Knowledge = observation + memory + computation*

Blaise Genest[1], Doron Peled[2], and Sven Schewe[3]

[1]CNRS, IRISA, Rennes, France
[2]Bar Ilan University, Israel
[3]University of Liverpool, UK

Abstract. We compare three notions of knowledge in concurrent system: memoryless knowledge, knowledge of perfect recall, and causal knowledge. Memoryless knowledge is based only on the current state of a process, knowledge of perfect recall can take into account the local history of a process, and causal knowledge depends on the causal past of a process, which comprises the information a process can obtain when all processes exchange the information they have when performing joint transitions. We compare these notions in terms of knowledge strength, number of bits required to store this information, and the complexity of checking if a given process has a given knowledge. We show that all three notions of knowledge can be implemented using finite memory. Causal knowledge proves to be strictly more powerful than knowledge with perfect recall, which in turn proves to be strictly more powerful than memoryless knowledge. We show that keeping track of causal knowledge is cheaper than keeping track of knowledge of perfect recall.

1 Introduction

Knowledge represents the information that processes can have about each other and, consequently, about the state of the entire system. In concurrency theory, there are multiple definitions of knowledge based on the specification of the system, a limited view of the other processes, and some information related to the observed history [11]. We study three types of knowledge for concurrent systems. According to the first type, *memoryless knowledge*, a process knows everything consistent with all executions that end in its current local state. For the second type, *knowledge of perfect recall* [3, 11, 12], a process knows everything consistent with all executions that share the same local history visible to this process. We define a third type of knowledge, *causal knowledge*, where a process knows everything consistent with all executions that have the same past, where the past of a process includes the past of other processes up to their last joint transition.

We are interested in the implementation of different kinds of knowledge as a transformation of the system under consideration. The transformation can use additional variables in order to collect information about history, and also to pass this information from process to process as part of the scheduled system synchronization. In particular,

* The work of the second author was partly supported by ISF grant 126-12 "Practical Synthesis of Control for Distributed Systems", and partly done while invited professor in University of Rennes 1

such a transformation can keep information related to the observable history in order to obtain additional knowledge. This transformation cannot change the values of the original variables of the program (including program counters) or the enabledness condition of the transitions (operations) of the system. Thus, the executions of the original system are projections of the executions of the transformed system; only further information is collected in new variables. The different kinds of knowledge become memoryless knowledge after the transformation, which stores all information required in the processes' local states.

This transformation can be used to monitor the global behavior of a system by local processes. For example we may use it to control the system to force it to satisfy some global property by blocking transitions based on knowledge [1, 8, 16]. Another application is to perform some run time checking that a process satisfies some global properties when reaching particular local states.

Our study differs from the classical question of model-checking knowledge [11], as it does not attempt to provide algorithms for checking the knowledge of processes. Instead, we are interested in providing the run-time support to use knowledge. In particular we are interested in the implementing algorithms and their complexity. When comparing the commonly used memoryless knowledge and knowledge of perfect recall [1, 11], there is a tradeoff between the amount of knowledge available to processes and the complexity of maintaining it. We can know more properties under perfect recall, but have to maintain some history related information for that. Quite surprisingly, the new definition of causal knowledge both improves our knowledge and reduces the time and space complexity required when compared to knowledge of perfect recall. The price to pay for this is increased communication: processes have to update each other, through communication, when performing joint transitions in order to achieve this type of knowledge. We show that implementing the third kind of knowledge, knowledge based on causality, can be obtained using a construction based on the "gossip" automata of Mukund and Sohoni [14], related to the Zielonka construction [17, 7, 5].

We establish complexity results for implementing the different types of knowledge. In particular, we show that causal knowledge requires less memory, thanks to the sharing of information during communication. It is, however, interesting to note the stark difference in cost between implementing causal knowledge and knowledge of perfect recall: communication does not only improve knowledge, it also saves resources.

2 Transition systems

Definition 1. *A transition system is a tuple* $\mathsf{Tr} = \langle P, V, L_v, T, L_t, S, s_0, R\rangle$ *where*

- P *is a finite set of processes.*
- V *is a finite set of Boolean variables.*
- $L_v : V \to P$ *is a mapping from variables to processes, such that each variable* v *is local to the process* $L_v(p)$*. Let* $V_p = \{x \mid L_v(x) \in p\}$ *(the set of variables of process* p*).*
- T *is a finite set of transitions, where each transition* $\tau \in T$ *has an enabling condition* $\mathsf{en}_\tau \subseteq 2^V$*, which is a propositional property, and a transformation* $f_\tau : 2^V \to 2^V$

over the set of variables. The enabledness condition and transformation have some constraints as described below.

$L_t : T \to 2^P$ *maps each transition to the set of processes that execute it. The transition is executed synchronously by the processes in* $L_t(\tau)$. *Let* $\mathsf{var}(\tau) = \bigcup_{p \in L_t(\tau)} V_p$. *Then only the variables* $\mathsf{var}(\tau)$ *can be used in* en_τ, *and* f_τ *can use and change only these variables.*

$S \subseteq 2^V$ *is a finite set of states, where each state of* Tr *is a possible assignments of values to the variables of* V.

$s_0 \in S$ *is the* initial state.

$R \subseteq S \times S$ *is a relation over* S. *We have* $(s, s') \in R$ *exactly when there exists some transition* $\tau \in T$ *such that* $s \models \mathsf{en}_\tau$ *and* $s' = f_\tau(s)$. *We say that* τ *can be executed (is enabled) from* s, *producing* s'.

For some future constructions, it will be convenient to assume at times that the system is first executing some initial ("mythological") transition ℓ, shared by all the processes, i.e., $L_t(\ell) = P$, ending up with the initial state s_0 (starting with some ("mythological") initial state s_{-1}).

We assume that S is the set of states reachable from s_0, such that the global state space S and the global transition relation R over S are defined by the other components.

The size $|\mathsf{Tr}|$ of a transition system Tr is the number $|P|$ of processes plus the number $|V|$ of variables plus the number $|T|$ of transitions.

We define a *local state* $s|_p$ as the projection of global state $s \in S$ on the local variables V_p of process p. For a set of processes $P' \subseteq P$, the semi-local state $s|_{P'}$ is the projection of s to the variables in $\bigcup_{p \in P'} V_p$. In particular, we have $s|_P = s$.

