. We set $P'_i(u) = (P_i(u),\Sigma_i)$ where $\Sigma_i = \{\psi \in \Phi_i \mid \mathcal{I}(E,\boldsymbol{P}),(r,|u|) \models \psi\}$. Observe that this is independent of the particular r chosen (since for all runs satisfying this condition the corresponding points are indistinguishable by agent i).

Now note the following. For every run $r = s_0s_1\ldots$ of $\mathcal{I}(E,\boldsymbol{P})$, the sequence $r' = (s_0,\Psi_0)(s_1,\Psi_1)\ldots$ with $\Psi_0 = \emptyset$ and $\Psi_{m+1} = \{\psi \in \Phi \mid \mathcal{I}(E,\boldsymbol{P}),(r,m) \models \psi\}$ for every m is a run in $\mathcal{I}(E',\boldsymbol{P}')$. And every run of $\mathcal{I}(E',\boldsymbol{P}')$ is obtained in this way. Moreover, from the definition of the observation functions O'_i, we have $O_i(r,m) = O'_i(r',m)$. It follows that that for every formula ψ over P only, each run r, and each m, we have

$\mathcal{I}(E, \boldsymbol{P}), (r, m) \models \psi$ iff $\mathcal{I}(E', \boldsymbol{P'}), (r', m) \models \psi$. This, in turn, implies that for each $\psi \in \Phi^*$, each run r, and each m, we have $\mathcal{I}(E, \boldsymbol{P}), (r, m) \models \psi$ iff $\psi \in \Psi_{m+1}$ iff $\mathcal{I}(E', \boldsymbol{P'}), (r', m) \models \mathsf{X}p_\psi$. By a straightforward induction it follows that $\mathcal{I}(E, \boldsymbol{P}), (r, m) \models \psi$ iff $\mathcal{I}(E', \boldsymbol{P'}), (r', m) \models \overline{\psi}$, for all subformulas ψ of φ. In particular, we obtain $\mathcal{I}(E', \boldsymbol{P'}), (r', 0) \models \overline{\varphi}$, since $\mathcal{I}(E, \boldsymbol{P}), (r, 0) \models \varphi$. Moreover, if $\mathcal{I}(E', \boldsymbol{P'}), (r', m) \models \mathsf{X}p_{\mathsf{K}_i\psi}$ then $\mathcal{I}(E, \boldsymbol{P}), (r, m) \models \mathsf{K}_i\psi$ hence $\mathcal{I}(E, \boldsymbol{P}), (r, m) \models \psi$ and $\mathcal{I}(E', \boldsymbol{P'}), (r', m) \models \overline{\psi}$. This shows that $\mathcal{I}(E', \boldsymbol{P'}) \models \chi$. This shows that $\boldsymbol{P'}$ realizes φ' in E'.

5.2.2 From E' to E

For the converse, we assume we are given a joint protocol $\boldsymbol{P'}$ that realizes φ' in E'. We turn this into a joint protocol $\boldsymbol{P}$ in a straightforward way, namely, for each u, if $P'_i(u) = (a, \Psi)$, we set $P_i(u) = a$. We observe that if $r = (s_0, \Psi_0), (s_1, \Psi_1), \ldots$ is a run of $\mathcal{I}(E', \boldsymbol{P'})$, then $r{\downarrow} = s_0 s_1 \ldots$ is a run of $\mathcal{I}(E, \boldsymbol{P})$. Moreover, there are no other runs in $\mathcal{I}(E, \boldsymbol{P})$, and $O'_i((r, m)) = O_i((r{\downarrow}, m))$. We claim that if $\mathcal{I}(E', \boldsymbol{P'}), (r, m) \models \overline{\psi}$, then $\mathcal{I}(E', \boldsymbol{P'}), (r, m) \models \psi$ and $\mathcal{I}(E, \boldsymbol{P}), (r{\downarrow}, m) \models \psi$.

The proof goes by induction on the size of ψ. The claim is trivially true for atomic formulas and negated atomic formula, and, clearly, the induction step goes through for disjunction and conjunction and temporal operators. (Note that this would not go through for negation!)

So let's look at a knowledge operator, say $\psi = \mathsf{K}_i\rho$, and let's assume $\mathcal{I}(E', \boldsymbol{P'}), (r, m) \models \overline{\psi}$. By definition of the flattened variant, we have $\overline{\psi} = \mathsf{X}p_\psi$. Thus, $\mathcal{I}(E', \boldsymbol{P'}), (r, m) \models \mathsf{X}p_\psi$, and hence $\psi \in \Psi(r, m+1)$. By construction, we then get $P'_i(O_i((r, m))) = (a, \Psi)$ for some action a and some set Ψ where $\psi \in \Psi$. This, in turn, means $\psi \in \Psi(r', m+1)$ for every point (r', m) of $\mathcal{I}(E', \boldsymbol{P'})$ with $(r', m) \sim_i (r, m)$. Fix such a point. From the semantics of X, we can conclude $\mathcal{I}(E', \boldsymbol{P'}), (r', m) \models \mathsf{X}p_\psi$. Since $\mathcal{I}(E', \boldsymbol{P}) \models \chi$, we get that $\mathcal{I}(E', \boldsymbol{P'}), (r', m) \models \overline{\rho}$, which, by induction hypothesis, yields $\mathcal{I}(E', \boldsymbol{P'}), (r', m) \models \rho$ and $\mathcal{I}(E, \boldsymbol{P}), (r'{\downarrow}, m) \models \rho$. Since (r', m) was chosen arbitrarily, we finally get $\mathcal{I}(E', \boldsymbol{P'}), (r', m) \models \mathsf{K}_i\rho$ and $\mathcal{I}(E, \boldsymbol{P}), (r{\downarrow}, m) \models \mathsf{K}_i\rho$. This proves the claim.

