# Decompositional Reasoning about the History of Parallel Processes*

Luca Aceto[1], Arnar Birgisson[2], Anna Ingolfsdottir[1], and MohammadReza Mousavi[3]

[1] School of Computer Science, Reykjavik University, Iceland
[2] Department of Computer Science and Engineering, Chalmers University of Technology, Sweden
[3] Department of Computer Science, TU/Eindhoven, The Netherlands

**Abstract.** This paper presents a decomposition technique for Hennessy-Milner logic with past and its extension with recursively defined formulae. In order to highlight the main ideas and technical tools, processes are described using a subset of CCS with parallel composition, nondeterministic choice, action prefixing and the inaction constant. The study focuses on developing decompositional reasoning techniques for parallel contexts in that language.

## 1 Introduction

State-space explosion is a major obstacle in model checking logical properties. One approach to combat this problem is compositional reasoning, where properties of a system as a whole are deduced in a principled fashion from properties of its components. The study of compositional proof systems for various temporal and modal logics has attracted considerable attention in the concurrency-theory literature and several compositional proof systems have been proposed for such logics over (fragments of) process calculi. (See, e.g., [6, 36, 37, 41].) A related line of research is the one devoted to (de)compositional model checking [5, 19, 25, 31, 42]. Decompositional reasoning aims at automatically decomposing the global property to be model checked into local properties of (possibly unknown) components—a technique that is often called *quotienting*. In the context of process algebras, as the language for describing reactive systems, and (extensions of) Hennessy-Milner logic (HML), as the logical specification formalism for describing their properties, decompositional reasoning techniques date back to the seminal work of Larsen and Liu in the 1980's and early 1990's [29, 31], which is further developed in, e.g., [7, 9, 10, 12, 18, 23, 25, 26, 35]. However, we are not aware

* The work of Aceto, Birgisson and Ingolfsdottir has been partially supported by the projects "New Developments in Operational Semantics" (nr. 080039021) and "Metatheory of Algebraic Process Theories" (nr. 100014021) of the Icelandic Research Fund. Birgisson has been further supported by research-student grant nr. 080890008 of the Icelandic Research Fund and by grants from the Swedish research agencies SSF and VR.

of any such decomposition technique that applies to reasoning about the "past". This is particularly interesting in the light of recent developments concerning reversible processes [13, 34] and knowledge representation (epistemic aspects) inside process algebra [14, 20], all of which involve some notion of specification and reasoning about the past. Moreover, a significant body of evidence indicates that being able to reason about the past is useful in program verification [22, 28, 32].

In this paper, we address the problem of developing a decomposition technique for Hennessy-Milner logic with past [16, 17, 27] and for its extension with recursively defined formulae. This way, we obtain a decomposition technique for the modal $\mu$-calculus with past [21, 33]. Apart from its intrinsic interest, the decompositionality results we present in this paper also shed light on the expressiveness of the logics we consider. For example, as shown in, e.g., [2, 3], the closure of a logic with respect to quotienting is closely tied to its ability to express properties that can be tested by performing reachability analysis of processes in the context of so-called test automata. As the language for describing processes, in order to highlight the main ideas and technical tools in our approach, we use a subset of CCS with parallel composition, nondeterministic choice, action prefixing and the inaction constant. Our results, however, extend naturally to other classic parallel composition operators from the realm of process algebra, such as the general one considered in the literature on ACP [8], and to a setting where (possibly infinite) synchronization trees [40] are used as a model of process behaviour.

As the work presented in this paper shows, the development of a theory of decompositional reasoning in a setting with past modalities involves subtleties and design decisions that do not arise in previous work on HML and Kozen's $\mu$-calculus [24]. For instance, the decompositionality result for HML with past and its extension with recursively defined formulae rests on a decomposition of computations of parallel processes into *sets* of pairs of computations of their components, whose concurrent execution might have produced the original parallel computations. Moreover, as explained in detail in the main body of the paper, the presence of past modalities leads us to consider computations of the components of a parallel process that may explicitly include *stuttering steps*—that is, steps where the component under consideration is idle, while a computation step takes place elsewhere in the parallel system. The main results of the paper (Theorems 1 and 2) roughly state that if a computation $\pi$ of a parallel process $p \parallel q$ satisfies a formula $\varphi$ in one of the logics we study then, no matter what decomposition of $\pi$ we pick, the contribution of $p$ to the computation $\pi$ will satisfy the "quotient of $\varphi$ with respect to the contribution of $q$ to $\pi$." Conversely, if there is *some way* of decomposing $\pi$, in such a way that the contribution of $p$ to the computation $\pi$ satisfies the "quotient of $\varphi$ with respect to the contribution of $q$ to $\pi$", then the computation $\pi$ of the parallel process $p \parallel q$ is guaranteed to satisfy $\varphi$.

The rest of this paper is structured as follows. Section 2 introduces preliminary definitions and the extension of Hennessy-Milner logic with past. Section 3 discusses how parallel computations are decomposed into their components. Section 4 presents the decompositional reasoning technique and the first main theorem of the paper. Section 5 extends the theory to recursively defined formulae, and Section 6 discusses related work and possible extensions of our results. Due to space limitation, the proofs of the results are included in the extended version of this paper [1].