Definition 2. *A* history *of a transition system* Tr *is an alternating sequence* $h = s_0\tau_1 s_1 \tau_2 s_2 \ldots s_n$ *of states and transitions such that, for each* $i \geq 0$, $s_i \models \mathsf{en}_{\tau_{i+1}}$ *and* $s_{i+1} = f_{\tau_{i+1}}(s_i)$. *We denote by* $\mathsf{last}(h)$ *the last state* s_n *of* h, *and* $\mathsf{last}_p(h) = \mathsf{last}(h)|_p$.

A state s is *reachable* if $s = \mathsf{last}(h)$ for some history h. Note that it is PSPACE-complete to check whether a state is reachable in a transition system [13]. As the initial state is unique and the effect of transitions is deterministic, we sometimes use only the sequence of transitions $\tau_1\tau_2\tau_3 \ldots$ to denote a history or execution $s_0\tau_1 s_1 \tau_2 s_2 \tau_3 s_3 \ldots$ of a transition system.

3 Notions of knowledge

In order to avoid using a specific logical formalism, we define state properties abstractly:

Definition 3. *A* (state) property φ *of a transition system* Tr *is a subset of its states. That is,* $\varphi \subseteq S$. *A state* s *satisfies* φ, *denoted* $s \models \varphi$, *if* $s \in \varphi$. *An history* h *satisfies* φ, *denoted* $h \models \varphi$, *if* $\mathsf{last}(h) \in \varphi$.

Note that properties can be defined compactly, using, for example, propositional logic. In order to define a general notion of knowledge of state properties, the different

kinds of knowledge are abstracted as information available to a process. In order to define different kinds of knowledge, we define an equivalence relation between histories. Let Γ represent a type of knowledge. (The types of knowledge that we consider will be presented later.)

Definition 4. *Let $\equiv_p^\Gamma$ be an equivalence relation between histories with respect to process $p \in P$ in a transition system* Tr. *Process p in a transition system* Tr *knows a state property φ after history h, according to knowledge type Γ, denoted $h \models K_p^\Gamma \varphi$, if, for each history h' such $h \equiv_p^\Gamma h'$, $\mathsf{last}(h') \models \varphi$.*

We study three types of knowledge: memoryless, perfect recall, and causal knowledge. Accordingly, Γ is ML, PR and C, respectively.

Definition 5. *We say that the knowledge type Γ is* deeper[1] *than knowledge type Γ', denoted $\Gamma \rhd \Gamma'$, if, for each history h, process p and property φ, $h \models K_p^{\Gamma'} \varphi$ implies $h \models K_p^\Gamma \varphi$. If $\Gamma \rhd \Gamma'$, but $\Gamma' \not\rhd \Gamma$, then we call Γ* strictly deeper *than Γ', denoted $\Gamma \blacktriangleright \Gamma'$.*

A simple observation that follows immediately from the above definition can be used to show that one kind of knowledge is deeper than another:

Observation 1 *Let Γ and Γ' be two notions of knowledge with $\equiv_p^\Gamma \subseteq \equiv_p^{\Gamma'}$ for each p. Then $\Gamma \rhd \Gamma'$.*

Memoryless knowledge. This is a conservative version of knowledge, where a property φ is known if it holds in all the states with the same local state of process p. That is, $h \equiv_p^{ML} h'$ if $\mathsf{last}_p(h) = \mathsf{last}_p(h')$.

Knowledge of perfect recall. In the epistemic community, knowledge of perfect recall [3, 11, 12] refers to the ability of process p to use local observation to distinguish between different histories. We can define, in fact, multiple different versions of knowledge of perfect recall:

- $PR(l)$ A process can view (and recall) its local states along the executions (this is the version that is used in [11]).
- $PR(t)$ A process can view the occurrences of transitions in which it participates.
- $PR(lt)$ A process can view both the local state and the executed transition.
- $PR(ct)$ A process can, when executing a transition, view the combined local state of the processes involved in this transition.

We choose $PR = PR(lt)$ as our canonical definition of knowledge of perfect recall. The observations in this case are sequences of $p{-}events$, as defined below.

Definition 6. *A $p{-}event$ is a pair $\langle \tau, r \rangle$, where $\tau \in T$, $p \in L_t(\tau)$, and r is a local state of p. A $p{-}event$ is obtained from a history by taking a transition τ that is executed and involves the process p and the local state just after its execution. We define the sequence of $p{-}events$ of a history h, $\mathsf{Ev}_p(h)$, inductively. For $h = s_0$, $\mathsf{Ev}_p(s_0) = \epsilon$, the empty word. Let $h' = h\,\tau\,s$ (that is, h' extends h with a transition τ, leading to state s). Now, if $p \in L_t(\tau)$, then $\mathsf{Ev}_p(h') = \mathsf{Ev}_p(h)\langle \tau, s|_p \rangle$, and otherwise $\mathsf{Ev}_p(h') = \mathsf{Ev}_p(h)$.*

[1] We use the term "deeper" instead of "stronger", as the latter is associated with an implication of the opposite direction: in logic, φ is stronger than φ' when $\varphi \to \varphi'$.

For instance, for $p \in L_t(\tau)$, $p \notin L_t(\tau')$, and $h = s_0\,\tau\,s\,\tau'\,t\,\tau\,r$, we have $\mathsf{Ev}_p(h) = \langle \tau, s|_p\rangle\langle \tau, r|_p\rangle$.

Definition 7. *Knowledge of perfect recall is based on the equivalence $\equiv_p^{PR}$ such that $h \equiv_p^{PR} h'$ exactly when $\mathsf{Ev}_p(h) = \mathsf{Ev}_p(h')$.*

Similarly, $PR(t)$ is defined based on the projection of $\mathsf{Ev}_p(h)$ on its first components, while $PR(l)$ is defined based on the projection of $\mathsf{Ev}_p(h)$ on its second components. Also, $PR(ct)$ is defined based on a sequence of extended events of the form $\langle \tau, s|_{L(\tau)}\rangle$, using the respective semi-local states (rather than the local states) in $\mathsf{Ev}_p(h)$. In the example above, this would be $\mathsf{Ev}_p(h) = \langle \tau, s|_{L_t(\tau)}\rangle\langle \tau, r|_{L_t(\tau)}\rangle$.

Notice that the sequence Ev_p can grow arbitrarily as the history grows. However, it has been shown [12, 1] that a bounded implementation of $PR(l)$ is possible. We will give a uniform implementations for all versions of PR in the next section. We now compare ML and the different definitions of PR.