6 Conclusion

We have shown that there exist classes of environments and formulas (broadcast, or hierarchical environments and positive formulas) for which synthesis from specifications in the logic of linear time and knowledge is decidable. These results suggest several directions for further research. One is to obtain

a more general characterization of the decidable cases, as has been done for temporal specifications [Mad01]. There are also good reasons to explore versions of these problems where the temporal logic used is for branching time rather than linear time. In particular, results on branching-time versions would be directly applicable to classical security notions such as deducibility on strategies, which we have closely appromixated but not precisely captured with our notion of strong deducibility on strategies. Where such notions are found to be decidable, it is moreover of interest to find precise complexities and develop specially tailored decision procedures for the automation of security analysis. Another potential area of application is the compilation of knowledge-based programs [FHMV95]. Finally, our assumption that the protocols synthesized are deterministic should also be relaxed, particularly as it is often the case that security is attained by creative use of non-determinism.

References

[AAE04] P. C. Attie, A. Arora, and E. A. Emerson. Synthesis of fault-tolerant concurrent programs. *ACM Transactions on Programming Languages and Systems*, 26(1):125–1851, 2004.

[AE98] P. C. Attie and E. A. Emerson. Synthesis of concurrent systems with many similar processes. *ACM Transactions on Programming Languages and Systems*, 20(1):51–115, 1998.

[AE01] P. C. Attie and E. A. Emerson. Synthesis of concurrent programs for an atomic read/write model of computation. *ACM Transactions on Programming Languages and Systems*, 23(2):187–242, 2001.

[Ant95] M. Antoniotti. *Synthesis and verification of discrete controllers for robotics and manufacturing devices with temporal logic and the Control-D system.* PhD thesis, New York University, New York, 1995.

[AVW03] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controlers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

[Cha88] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *J., Cryptology*, (1):65–75, 1988.

[EC82] E.A. Emerson and E.M. Clarke. Using branching time logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[EvdMS03] K. Engelhardt, R. van der Meyden, and K. Su. Modal logics with a hierarchy of local propositional quantifiers. In P. Balbiani, N. Suzuki, F. Wolter, and M. Zakharyaschev, editors, *Advances in Modal Logic*, volume 4, pages 9–30. World Scientific, 2003.

[FF90] M. Fujita and H. Fujisawa. Specification, verification, and synthesis of control circuits with propositional temporal logic. In J.A. Darringer and F.J. Rammig, editors, *Computer Hardware Description Languages and Their Applications*, pages 265–279. North-Holland, 1990.

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *LNCS*. Springer, New York, 2002.

[HO03] J. Y. Halpern and K. O'Neill. Anonymity and information hiding in multiagent systems. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, pages 75–88, 2003.

[KV97] O. Kupferman and M.Y. Vardi. Synthesis with incomplete informatio. In *2nd International Conference on Temporal Logic*, pages 91–106, Manchester, July 1997.

[Mad01] P. Madhusudan. *Control and Synthesis of Open Reactive Systems*. PhD thesis, University of Madras, Nov 2001.

[May03] Richard Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1–3):337–354, March 2003.

[Mey96] R. van der Meyden. Finite state implementations of knowledge-based programs. In *Proceedings of the Conference on Foundations of Software Technology and Theoretical Computer Science*, Springer LNCS No. 1180, pages 262–273, Hyderabad, India, December 1996.

[MS04] R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proc. 17th IEEE Computer Security Foundations Workshop*, pages 280–291, June 2004.

[MV98] R. van der Meyden and M. Y. Vardi. Synthesis from knowledge-based specifications. In *CONCUR'98, 9th International Conf. on Concurrency Theory*, Springer LNCS No. 1466, pages 34–49, Sept 1998.

[MW84] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, January 1984.

[PR89a] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symposium on Principles of Programming Languages*, Austin, January 1989.

[PR89b] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. 16th Int. Colloquium on Automata, Languages and Programming*, volume 372, pages 652–671. Lecture Notes in Computer Science, Springer-Verlag, July 1989.

[PR90] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. 31st IEEE Symposium on Foundation of Computer Science*, pages 746–757, 1990.

[Ros92] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1992.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, pages 134–191. Elsevier, Amsterdam, 1990.

[Tho97] Wolfgang Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 389–455. Springer, Berlin, 1997.

[Var95] M.Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In P. Wolper, editor, *Computer Aided Verification, Proc. 7th Int'l Conf.*, volume 939 of *Lecture Notes in Computer Science*, pages 267–292. Springer-Verlag, Berlin, 1995.

[Wal04] I. Walukiewicz. A landscape with games in the background. In *Proc. IEEE Symposium on Logic In computer Science*, pages 356–366, 2004.

[Wit90] D.M. Wittbold, J.T.and Johnson. Information flow in nondeterministic systems. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 144–161, 1990.

[WTD91] H. Wong-Toi and D.L. Dill. Synthesizing processes and schedulers from temporal specifications. In E.M. Clarke and R.P. Kurshan, editors, *Computer-Aided Verification'90*, volume 3 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 177–186. AMS, 1991.