## 2 Preliminaries

A *labelled transition system* (LTS) is a triple $\langle P, A, \longrightarrow \rangle$ where

- $P$ is a set of process names,
- $A$ is a finite set of action names, not including a *silent action* $\tau$ (we write $A_\tau$ for $A \cup \{\tau\}$), and
- $\longrightarrow \subseteq P \times A_\tau \times P$ is the *transition relation*; we call its elements *transitions* and usually write $p \xrightarrow{\alpha} p'$ to mean that $(p, \alpha, p') \in \longrightarrow$.

We let $p, q, \ldots$ range over $P$, $a, b, \ldots$ over $A$ and $\alpha, \beta, \ldots$ over $A_\tau$.

For any set $S$, we let $S^*$ be the set of finite sequences of elements from $S$. Concatenation of sequences is represented by juxtaposition. $\lambda$ denotes the empty sequence and $|w|$ stands for the length of a sequence $w$.

Given an LTS $\mathcal{T} = \langle P, A, \longrightarrow \rangle$, we define a *path from* $p_0$ to be a sequence of transitions $p_0 \xrightarrow{\alpha_0} p_1$, $p_1 \xrightarrow{\alpha_1} p_2$, $\ldots$, $p_{n-1} \xrightarrow{\alpha_{n-1}} p_n$ and usually write this as $p_0 \xrightarrow{\alpha_0} p_1 \xrightarrow{\alpha_1} p_2 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{n-1}} p_n$.

We use $\pi, \mu, \ldots$ to range over paths. A *computation from* $p$ is a pair $(p, \pi)$, where $\pi$ is a path from $p$, and we use $\rho, \rho', \ldots$ to range over computations. $\mathcal{C}_\mathcal{T}(p)$, or simply $\mathcal{C}(p)$ when the LTS $\mathcal{T}$ is clear from the context, is the set of computations from $p$ and $\mathcal{C}_\mathcal{T}$ is the set of all computations in $\mathcal{T}$.

For a computation $\rho = (p_0, \pi)$, where $\pi = p_0 \xrightarrow{\alpha_0} p_1 \xrightarrow{\alpha_1} p_2 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{n-1}} p_n$, we define $\mathrm{first}(\rho) = \mathrm{first}(\pi) = p_0$, $\mathrm{last}(\rho) = \mathrm{last}(\pi) = p_n$, and $|\rho| = |\pi| = n$.

Concatenation of computations $\rho$ and $\rho'$ is denoted by their juxtaposition $\rho\rho'$ and is defined iff $\mathrm{last}(\rho) = \mathrm{first}(\rho')$. When $\mathrm{last}(\rho) = p$ we write $\rho(p \xrightarrow{\alpha} q)$ as a shorthand for the slightly longer $\rho(p, p \xrightarrow{\alpha} q)$. We also use $\rho \xrightarrow{\alpha} \rho'$ to denote that there exists a computation $\rho'' = (p, p \xrightarrow{\alpha} p')$, for some processes $p$ and $p'$, such that $\rho' = \rho\rho''$.

**Definition 1 (Hennessy-Milner logic with past).** *Let $\mathcal{T} = \langle P, A, \rightarrow \rangle$ be an LTS. The set $HML_\leftarrow(A)$, or simply $HML_\leftarrow$, of* Hennessy-Milner logic formulae with past *is defined by the following grammar, where $\alpha \in A_\tau$.*

$$\varphi, \psi ::= \top \mid \varphi \wedge \psi \mid \neg\varphi \mid \langle\alpha\rangle\varphi \mid \langle\leftarrow\alpha\rangle\varphi.$$

*We define the* satisfaction relation $\models \subseteq \mathcal{C}_\mathcal{T} \times HML_\leftarrow$ *as the least relation that satisfies the following clauses:*

- $\rho \models \top$ *for all* $\rho \in \mathcal{C}_\mathcal{T}$,

– *$\rho \vDash \varphi \wedge \psi$ iff $\rho \vDash \varphi$ and $\rho \vDash \psi$,*
– *$\rho \vDash \neg\varphi$ iff not $\rho \vDash \varphi$,*
– *$\rho \vDash \langle\alpha\rangle\varphi$ iff $\rho \xrightarrow{\alpha} \rho'$ and $\rho' \vDash \varphi$ for some $\rho' \in \mathcal{C}_{\mathcal{T}}$, and*
– *$\rho \vDash \langle\leftarrow\alpha\rangle\varphi$ iff $\rho' \xrightarrow{\alpha} \rho$ and $\rho' \vDash \varphi$ for some $\rho' \in \mathcal{C}_{\mathcal{T}}$.*

*For a process $p \in P$, we take $p \vDash \varphi$ to mean $(p, \lambda) \vDash \varphi$.*