Lemma 1. *Knowledge of perfect recall is strictly deeper than memoryless knowledge. More precisely, $PR(ct) \blacktriangleright PR(lt) \blacktriangleright PR(l) \blacktriangleright ML$ and $PR(lt) \blacktriangleright PR(t)$. However, $PR(t)$ is incomparable (with respect to $\rhd$) with ML and with $PR(l)$.*

Proof. By definition, the relation $\equiv_p^{PR(ct)}$ refines $\equiv_p^{PR(lt)}$, which refines both $\equiv_p^{PR(l)}$ and $\equiv_p^{PR(t)}$, for all $p \in P$. Now, $\equiv_p^{PR(l)}$ keeps the sequence of states of p, and in particular the last one. Hence $h \equiv_p^{PR(l)} h'$ implies $\mathsf{last}_p(h) = \mathsf{last}_p(h')$, hence the relation $\equiv_p^{PR(l)}$ refines $\equiv_p^{ML}$. We now show that the implications are strict.

"$PR(ct)$ vs. $PR(lt)$": To show strictness, we consider the histories $h = ac$ and $h' = bc$ for the transition system from Figure 1. Obviously, $h \models \{2\}$ and $h \models K_{P_2}^{PR(ct)}\{2\}$, because, under $PR(ct)$, P_2 knows after executing the joint c transition that P_1 is in (the sink) state 2. At the same time, $h' \not\models \{2\}$, and under $PR(lt)$, P_2 sees the same sequence of $P_2-events$: $\mathsf{Ev}_{P_2}^{PR(lt)}(h) = \mathsf{Ev}_{P_2}^{PR(lt)}(h')$. Thus, $h \not\models K_{P_2}^{PR(lt)}\{2\}$.

"$PR(t)$ vs. $PR(l)$": Consider the histories $h = ad$ and $h' = bc$ for the transition system from Figure 1. Obviously, $h \models \{2\}$ and $h \models K_{P_2}^{PR(t)}\{2\}$, because, under $PR(t)$, P_2 knows after executing the joint d transition that P_1 is in (the sink) state 2. At the same time, $h' \not\models \{2\}$, but, under $PR(l)$, P_2 sees the same sequence of $P_2-events$: $\mathsf{Ev}_{P_2}^{PR(l)}(h) = \mathsf{Ev}_{P_2}^{PR(l)}(h')$. Thus, $h \not\models K_{P_2}^{PR(l)}\{2\}$. This also implies strictness for "$PR(lt)$ vs. $PR(l)$" and implies that ML is not deeper than $PR(t)$.

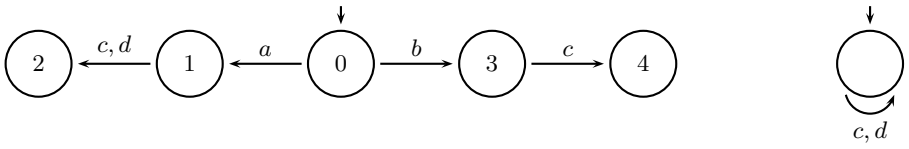

Fig. 1. Local state space of two Processes: P_1 (left) with a variable that can take 5 values $\{0, 1, 2, 3, 4\}$ and P_2 without variable. Transitions c, d are joint between P_1 and P_2.

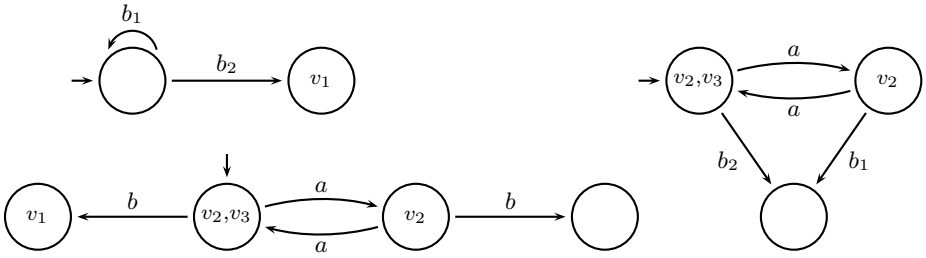

Fig. 2. The bottom left shows the global state space for two Processes P_1 (top left) and P_2 (right), with $V_{P_1} = \{v_1\}$ and $V_{P_2} = \{v_2, v_3\}$. Boolean variables represented in the states are those with value true. Transition a is local to P_2. It toggles the value of the variable v_3. Transition b is joint between P_1 and P_2. It first assigns v_1 the value of v_3 and then sets v_2 and v_3 to false. Both transitions are enabled if, and only if, v_2 is true. Initially, v_2 and v_3 are set to true and v_1 is set to false. Thus, the effect of b on the each process depends on the state of the other process. To reflect this, the action b is graphically 'split' into b_1 and b_2 in the graphic representation of the processes. Note, however, that b_1 and b_2 refer to the same action, b.

"$PR(l)$ vs. ML": We consider the histories $h = \varepsilon$ and $h' = bc$ for the transition system from Figure 1. Obviously, $h \models \{0,1,2,3\}$ and $h \models K_{P_2}^{PR(l)}\{0,1,2,3\}$, because, under $PR(l)$, P_2 knows after h that it has not taken part in any transition, and 4 is only reachable upon taking a c transition. At the same time, $h' \not\models \{0,1,2,3\}$, but $\mathsf{last}(h)|_{P_2} = \mathsf{last}(h')|_{P_2}$. Thus, $h \not\models K_{P_2}^{ML}\{0,1,2,3\}$.

"ML vs. $PR(t)$": To show that $PR(t)$ is not deeper than ML, and thus not deeper than $PR(l)$ and $PR(lt)$, we consider the transition system from Figure 2. Consider the histories $h = b$ and $h' = ab$. Obviously, $h \models \{v_1\}$ and $h' \not\models \{v_1\}$. Under $PR(t)$, P_1 sees the same sequence of $P_1-events$ for h, h': $\mathsf{Ev}_{P_1}^{PR(t)}(h) = \mathsf{Ev}_{P_1}^{PR(t)}(h')$. Thus, $h \not\models K_{P_1}^{PR(t)}\{v_1\}$. However, $\mathsf{last}(h)|_{P_1} = \{v_1\}$, hence $h \models K_{P_1}^{ML}\{v_1\}$ holds.

Notice that $PR(t)$ is deeper than $PR(l)$ when, for all processes p and all transitions τ, the p-local state after the transition τ only depends on the p-local state before τ. In this case, the history of local states can be retrieved from the history of transitions. This is, for example, the case for products of finite state systems.