We make use of some standard short-hands for Hennessy-Milner-type logics, such as $\bot = \neg\top$, $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$, $[\alpha]\varphi = \neg\langle\alpha\rangle(\neg\varphi)$ and $[\leftarrow\alpha]\varphi = \neg\langle\leftarrow\alpha\rangle(\neg\varphi)$. For a finite set of actions $B$, we also use the following notations.

$$\langle\leftarrow B\rangle\varphi = \bigvee_{\alpha\in B} \langle\leftarrow\alpha\rangle\varphi \qquad\qquad [\leftarrow B]\varphi = \bigwedge_{\alpha\in B} [\leftarrow\alpha]\varphi$$

It is worth mentioning that the operators $\langle\cdot\rangle$ and $\langle\leftarrow\cdot\rangle$ are not entirely symmetric. The future is nondeterministic; the past is, however, always deterministic. This is by design, and we could have chosen to model the past as nondeterministic as well, i.e., to take a possibilistic view where we would consider all possible histories. Overall, the deterministic view is more appropriate for our purposes. See, e.g., [28] for a clear discussion of possible approaches in modelling the past and further references.

## 3 Decomposing Computations

In this section, following [5, 25, 31], we aim at defining a notion of "formula quotient with respect to a process in a parallel composition" for formulae in $\mathit{HML}_{\leftarrow}$. In our setting, this goal translates into a theorem of the form $\rho \vDash \varphi$ iff $\rho_1 \vDash \varphi/\rho_2$, where $\rho, \rho_1, \rho_2$ are computations such that $\rho$ is a computation of a "parallel process" that is, in some sense, the "parallel composition" of $\rho_1$ and $\rho_2$.

In the standard setting, definitions of "formula quotients" are based on local information that can be gleaned from the operational semantics of the chosen notion of parallel composition operator. In the case of computations, however, such local information does not suffice. A computation arising from the evolution of two processes run in parallel has the form $(p \parallel q, \pi)$, where $p \parallel q$ is a syntactic representation of the initial state and $\pi$ is the path leading to the current state. The path $\pi$, however, may involve contributions from both of the parallel components. Separating the contributions of the components for the purposes of decompositional model checking requires us to unzip these paths into separate paths that might have been observed by considering only one argument of the composition. This means that we have to find two paths $\pi_p$ and $\pi_q$ such that $(p, \pi_p)$ and $(q, \pi_q)$ are, in some sense, independent computations that run in parallel will yield $(p \parallel q, \pi)$.

*CCS Computations and Their Decomposition* For this study, in order to highlight the main ideas and technical tools in our approach, we restrict ourselves to a subset of CCS, namely CCS without renaming, restriction or recursion. (We discuss possible extensions of our results in Section 6.) Processes are thus defined by the grammar

$$p, q \quad ::= \quad 0 \mid \alpha.p \mid p + q \mid p \parallel q$$

and their operational semantics is given by the following rules.

$$\frac{}{\alpha.p \xrightarrow{\alpha} p} \qquad \frac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{\alpha} q'}{p + q \xrightarrow{a} q'}$$

$$\frac{p \xrightarrow{\alpha} p'}{p \parallel q \xrightarrow{\alpha} p' \parallel q} \qquad \frac{q \xrightarrow{\alpha} q'}{p \parallel q \xrightarrow{\alpha} p \parallel q'} \qquad \frac{p \xrightarrow{a} p' \quad q \xrightarrow{\bar{a}} q'}{p \parallel q \xrightarrow{\tau} p' \parallel q'}$$

We write $p \xrightarrow{\alpha} q$ to denote that this transition is provable by these rules. We assume also that $\bar{\cdot} : A \to A$ is a bijective function on action names such that $\bar{\bar{a}} = a$.

The decomposition of a computation resulting from the evolution of two parallel components must retain the information about the order of steps in the interleaved computation. We do so by modelling the decomposition using *stuttering computations*. These are computations that are not only sequences of transition triplets, but may also involve pseudo-steps labelled with $\dashrightarrow$. Intuitively, $p \dashrightarrow p$ means that process $p$ has remained idle in the last transition performed by a parallel process having $p$ as one of its parallel components. We denote the set of stuttering computations with $\mathcal{C}^*_{\mathcal{T}}$ or simply $\mathcal{C}^*$. For example, the computation $(a.0 \parallel b.0, a.0 \parallel b.0 \xrightarrow{a} 0 \parallel b.0 \xrightarrow{b} 0 \parallel 0)$ is decomposed into the stuttering computations $(a.0, a.0 \xrightarrow{a} 0 \dashrightarrow 0)$ and $(b.0, b.0 \dashrightarrow b.0 \xrightarrow{b} 0)$. However, the decomposition of a parallel computation is not in general unique, as there may be several possibilities stemming from different synchronization patterns. For example consider a computation with path $(a.0 + b.0) \parallel (\bar{a}.0 + \bar{b}.0) \xrightarrow{\tau} 0 \parallel 0$. From this computation it is not possible to distinguish if the transition labelled with $\tau$ was the result of communication of the $a$ and $\bar{a}$ actions, or of the $b$ and $\bar{b}$ actions. We thus consider *all* possibilities simultaneously, i.e., a decomposition of a computation is actually *a set* of pairs of components.

The following function over paths defines the decomposition of a computation.

$$D(\lambda) = \{(\lambda, \lambda)\}$$

$$D(\pi(p \parallel q \dashrightarrow p \parallel q)) = \{(\mu_1(p \dashrightarrow p), \mu_2(q \dashrightarrow q)) \mid (\mu_1, \mu_2) \in D(\pi)\}$$

$$D(\pi(p \parallel q \xrightarrow{\alpha} p' \parallel q')) = \begin{cases} \{(\mu_1(p \xrightarrow{\alpha} p'), \mu_2(q \dashrightarrow q)) \\ \qquad \mid (\mu_1, \mu_2) \in D(\pi)\} & \text{if } q = q' \\ \{(\mu_1(p \dashrightarrow p), \mu_2(q \xrightarrow{\alpha} q')) \\ \qquad \mid (\mu_1, \mu_2) \in D(\pi')\} & \text{if } p = p' \\ \{(\mu_1(p \xrightarrow{a} p'), \mu_2(q \xrightarrow{\bar{a}} q')) \\ \qquad \mid (\mu_1, \mu_2) \in D(\pi), a \in A, \\ \qquad\quad p \xrightarrow{a} p', q \xrightarrow{\bar{a}} q'\} & \text{otherwise} \end{cases}$$

Note that if $(\mu_1, \mu_2)$ is a decomposition of a computation $\pi$, then the three computations have the same length. Furthermore $\mathrm{last}(\pi) = \mathrm{last}(\mu_1) \parallel \mathrm{last}(\mu_2)$.

Another notable property of path decomposition is that it is injective, i.e., a pair $(\mu_1, \mu_2)$ can only be the decomposition of at most one path.

**Lemma 1.** *Let $\pi_1$ be a path of a parallel computation and $(\mu_1, \mu_2) \in D(\pi_1)$. If $\pi_2$ is a path such that $(\mu_1, \mu_2) \in D(\pi_2)$ also, then $\pi_1 = \pi_2$.*

We now aim at defining the quotient of an *HML*$_{\leftarrow}$-formula $\varphi$ with respect to a computation $(q, \mu_2)$, written $\varphi/(q, \mu_2)$, in such a way that a property of the form

$$(p \parallel q, \pi) \vDash \varphi \quad \Leftrightarrow \quad (p, \mu_1) \vDash \varphi/(q, \mu_2)$$

holds when $(\mu_1, \mu_2) \in D(\pi)$. However, since we are dealing with sets of decompositions, we need to quantify over these sets. It turns out that a natural way to do so, which also gives a strong result, is as follows. Given that a composed computation satisfies a formula, we prove in Section 4 that one component of *every* decomposition satisfies a formula quotiented with the other component:

$$(p \parallel q, \pi) \vDash \varphi \;\Rightarrow\; \forall (\mu_1, \mu_2) \in D(\pi) : (p, \mu_1) \vDash \varphi/(q, \mu_2).$$

On the other hand, to show the implication from right to left, we need only one witness of a decomposition that satisfies a quotiented formula to deduce that the composed computation satisfies the original one:

$$\exists (\mu_1, \mu_2) \in D(\pi) : (p, \mu_1) \vDash \varphi/(q, \mu_2) \;\Rightarrow\; (p \parallel q, \pi) \vDash \varphi.$$

In order to define the quotienting transformation, we need a logic that allows us to describe properties of computations involving explicit pseudo-steps. To this end, we now extend *HML*$_{\leftarrow}$ with two additional modal operators.

**Definition 2 (Stuttering Hennessy-Milner logic with past).** *Consider an LTS $\mathcal{T} = \langle P, A, \rightarrow \rangle$. The set $HML^*_{\leftarrow}(A)$, or simply $HML^*_{\leftarrow}$, of* stuttering Hennessy-Milner logic formulae with past *is defined by the grammar*

$$\varphi, \psi ::= \top \;\mid\; \varphi \wedge \psi \;\mid\; \neg\varphi \;\mid\; \langle \alpha \rangle \varphi \;\mid\; \langle \leftarrow \alpha \rangle \varphi \;\mid\; \langle \dashrightarrow \rangle \varphi \;\mid\; \langle \dashleftarrow \rangle \varphi$$

*where $\alpha \in A_\tau$. The* satisfaction relation $\vDash^* \subseteq \mathcal{C}^*_{\mathcal{T}} \times HML^*_{\leftarrow}$ *is defined in the same manner as for Hennessy-Milner logic with past, by extending Definition 1 with the following two items.*

- $\rho \vDash^* \langle \dashrightarrow \rangle \varphi$ *iff* $\rho(p \dashrightarrow p) \vDash^* \varphi$ *where* $p = \mathrm{last}(\rho)$.
- $\rho \vDash^* \langle \dashleftarrow \rangle \varphi$ *iff* $\rho' \vDash^* \varphi$ *where* $\rho = \rho'(p \dashrightarrow p)$ *for some* $p$.