Causal knowledge. This notion is related to partial order semantics [10], and has been used informally in distributed games [4, 15, 6]. However, as far as we know, it has not been used in an epistemic framework before. The assumption is that processes may exchange information each time they perform a joint transition.

We first define the chain of transitions that can affect the view of a process p in a given history. The exact ordering of transitions is not necessarily known to p, hence the information function is represented as a partial order. We now define the partial order associated with a history.

Definition 8. *The* partial order $PO(h)$ *associated with a history* $h = s_0\tau_1 s_1\tau_2 s_2 \ldots \tau_n s_n$ *is a triple* $\langle E, \lambda, \prec\rangle$ *where*

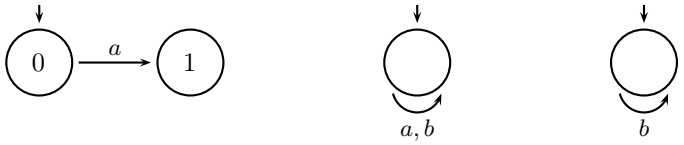

Fig. 3. Processes P_1 with a boolean variable (left), P_2 (middle) and P_3 (right) without variables

- $E = \{e_0, e_1, \ldots, e_m\}$ *is the set of occurences of events in* h*.*
- $\lambda : E \to T$ *labels* $\lambda(e_i) = t_i$*,*
- $\prec \subseteq E \times E$ *is the smallest partial order relation (i.e., transitive, reflexive and asymmetric relation) satisfying the following: if* $e, e' \in E$ *with* $L_t(\lambda(e)) \cap L_t(\lambda(e')) \neq \emptyset$ *and* e *appearing before* e' *in* h *then* $e \prec e'$*.*

For instance, for three processes p, q, and r and three transitions a, b, and c with $L_t(a) = p$, $L_t(b) = q$, $L_t(c) = \{p, r\}$, we have $PO_p(abc) = PO_p(acb) = \langle\{e_a, e_b, e_c\}, \prec, \lambda\rangle$ with $\lambda(e_x) = x$ for all $x \in \{a, b, c\}$ and $e_a \prec e_c$. Process p sees transitions in the past (for $\prec$) of the last transition on p.

Definition 9. *The causal view* $Ca_p(h)$ *of process* p *in history* h *includes all the occurrences of events that precede its last occurrence according to the partial order* $PO(h) = \langle E, \prec \ \lambda\rangle$*. If* e *is the latest occurrence of* h *that involves* p*, then let* $E' = \{e' \in E | e' \preceq e\}$*. Then,* $Ca_p(h) = \langle E', \prec \cap E' \times E', \lambda|_{E'}\rangle$*.*

The equivalence $\equiv_p^C$ *on histories, used to define causal knowledge, is based on* $h \equiv_p^C h'$ *iff* $\mathsf{Ca}_p(h) = \mathsf{Ca}_p(h')$*.*

Lemma 2. *Causal knowledge is strictly deeper than knowledge of perfect recall.*

Proof (Sketch). We first prove $Ca \rhd PR(ct)$. By Lemma 1, this implies $Ca \rhd PR$ for all version of PR. Although the causal view $\mathsf{Ca}_p(h)$ of a process p in a history h is a partial order, it contains, according to Definition 9, in particular, all the occurrences of transitions of p. The occurrences of transitions in which p participates are totally ordered by $\prec$ in the causal view. Given the unique mythological event and the causal view, one can also construct the $p{-}events$ corresponding to the occurrences in E and, in particular, the occurrences in which p participates. To do this, one can complete the partial order $\prec$ into some total order that contains it, and start to calculate the global state after each occurrence, taking the relevant component of p for transitions involving this process. Although the global states generated in this way are not necessarily the ones appearing in h, one can show by induction over the length of the constructed sequence that the $p{-}events$ are the same. This is the case, because occurrences of h that are not in E do not affect the values of occurrences in E. Moreover, by the disjointness of the variables for each process, the order of occurrences not in E can be commuted with occurrences in E to appear at the end, without affecting the value of the p local states.

To show that causal knowledge is *strictly* deeper than knowledge of perfect recall, we consider the histories $h = ab$ and $h' = b$ in the transitions system from Figure 3.

Obviously, $h \models \{1\}$ holds. Further, $h \models K_{P_3}^{Ca}\{1\}$: the partial order $PO(h)$ associated with h is the total order $\langle\{e_1, e_2\}, \lambda(e_1) = a, \lambda(e_2) = b, e_1 < e_2\}\rangle$, and we have $Ca_{P3}(h) = PO(h)$. That is, Process P_3 knows that P_1 is in 1.

At the same time, $h' \not\models \{1\}$, but, under $PR(ct)$, P_3 sees the same sequence of $P_3-events$: $\mathsf{Ev}_{P_3}^{PR(ct)}(h) = \mathsf{Ev}_{P_3}^{PR(ct)}(h')$. Thus, $h \not\models K_{P_3}^{PR(ct)}\{1\}$. □

Notice that the third process in the proof is necessary, because, for two process and $PR(ct)$, each process learns the global state of the transition system when executing a joint transition. Ca and $PR(lt)$ can be separated using the transition system with two processes from Figure 1. This indeed follows from the proof that $PR(lt)$ is not deeper than $PR(ct)$.

4 Application of knowledge

Knowledge can be applied to control systems by blocking some transitions. The more is known about a transition system, the less restrictive the control needs to be. Consider the system with three processes arb, p, p' from Figure 4. Process p needs to access a critical section twice, and process p' needs to access it once. Process arb helps process p, p' to access the critical section in a mutually exclusive way.

Processes p and p' can try to enter the critical section using an e and e' transition, respectively, which they share with the arbiter, and leave it using the shared l and l' transition, respectively. The effect of transition e (resp. e') depends on the state of the arbiter. If arb has given permission to the other process to enter the critical section, and not yet received the respective 'leave' transition, transition e does not change the state. In Figure 4, e is therefore split into e_1 (the case where p progresses) and e_2 (the case where p stays in its previous state). Similarly, e' is split into e'_1 and e'_2.

Process p can also ignore the arbiter, and progress using an i transition.

Without control, the system is too permissive. For example, it allows for the history $h = e'i$ (with $h \models c \wedge c'$), where first p' enters the critical section through the e' transition, followed by p entering the critical section.