*Similarly,* $\vDash^* \in P \times HML^*_{\leftarrow}$ *is defined by* $p \vDash^* \varphi$ *if and only if* $(p, \lambda) \vDash^* \varphi$.

The satisfaction relations $\vDash^*$ and $\vDash$ coincide over $\mathcal{C}_{\mathcal{T}} \times HML_{\leftarrow}$.

*Why are the pseudo-steps necessary?* One may ask why we need to extend both the computations and the logic to include the notion of pseudo-steps. The reason for doing so is to capture information about the interleaving order in component computations. This in turn is necessary because the original logic can differentiate between different interleavings of parallel processes. For an example, consider the computation $(a.0 \parallel b.0, \pi)$, where $\pi = a.0 \parallel b.0 \xrightarrow{a} 0 \parallel b.0 \xrightarrow{b} 0 \parallel 0$. Clearly this computation does *not* satisfy the formula $\langle\leftarrow a\rangle\top$.

Another interleaving of the same parallel composition is the computation $(a.0 \parallel b.0, \pi')$, where $\pi' = a.0 \parallel b.0 \xrightarrow{b} a.0 \parallel 0 \xrightarrow{a} 0 \parallel 0$. This computation, on the other hand, does satisfy $\langle\leftarrow a\rangle\top$. Since the logic can distinguish between different interleaving orders of a parallel computation, it is vital to maintain information about the interleaving order in our decomposition. If the decomposition of the above computations only considered the actions contributed by each component, this information would be lost and the two paths would have the same decomposition. As a result, we could not reasonably expect to test if they satisfy the formula $\langle\leftarrow a\rangle\top$ in a decompositional manner.

## 4 Decompositional Reasoning

We now define the quotienting construction over formulae structurally. The complete quotienting transformation is given in Table 1. Below we limit ourselves to discussing the quotienting transformation for formulae of the form $\langle\leftarrow\alpha\rangle\varphi$.

To define the transformation for formulae of that form, we examine several cases separately. First we consider the case when $\rho$ has the empty path. In this case it is obvious that no backward step is possible and therefore:

$$(\langle\leftarrow\alpha\rangle\varphi)/(p, \lambda) = \bot.$$

The second case to consider is when $\rho$ ends with a pseudo-transition. In this case the only possibility is that the other component (the one we are testing) is able to perform the backward transition.

$$(\langle\leftarrow\alpha\rangle\varphi)/\rho'(p' \dashrightarrow p') = \langle\leftarrow\alpha\rangle(\varphi/\rho')$$

The third case applies when $\rho$ does indeed end with the transition we look for. In this case the other component must end with a matching pseudo-transition.

$$(\langle\leftarrow\alpha\rangle\varphi)/\rho'(p'' \xrightarrow{\alpha} p') = \langle\dashleftarrow\rangle(\varphi/\rho') \tag{1}$$

The only remaining case to consider is when $\rho$ ends with a transition different from the one we look for. We split this case further and consider again separately the cases when $\alpha \in A$ and when $\alpha = \tau$. The former case is simple: if $\rho$ indicates that the last transition has a label other than the one specified in the diamond operator, the composite computation cannot satisfy $\langle\leftarrow a\rangle\varphi$ because the other component must have performed a pseudo-step.

$$(\langle\leftarrow a\rangle\varphi)/\rho'(p'' \xrightarrow{\beta} p') = \bot \quad \text{where } a \neq \beta$$

$$
\begin{array}{l}
\top/\rho = \top \\
(\varphi_1 \wedge \varphi_2)/\rho = \varphi_1/\rho \wedge \varphi_2/\rho \\
(\neg\varphi)/\rho = \neg(\varphi/\rho) \\
(\langle a\rangle\varphi)/\rho = \langle a\rangle\,(\varphi/\rho(p' \dashrightarrow p')) \vee \left(\bigvee_{\rho' : \rho \xrightarrow{a} \rho'} \langle \dashrightarrow\rangle(\varphi/\rho')\right) \\
(\langle \tau\rangle\varphi)/\rho = \langle \tau\rangle\,(\varphi/\rho(p' \dashrightarrow p')) \vee \left(\bigvee_{\rho' : \rho \xrightarrow{\tau} \rho'} \langle \dashrightarrow\rangle(\varphi/\rho')\right) \vee \left(\bigvee_{\rho', a : \rho \xrightarrow{a} \rho'} \langle \bar{a}\rangle(\varphi/\rho')\right) \\
(\langle \leftarrow\alpha\rangle\varphi)/(p,\lambda) = \bot \\
(\langle \leftarrow\alpha\rangle\varphi)/\rho'(p' \dashrightarrow p') = \langle \leftarrow\alpha\rangle(\varphi/\rho') \\
(\langle \leftarrow\alpha\rangle\varphi)/\rho'(p'' \xrightarrow{\alpha} p') = \langle \dashleftarrow\rangle(\varphi/\rho') \\
(\langle \leftarrow a\rangle\varphi)/\rho'(p'' \xrightarrow{\beta} p') = \bot \quad \text{where } a \neq \beta \\
(\langle \leftarrow\tau\rangle\varphi)/\rho'(p'' \xrightarrow{b} p') = \langle \leftarrow\bar{b}\rangle(\varphi/\rho') \\
(\langle \dashrightarrow\rangle\varphi)/\rho = \langle \dashrightarrow\rangle\,(\varphi/\rho(p' \dashrightarrow p')) \\
(\langle \dashleftarrow\rangle\varphi)/\rho = \begin{cases} \langle \dashleftarrow\rangle(\varphi/\rho') & \text{if } \rho = \rho'(p' \dashrightarrow p') \\ \bot & \text{otherwise} \end{cases}
\end{array}
$$