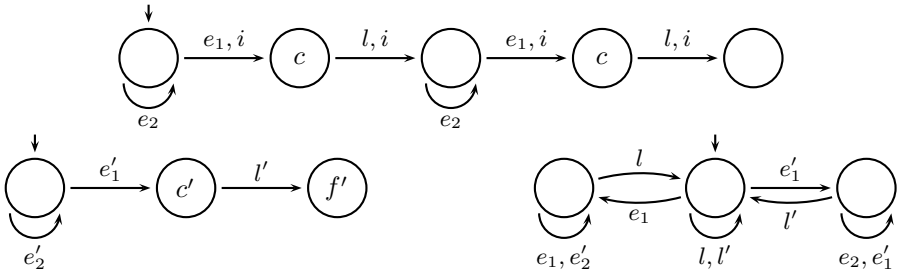

Fig. 4. Three Processes: p (top), p' (bottom left), and arb (bottom right). Only some local boolean variables are represented: c and c' (set to 1 for p and p' in critical section, respectively), and f' set to 1 for p' has finished. The remaining variables are omitted. Effect of the transitions e (resp. e') on process p (resp. p') depends on the arbiter state. It is therefore split into e_1, e_2 (resp. e'_1, e'_2).

However, the following control can be written easily using knowledge: If $K_p(f')$, then allow the 'ignore' transition i. Else, disallow it. Clearly, this only allows p to ignore the arbiter if no future conflict is possible, as p' will remain in this sink state and does no longer compete for entering the critical section.

Now we compare different notions of knowledge. With memoryless knowledge, just looking at its local state, process p will never be able to ignore the arbiter, using i.

With perfect recall, consider the following history: $h = e'el'e$. After this history, process p knows under PR that p' had been in the critical section when it first requested entry, but has left it meanwhile. It thus knows that f' holds henceforth, and p can make use of i, ignoring the arbiter.

With causal recall, Process p can make even more use of i. Consider the history $h = e'l'e$.

With perfect recall, p cannot distinguish it from $h' = e$, and can make no use of i. With causal recall, the knowledge of the arbiter that p' is in its sink state (such that f' holds henceforth) is transferred to p through the shared transition e. It thus knows that f' holds henceforth, and p can make use of i, ignoring the arbiter.

5 Implementation of knowledge using bounded memory

The notions of knowledge discussed in this paper are quite abstract: they represent some mathematical definition based on the observation that the processes can have. These observations are not directly implemented by the processes. Except for theoretical reasoning about programs, knowledge can be used in order to control the program [8]. If the processes need to *use* such observations so that they can act based on their knowledge, they necessarily need to store the observation and act on it. If the observation information is stored and available to a process, it can decide, based on some precalculated knowledge table, to restrict its behavior accordingly.

The definitions of knowledge of perfect recall and knowledge with causal memory are based on unbounded observations. We are interested in transforming the transition system for these two kinds of knowledge, such that only a bounded amount of information is needed. The transformation will add variables and augment the transitions in such a way, that one can control the system based on the knowledge through a precalculated table. In essence, such transformations convert the original system into a new system, where the knowledge can be observed by a process from its most recent local state. We provide complexity measures for both the transformations.

5.1 Implementation of memoryless knowledge

The implementation of memoryless knowledge of a process is simple, as the observation that is used consists only of the local state of the process. To decide the current knowledge regarding a property φ from a p-local state s_p, we recall that p knows that φ holds if, and only if, for all *reachable* states s with $s|_p = s_p$, $s \models \varphi$ holds. It therefore suffices to check the existence of a global state s with $s|_p = s_p$ and $s \not\models \varphi$.

This is a simple reachability problem. An implementation of memoryless knowledge may or may not use an offline precomputation.

Online only. The reachability problem can be solved in time $|\mathsf{Tr}| \cdot 2^{\mathcal{O}(|V|)}$ (or in PSPACE) by constructing all reachable states ending with the observed local state.

With an offline precomputation. Alternatively, for each p-local state, one can first compute the reachable states with this p-local state in a preprocessing step. One can then save this knowledge with respect to p-local states in a binary tree with $2^{|V_p|}$ entries. Accessing this tree at runtime only takes time linear in the number of variables local to p, that is, time $\mathcal{O}(|V_p|)$. The offline construction of the tree during the preprocessing step can be done in time $|\mathsf{Tr}| \cdot 2^{\mathcal{O}(|V|)}$.

5.2 Implementing Knowledge of Perfect Recall

We describe the transformation of knowledge for all version of perfect recall. The transformation was already known for $PR(l)$ [12, 1]. The idea of the construction is that each process p can consult a global automaton, representing the transformation of the entire system. A process p is only aware of the occurrences of its own transitions. Hence, upon an occurrence of a transition τ with $p \in L_t(\tau)$, the automaton moves according to τ. However, process p is not aware of further moves of transitions not involving p. Thus, the actual global state of the system can further change through the firing of any sequence of such transitions. A subset construction can be used to encode the possible global states that can be reached without being distinguished by p after a transition τ.

Definition 10. *Let $\langle S, s_0, T, \delta\rangle$ be a global automaton for the system Tr. Recall the notation $\delta^*(S, \rho)$ that stands for the usual extension of δ from a single state and a single transition into a set of states and a finite (possibly empty) sequences of transitions. Let, for a process p,*

- *$T_p = \{\tau \in T \mid p \in L_t(\tau)\}$ be the set of transtitions executed by p (possibly joined by other processes) and*
- *$I_p = T \setminus T_p$ be the set of transitions that do not involve p.*

Then we construct a deterministic automaton $\mathcal{D}_p = \langle 2^S, S_0, p{-}events, \delta_p\rangle$ such that

- $S_0 = \{\delta^*(\{s_0\}, \rho) \mid \rho \in {I_p}^*\}$,
- $\delta_p(S', \langle \tau, r\rangle) = \bigcup_{\rho \in I_p^*} \delta^*(\{s' \mid \exists s \in S'.\ \delta(s, \tau) = s' \wedge s'|_p = r\}, \rho)$.

$\mathcal{D}_p$ reads a sequence $\tau \in {T_p}^*$ of $p{-}events$. Its state reflects, in which global state the system can be at a point in time, where process p has seen a sequence of $p{-}events$.

Lemma 3. *For a given transition system Tr with automaton $\langle S, s_0, T, \delta\rangle$, a process $p \in P$, and a sequence $h \in T^*$, we have that $s \in {\delta_p}^*\big(S_0, \mathsf{Ev}_p(h)\big)$ if, and only if, there is a sequence h' with $\mathsf{Ev}_p(h) = \mathsf{Ev}_p(h')$ such that $s \in \delta^*(s_0, h')$.*

This can be shown by induction over the length of $\mathsf{Ev}_p(h)$.

For each property φ, we equip $\mathcal{D}_p$ with an acceptance mechanism to obtain $\mathcal{D}_p^\varphi = \langle 2^S, S_0, p{-}events, \delta_p, F_\varphi\rangle$, where the set of final states is:

$$F_\varphi = \{S' \subseteq S \mid \forall s \in S'.\ s \models \varphi\}.$$

For D_p^φ, Lemma 3 provides the following corollary.

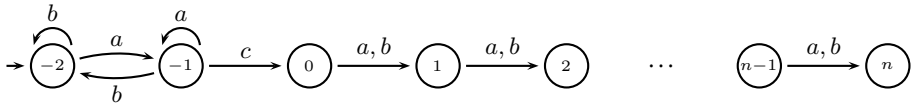

Fig. 5. Process 2 from the proof of Theorem 2.

Corollary 1. *For a given transition system* Tr *with automaton* $\langle S, s_0, T, \delta\rangle$*, a process* $p \in P$*, and a sequence* $h \in T^*$*, we have that* $\mathsf{Ev}_p(h)$ *is accepted by* $\mathcal{D}_p^\varphi$ *iff* $h \models K_p^{\mathrm{PR}}\varphi$*.*

Thus, $\mathcal{D}_p^\varphi$ can be used to check whether or not process p knows φ. The complexity of this construction is quite high: as a subset automaton, $\mathcal{D}_p^\varphi$ can have $2^{|S|}$ states. However, subset automata like $\mathcal{D}_p^\varphi$ can be represented succinctly, such that the representation of a state requires 'only' S bits. As S can be exponential in the number of variables, this translates to $\mathcal{O}(2^{2^{|V|}})$ states, where each state is represented by $\mathcal{O}(2^{|V|})$ bits.

Consequently, we can transform Tr by including, for each process p, an implementation of the automaton $\mathcal{D}_p^\varphi$. This may require $|S|$ additional variables to represent the state of $\mathcal{D}_p^\varphi$, which will be variables local to p. They have no influence on the enabledness of transitions.

These new variables intuitively reflect the subset of the states, in which the system might be in, or, likewise, the set of assignments to its variables consistent with the sequence of $p{-}events$ observed. This representation can be improved: the valuation of the p-local variables V_p is already given by the p-local state, and storing this information again would be redundant. Thus, $\mathcal{O}(2^{|V \setminus V_p|})$ variables are sufficient to represent the additional information.

The above construction implements the $PR = PR(lt)$ knowledge. Similarly, we can implement the other notions of knowledge of perfect recall:

$PR(t)\ \mathcal{D}_p^\varphi = \langle 2^S, S_0, T_p, \delta_p, F_\varphi\rangle$,
$\quad \delta_p(S', \tau) = \bigcup_{\rho \in I_p^*} \delta^*(\{s' \mid \exists s \in S'.\ \delta(s, \tau) = s'\}, \rho)$.
$PR(l)\ \mathcal{D}_p^\varphi = \langle 2^S, S_0, S|_p, \delta_p, F_\varphi\rangle$,
$\quad \delta_p(S', r) = \bigcup_{\rho \in I_p^*} \delta^*(\{s' \mid \exists s \in S'.\ \exists \tau \in T_p\, \delta(s, \tau)|_p = r\}, \rho)$.
$PR(ct)\ \mathcal{D}_p^\varphi = \langle 2^S, S_0, p{-}events, \delta_p, F_\varphi\rangle$,
$\quad \delta_p(S', \langle \tau, r\rangle) = \bigcup_{\rho \in I_p^*} \delta^*(\{s' \mid \exists s \in S'.\ \delta(s, \tau) = s' \wedge s'|_{L_t(\tau)} = r\}, \rho)$.
(Note that the $p{-}events$ for $PR(ct)$ and $PR(lt)$ are different.)

5.3 Lower bound on the transformations for Perfect Recall

We show that the exponential memory blow-up for implementing PR is unavoidable.

Theorem 2. *There exists a family of systems* $(\mathsf{Tr}_n)_{n\in\mathbb{N}}$ *with 2 processes* $\{1, 2\}$*, one variable with* $n + 3$ *valuations (or equivalently* $\lceil \log_2 n + 3 \rceil$ *binary variables), four transitions, and* $n + 3$ *states and a family of assertion* φ_n *such that knowing with* PR *whether* φ_n *holds requires* 2^n *memory states.*

Notice that using $|V|$ binary variables, a counter up to $2^{|V|}$ can be encoded. Hence, Theorem 2 proves that one needs at least $2^{|V|} - 3$ bits of memory to implement PR, even when the description size of Tr_n is polynomial in $|V|$. For convenience, we use a variable with domain $\{-2, -1, 0, \ldots, n\}$ instead of encoding the values in binary. Process 2 is shown in Figure 5.

The proof uses the well-known family $(L_n)_{n\in\mathbb{N}} = \{w \in \{a,b\}^* \mid w = uav \text{ and } |v| = n\}$ of regular languages, accepted by a *non-deterministic* automaton with $n+2$ states but not by any *deterministic* automaton with less than 2^n states.

Proof. The systems are defined as follows:

- there is only one variable v, it is on process 2, and its domain is $\{-2, -1, 0, \ldots, n\}$. Its initial value is -2.
- there are three transitions, $\{a, b, c\}$, with $L_t(a) = L_t(b) = \{1, 2\}$ and $L_t(c) = \{2\}$.
- transitions a, b are enabled in a state v iff $v < n$.
- a leads from -2 and -1 to -1, b leads from -2 and -1 to -2, such that the states -1 and -2 distinguish if the last transition seen has been an a (which is the case in -1 but not in -2),
- c is only enabled if $v = -1$ and updates the valuation of v to 0, and
- finally, for $0 \leq v < n$, transitions a, b increment the value of v by 1.

The state property φ_n is $v \neq n$. Let H_n be the set of histories h such that the suffix of h is bw with the number of a, b in w is n (w can have 0 or one c). We have that $\forall h \in H_n$, $last(h) \models \varphi_n$. The reason is that after a c, there are at most n letters a, b. Now, writing $h = ubw$, there cannot be a c in u, as there are $n+1$ transition at least being done afterwards. That is, ub reached state -2. Now, it means that the first letter of w is not a c, and thus it is an a or a b. In any case, if c happens in w, there will be strictly less than n letters after it, and thus the valuation $v = n$ cannot be reached.