**Table 1.** Quotienting transformations of formulae in $HML^*_{\leftarrow}$, where $p' = \text{last}(\rho)$

If however the diamond operator mentions a $\tau$ transition, then we must look for a transition in the other component that can synchronise with the last one of $\rho$. Note that this case does not include computations ending with a $\tau$ transition, as that case is covered by Equation (1).

$$(\langle \leftarrow\tau\rangle\varphi)/\rho'(p'' \xrightarrow{b} p') = \langle \leftarrow\bar{b}\rangle(\varphi/\rho')$$

This covers all possible cases for $\langle \leftarrow\alpha\rangle\varphi/\rho$.

We are now ready to prove the main theorem in this section, to the effect that the quotienting of a formula $\varphi$ with respect to a computation $\rho$ is properly defined.

**Theorem 1.** *For CCS processes $p, q$ and a computation $(p \parallel q, \pi) \in \mathcal{C}(p \parallel q)$ and a formula $\varphi \in HML^*_{\leftarrow}$, we have*

$$(p \parallel q, \pi) \models^* \varphi \;\Rightarrow\; \forall(\mu_1,\mu_2) \in D(\pi) : (p,\mu_1) \models^* \varphi/(q,\mu_2) \tag{2}$$

*and, conversely,*

$$(p \parallel q, \pi) \models^* \varphi \;\Leftarrow\; \exists(\mu_1,\mu_2) \in D(\pi) : (p,\mu_1) \models^* \varphi/(q,\mu_2). \tag{3}$$

Theorem 1 uses the existential quantifier in the right-to-left direction. This makes it easy to show that a computation of a process of the form $p \parallel q$ satisfies a formula, given only one witness of a decomposition with one component satisfying the corresponding quotient formula. Note, however, that the set of decompositions of any given process is never empty, i.e., every parallel computation has a decomposition. This allows us to write the above theorem in a more symmetric form.

**Corollary 1.** *For CCS processes $p, q$, a parallel computation $(p \parallel q, \pi)$ and a formula $\varphi \in HML^*_{\leftarrow}$, we have $(p \parallel q, \pi) \models^* \varphi$ iff $(p, \mu_1) \models^* \varphi/(q, \mu_2)$, for each $(\mu_1, \mu_2) \in D(\pi)$.*

## 5 Adding recursion to $HML^*_{\leftarrow}$

In this section, we extend the results from Section 4 to a version of the logic $HML^*_{\leftarrow}$ that includes (formula) variables and a facility for the recursive definition of formulae. Following, e.g., [30], the intended meaning of a formula variable is specified by means of a declaration, i.e., a mapping from variables to formulae, which may themselves contain occurrences of variables. A declaration is nothing but a system of equations over the set of formula variables.

By using the extension of the logic $HML^*_{\leftarrow}$ discussed in this section, we can reason about properties of processes and computations that go beyond one step of lookahead or look-back. For example we can phrase the question "Has the action $\alpha$ ever happened in the past?" as the least model of a suitable recursive logical property.

**Definition 3.** *Let $A$ be a finite set of actions and let $\mathcal{X}$ be a finite set of identifiers. The set $HML^*_{\leftarrow,\mathcal{X}}(A)$, or simply $HML^*_{\leftarrow,\mathcal{X}}$, is defined by the grammar*

$$\varphi, \psi ::= \top \mid \varphi \wedge \psi \mid \neg\varphi \mid \langle\alpha\rangle\varphi \mid \langle\leftarrow\alpha\rangle\varphi \mid \langle\dashrightarrow\rangle\varphi \mid \langle\dashleftarrow\rangle\varphi \mid X$$

*where $X \in \mathcal{X}$. A* declaration *over $\mathcal{X}$ is a function $\mathcal{D} : \mathcal{X} \rightarrow HML^*_{\leftarrow,\mathcal{X}}$, assigning a formula to each variable contained in $\mathcal{X}$, with the restriction that each occurrence of a variable in a formula in the range of $\mathcal{D}$ is* positive*, i.e., any variable is within the scope of an even number of negations.*

When reasoning about recursive formulae, it is technically convenient to define their meaning (i.e., the set of computations that satisfy them) denotationally, because well-definedness of the semantics of recursive formulae relies on Tarski's fixed point theory. This in turn, depends on a notion of *monotone function* with respect to lattice, which is best described by the denotation function and the usual subset ordering on the set of states satisfying a formula. For the sake of clarity, we rephrase Definition 2 in a denotational setting. As it is customary, the following definition makes use of a notion of environment to give meaning to formula variables. An *environment* is a function $\sigma : \mathcal{X} \rightarrow \mathcal{P}(\mathcal{C}^*)$. Intuitively, an environment assigns to each variable the set of computations that are assumed to satisfy it. We write $\mathcal{E}_{\mathcal{X}}$ for the set of environments over the set of (formula) variables $\mathcal{X}$. It is well-known that $\mathcal{E}_{\mathcal{X}}$ is a complete lattice when environments are ordered pointwise using set inclusion.