Notice now that $h \in H_n$ iff $\mathsf{Ev}_1(h) = ubw$ with $w \in \{a,b\}^n$ (process 1 has a unique state as it has no variable, hence we do not indicate it in the $p-events$). Thus for all h' such that $\mathsf{Ev}_1(h') = \mathsf{Ev}_1(h)$, we have $h' \in H_n$. Thus process 1 knows φ_n after any $h \in H_n$ using PR.

Assume by contradiction that there is an implementation of PR with less than 2^n memory states. Process 1 has no variables, such that its memory is updated only based on the sequence $Ev_1(h)$. As there are less than 2^n states, there exists 2 histories $u_1 \cdots u_n \neq u'_1 \cdots u'_n \in \{a,b\}^*$ leading to the same state s of the implementation. Let $u'_i \neq u_i$, let say $u_i = b$ and $u'_i = a$. Now, let us consider the histories $h = u_1 \cdots u_n a^{n-i}$ and $h' = u'_1 \cdots u'_i c u'_{i+1} \cdots u'_n a^{n-i}$. Clearly, $h \in H_n$, and thus process 1 knows φ_n after h using PR. However, the memory state after histories h, h' are the same, as a same sequence of 1-event is seen from state s. However, $last(h') = (v = n) \not\models \varphi_n$, hence 1 does not know φ_n after h using this implementation. A contradiction. □

5.4 Implementation of Causal Knowledge

In order to provide a finite representation for causal knowledge we will adapt a construction by Mukund and Sohoni [14] for gossip automata.

Recall Definition 8 of a partial order $PO(h) = \langle E, \lambda, \prec \rangle$ associated with a history h and the causal view of a process p in h (Definition 9). We use the following notation:

– Recall that $\mathsf{last}_p(h) \in 2^{V_p}$ is the p-state reached by h (or equivalently by $PO(h)$),
– $\mathsf{latest}_p(h) = \max_{\prec}\{e \in E \mid p \in L_t(\lambda(e))\}$. This is the most recent occurrence of an event in h that is executed by p, and therefore the last occurrence of an event in $Ca_p(h)$. Notice that the p-state reached on $Ca_p(h)$ is also $\mathsf{last}_p(h)$,
– $\mathsf{latest}_{p\leftarrow q}(h) = \max_{\prec}\{e \in E \mid q \in L_t(\lambda(e))$ and $e \prec \mathsf{latest}_p(E)\}$. This is the most recent occurrence on q that precedes (or is the same as) the most recent occurrence of p. We denote by $\mathsf{last}_{p\leftarrow q}(h)$ the q-state reached on $Ca_p(h)$, which corresponds to the q-state reached by $\mathsf{latest}_{p\leftarrow q}(h)$.

The set $\mathsf{R}_p(h) = (\mathsf{last}_{p\leftarrow q}(h))_{q\in P}$ is the global state reached by $Ca_p(h)$. We can define the associated equivalence relation:

Definition 11. $h \approx_p h'$ *iff* $\mathsf{R}_p(h) = \mathsf{R}_p(h')$.

We can define a knowledge R based on this $\approx_p$. It is immediately clear that $\mathsf{Ca} \rhd \mathsf{R}$, since $\equiv_p^{\mathsf{C}}$ refines $\approx_p$. In fact, we have equality:

Lemma 4. R *is as deep as* Ca.

Proof. (Sketch) It is enough to show that for every history h, $\{\mathsf{last}(h') \mid Ca_p(h') = Ca_p(h)\} = \{\mathsf{last}(h') \mid R_p(h') = R_p(h)\}$.

We have trivially $\{h' \mid Ca_p(h') = Ca_p(h)\} \subseteq \{h' \mid R_p(h') = R_p(h)\}$ for every history h. Hence it suffices to prove that $\{\mathsf{last}(h') | R_p(h') = R_p(h)\} \subseteq \{\mathsf{last}(h') | Ca_p(h') = Ca_p(h)\}$ for every history h.

Let h' such that $R_p(h') = R_p(h)$. We thus have that the global state reached by $Ca_p(h)$ is the same as the global state reached by $Ca_p(h')$. By definition of $Ca_p(h')$, h' can be obtained from $Ca_p(h')$ by performing a sequence w of occurrences not on p. Now, consider doing this sequence w of occurrences from $Ca_p(h)$. It is possible as the global states reached by $Ca_p(h)$ and by $Ca_p(h')$ is the same. Hence we obtain a history $h'' = Ca_p(h)\, w$. Because the system is deterministic from a global state, $\mathsf{last}(h'') = \mathsf{last}(h')$ holds. We conclude by remarking that $Ca_p(h'') = Ca_p(hw) = Ca_p(h)$. □

This means that, for p, keeping $\mathsf{last}_{p\leftarrow q}(h)$ for all q is enough to implement Ca_p. Keeping this information is, however, not totally straightforward. Indeed, when performing transition a, all processes q involved in that transition a will have a value for $\mathsf{last}_{q\leftarrow r}(h)$ for all process r just before performing a. First, we have $\mathsf{last}_{p\leftarrow r}(ha) = \mathsf{last}_{q\leftarrow r}(ha)$ for all $p, q \in L_t(a)$ and all process r: each process will update the value in the same way. Let $state$ be the tuple $(\mathsf{last}_{s\leftarrow s}(h))_{s\in L_t(a)}$ aggregating all the latest s-state from all processes s involved in a. It is easy to see that, if $r \in L_t(a)$ is involved in a, then $\mathsf{last}_{q\leftarrow r}(ha) = \delta(state, a)_r$. That is, it suffices to deterministically perform a from $state$ and take the r-component. Now, the difficulties appear for $r \notin L_t(a)$, that is, if r is not involved in the transition. Then, it is easy to see that, for all $q \in L_t(a)$ involved in a (for which we need to update their state), $\mathsf{last}_{q\leftarrow r}(ha) = \mathsf{last}_{s\leftarrow r}(ha)$ for some process $s \in L_t(a)$. The question is, which one of all the process $s \in L_t(a)$ has the freshest information about r. If we know this, then every process p can keep accurately $\mathsf{last}_{p\leftarrow r}(h)$ for all r and implement Ca_p by the previous lemma.

It turns out that knowing which process among a set Q has the freshest information about any other process r is exactly what the gossip transformation of [14] does.

Knowledge	additional bits of information	on-the-fly complexity	with precomputation
memoryless	0	PSPACE($\|V\|$)	$O(\|V_p\|)$
perfect recall	$2^{\|V\|}$	EXPTIME($\|V\|$)	$O(\|V\| \cdot 2^{\|V\|})$
causal	$\|P\|^2 \log(\|P\|) \log(\|\mathsf{Tr}\|) + \|V\|$	PSPACE($\|V\|$)	$O(\|V\|)$

Table 1. Complexity of checking knowledge

Roughly speaking, the gossip transformation keeps a partial ordering regarding not only the occurrences $\mathsf{latest}_{p \leftarrow q}$ but also the occurrence $\mathsf{latest}_{p \leftarrow q \leftarrow r}$ (called the tertiary information), which corresponds to the latest occurrence on r before $\mathsf{latest}_{p \leftarrow q}$. Comparing these partial orders from every process $p \in L_t(a)$ involved in the transition a, one can determine who has the latest information on r for every process r [14]. As the number of processes is linear, the number of occurrences of the tertiary information is polynomial.

Keeping the partial order about occurrences of the tertiary information therefore only requires a polynomial number of bits. Notice that [9] (see also [17, 2] for the original timestamping) gives a construction that uses only $O(P^2 \log P)$ bits of memory.

We thus augment the program (the transitions, in our case) with variables that will implement the gossip automata construction, as well as the state $\mathsf{last}_{p \leftarrow q}$ for each process q. That is, the number of bits we need to implement Ca_p is $O(|V| + P^2 \log P)$.

There are again two alternative ways to check for a particular knowledge:

With offline precomputation: We precalculate a table that, for each $state = (\mathsf{last}_{p \leftarrow q})_{q \in P}$, tells whether every global state which can be reached from $state$ by performing only occurrences not on p models φ. If it is the case, then $s \models K_p^C \varphi$ holds. This can be held in a table of $\mathcal{O}(2^n)$ entries. The complexity to check on the fly using the table whether $state$ satisfies the property can then be done in PTIME.

Online construction: If the table is not calculated in advance, we need to perform a search for a global state not satisfying φ and reachable from $state$ using only occurrences not on p. This may takes exponential time (or, alternatively, PSPACE).

The various complexity results we obtained are summed up in Table 1.

6 Conclusions

Knowledge is the foundation for reasoning about the correctness of concurrent systems. It is a prerequisite for enforcing some global coordination with minimal synchronization. While the most basic notion of knowledge, which only depends on the current local state, is essentially an invariant (given the current local state of a process, the global state satisfies some property), knowledge can also be defined based on the observable history: 'knowledge of perfect recall' takes the local observable history of a process into account [3]. We add another notion of knowledge, one that allows not only to memorize local history, but also to update it through communication. We provide a corresponding new definition of knowledge, based on causality.

Knowledge has proven to be useful for the construction of control in concurrent systems [1, 8, 16]: based on the knowledge calculation, the system can be controlled to satisfy additional imposed global properties. Such constructions are monotonic in

the sense that they preserve the knowledge calculated before control was added. When memoryless knowledge is not sufficient, one may need to use constructions that exploit perfect recall or causal knowledge. The view we take in this paper is that using knowledge in this context amounts to a simple transformation of the system. Specifically, the construction we provide here for causal knowledge can be used for supporting such a control construction. It is interesting to observe that causal knowledge is cheaper than knowledge of perfect recall, both in terms of bits to remember and in terms of time complexity. Moreover, causal knowledge is stronger: it refines the knowledge available under perfect recall. However, the transformation, which is required for causal knowledge, is based on the ability to exchange information while performing a joint transition (by the observed or controlled system). If this is not allowed, one may revert to the weaker control through knowledge of perfect recall, where the controller may need to keep an expensive progress table that represents the reachable global states.

References

1. A. Basu, S. Bensalem, D. Peled, J. Sifakis. Priority scheduling of distributed systems based on model checking. *Formal Methods in System Design 39(3)*, pp. 229-245, 2011.
2. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. In particular, Chapter 8 by V. Diekert and A. Muscholl. World Scientific, Singapore, 1995.
3. R. Fagin, J.Y. Halpern, Y. Moses, M. Vardi. Reasoning About Knowledge. MIT Press, Cambridge, 1995.
4. P. Gastin, B. Lerman and M. Zeitoun. Distributed Games with Causal Memory Are Decidable for Series-Parallel Systems. Proc. of *FSTTCS'04*, LNCS 3328, 2004.
5. B. Genest, H. Gimbert, A. Muscholl and I. Walukiewicz. Optimal Zielonka-like Construction. Proc. of *ICALP'10*, LNCS 6199, pp. 52-63, 2010.
6. B. Genest, H. Gimbert, A. Muscholl, I. Walukiewicz. Asynchronous Games over Tree Architectures. Proc. of *ICALP'13*, LNCS 7966, pp. 275-286, 2013.
7. B. Genest and A. Muscholl. Constructing Exponential-size Deterministic Zielonka Automata. Proc. of *ICALP'06*, LNCS 4051, pp. 565-576, 2006.
8. S. Graf, D. Peled, S. Quinton. Monitoring Distributed Systems Using Knowledge. Proc. of *FMOODS/FORTE'11*, LNCS 6722, pp. 183-197, 2011.
9. R. Krishnan, S. Venkatesh. Optimizing the gossip automaton, Report TCS-94-3, School of Mathematics, SPIC Science Foundation, Madras, India, 1994.
10. A. Mazurkiewicz. Concurrent program schemes and their interpretation. Technical report, DAIMI Report PB-78, Aarhus University, 1977.
11. R. van der Meyden. Common Knowledge and Update in Finite Environment. *Information and Computation 140(2)*, pp. 115-157, 1998.
12. R. van der Meyden, N. V. Shilov. Model Checking Knowledge and Time in Systems with Perfect Recall. Proc. of *FSTTCS'99*, pp. 432-445, 1999.
13. A.R.Meyer, L.J. Stockmeyer. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. Proc. of *STOC'73*, pp. 1-9, 1973.
14. M. Mukund, M. Sohoni. Keeping Track of the Latest Gossip in a Distributed System. In *Distr. Computing* 10(3):137-148, 1997.
15. P. Madhusudan, P. S. Thiagarajan, S. Yang. The MSO Theory of Connectedly Communicating Processes. Proc. of *FSTTCS'05*, pp. 201-212, 2005.
16. S. L. Ricker, K. Rudie. Know means no: Incorporating knowledge into discrete-event control systems. *IEEE Trans. Automat. Contr. 45(9)*: 1656-1668, 2000.
17. W. Zielonka. Notes on finite asynchronous automata. In *R.A.I.R.O. - Informatique Théorique et Applications*, 21:99-135, 1987.