**Definition 4 (Denotational semantics of $HML^*_{\leftarrow,\mathcal{X}}$).** *Let $\mathcal{T} = \langle P, A, \rightarrow\rangle$ be an LTS. Let $\varphi$ be a $HML^*_{\leftarrow,\mathcal{X}}$ formula and let $\sigma$ be an environment. The* *denotation of $\varphi$ with respect to $\sigma$, written $[\![\varphi]\!]\sigma$, is defined structurally as follows:*

$$\begin{array}{rclrcl}
[\![\top]\!]\sigma & = & \mathcal{C}^*_{\mathcal{T}} & [\![\neg\varphi]\!]\sigma & = & \mathcal{C}^*_{\mathcal{T}} \setminus [\![\varphi]\!]\sigma \\
[\![X]\!]\sigma & = & \sigma(X) & [\![\varphi \wedge \psi]\!]\sigma & = & [\![\varphi]\!]\sigma \cap [\![\psi]\!]\sigma \\
[\![\langle\alpha\rangle\varphi]\!]\sigma & = & \langle\cdot\alpha\cdot\rangle[\![\varphi]\!]\sigma & [\![\langle\leftarrow\alpha\rangle\varphi]\!]\sigma & = & \langle\cdot\leftarrow\alpha\cdot\rangle[\![\varphi]\!]\sigma \\
[\![\langle\dashrightarrow\rangle\varphi]\!]\sigma & = & \langle\cdot\dashrightarrow\cdot\rangle[\![\varphi]\!]\sigma & [\![\langle\dashleftarrow\rangle\varphi]\!]\sigma & = & \langle\cdot\dashleftarrow\cdot\rangle[\![\varphi]\!]\sigma,
\end{array}$$

*where the operators $\langle\cdot\alpha\cdot\rangle, \langle\cdot\leftarrow\alpha\cdot\rangle, \langle\cdot\dashrightarrow\cdot\rangle, \langle\cdot\dashleftarrow\cdot\rangle : \mathcal{P}(\mathcal{C}^*_{\mathcal{T}}) \to \mathcal{P}(\mathcal{C}^*_{\mathcal{T}})$ are defined thus:*

$$\begin{array}{rcl}
\langle\cdot\alpha\cdot\rangle S & = & \{\rho \in \mathcal{C}^*_{\mathcal{T}} \mid \exists\rho' \in S : \rho \xrightarrow{\alpha} \rho'\} \\
\langle\cdot\leftarrow\alpha\cdot\rangle S & = & \{\rho \in \mathcal{C}^*_{\mathcal{T}} \mid \exists\rho' \in S : \rho' \xrightarrow{\alpha} \rho\} \\
\langle\cdot\dashrightarrow\cdot\rangle S & = & \{\rho \in \mathcal{C}^*_{\mathcal{T}} \mid \exists\rho' \in S : \rho \dashrightarrow \rho'\} \quad \textit{and} \\
\langle\cdot\dashleftarrow\cdot\rangle S & = & \{\rho \in \mathcal{C}^*_{\mathcal{T}} \mid \exists\rho' \in S : \rho' \dashrightarrow \rho\}.
\end{array}$$

*The satisfaction relation $\models_\sigma \subseteq \mathcal{C}^*_{\mathcal{T}} \times \mathit{HML}^*_{\leftarrow,\mathcal{X}}$ is defined by*

$$\rho \models_\sigma \varphi \quad \Leftrightarrow \quad \rho \in [\![\varphi]\!]\sigma.$$

It is not hard to see that, for formulae in $\mathit{HML}^*_{\leftarrow}$, the denotational semantics is independent of the chosen environment and is equivalent to the satisfaction relation offered in Definition 2.

The semantics of a declaration $\mathcal{D}$ is given by a *model* for it, namely by an environment $\sigma$ such that $\sigma(X) = [\![\mathcal{D}(X)]\!]\sigma$, for each variable $X \in \mathcal{X}$. For every declaration there may be a variety of models. However, we are usually interested in either the greatest or the least models, since they correspond to safety and liveness properties, respectively. In the light of the positivity restrictions we have placed on the formulae in the range of declarations, each declaration always has least and largest models by Tarski's fixed-point theorem [38]. See, e.g., [4, 30] for details and textbook presentations.

*Decomposition of formulae in $\mathit{HML}^*_{\leftarrow,\mathcal{X}}$* We now turn to the transformation of formulae, so that we can extend Theorem 1 to include formulae from $\mathit{HML}^*_{\leftarrow,\mathcal{X}}$. Our developments in this section are inspired by [23], but the technical details are rather different and more involved.

In Section 4 we defined how a formula $\varphi$ is quotiented with respect to a computation $\rho$. In particular, the quotiented formula $\top/\rho$ is $\top$ for any computation $\rho$. This works well in the non-recursive setting, but there is a hidden assumption that we must expose before tackling recursive formulae. In Theorem 1, the satisfaction relations are actually based on two different transition systems. By way of example, consider the expression on the right-hand side of (2), namely

$$\forall(\mu_1, \mu_2) \in D(\pi) : \ (p, \mu_1) \models \varphi/(q, \mu_2).$$

When establishing this statement, we have implicitly assumed that we are working within the transition system of computations from $p$ that are *compatible* with the computations from $q$—i.e., above, $\mu_1$ really is a path that is the counterpart of $\mu_2$ in a decomposition of the path $\pi$.

Intuitively, the set of computations that satisfy a quotient formula $\varphi/\rho$ is the set of computations that *are compatible with* $\rho$ and whose *composition with* $\rho$ satisfies the formula $\varphi$. However, defining $\top/\rho = \top$ does not match this intuition, if we take the denotational viewpoint of the formula $\top$ on the right-hand side as representing *all* possible computations. In fact, we expect $\top/\rho$ to represent only those computations that are compatible with $\rho$. We formalize the notion of pairs of compatible computations and refine our definition of $\top/\rho$.

**Definition 5.** *Paths $\mu_1$ and $\mu_2$ are* compatible *with each other if and only if they have the same length and one of the following holds if they are non-empty.*

- *If $\mu_1 = \mu_1'(p'' \xrightarrow{\tau} p')$ then $\mu_2 = \mu_2'(q' \dashrightarrow q')$ and $\mu_1'$ and $\mu_2'$ are compatible.*
- *If $\mu_1 = \mu_1'(p'' \xrightarrow{a} p')$ then either $\mu_2 = \mu_2'(q'' \xrightarrow{\bar{a}} q')$ or $\mu_2 = \mu_2'(q' \dashrightarrow q')$; and in both cases $\mu_1'$ and $\mu_2'$ are compatible.*
- *If $\mu_1 = \mu_1'(p'' \dashrightarrow p')$ then either $\mu_2 = \mu_2'(q'' \xrightarrow{\alpha} q')$, for some action $\alpha$, or $\mu_2 = \mu_2'(q' \dashrightarrow q')$; and in both cases $\mu_1'$ and $\mu_2'$ are compatible.*

*We say that two computations are compatible with each other if their paths are compatible.*

We now revise our transformation of the formula $\top$. We want $\top/\rho$ to be a formula that is satisfied by the set of all computations that are compatible with $\rho$. It turns out this can be expressed in $HML^*_{\leftarrow}$ as described below.

**Definition 6.** *Let $\pi$ be a path of transitions in the LTS $\mathcal{T} = \langle P, A, \rightarrow \rangle$. Then the $HML^*_{\leftarrow}$ formula $\top_\pi$ is defined as follows.*

$$\begin{aligned}
\top_\lambda &= [\leftarrow A_\tau]\bot \wedge [\dashleftarrow]\bot \\
\top_{\pi'(p \xrightarrow{\tau} p')} &= \langle \dashleftarrow \rangle \top_{\pi'} \\
\top_{\pi'(p \xrightarrow{a} p')} &= \langle \leftarrow \bar{a} \rangle \top_{\pi'} \vee \langle \dashleftarrow \rangle \top_{\pi'} \\
\top_{\pi'(p \dashrightarrow p')} &= \langle \leftarrow A_\tau \rangle \top_{\pi'} \vee \langle \dashleftarrow \rangle \top_{\pi'}
\end{aligned}$$

Our reader may notice that this is a rewording of Definition 5, and it is easy to see that the computations satisfying $\top_\pi$ are exactly the computations that have paths compatible with $\pi$. Now the revised transformation of $\top$ is

$$\top/(p, \pi) = \top_\pi, \tag{4}$$

which matches our intuition. For the constructs in the logic $HML^*_{\leftarrow}$, we can reuse the transformation defined in Section 4. We therefore limit ourselves to highlighting how to quotient formulae of the form $X$. However, instead of decomposing formulae of this form, we treat the quotient $X/\rho$ as a variable, i.e., we use the set $\mathcal{X} \times \mathcal{C}$ as our set of variables. The intuitive idea of such variables is as follows:

$$(p, \mu_1) \vDash_{\sigma'} X/(q, \mu_2) \Leftrightarrow (p \parallel q, \pi) \vDash_\sigma X \Leftrightarrow (p \parallel q, \pi) \in \sigma(X),$$

where $\sigma$ is an environment for a declaration $\mathcal{D}$ over the variables $\mathcal{X}$, $\sigma'$ is an environment for a declaration $\mathcal{D}'$ over the variables $\mathcal{X} \times \mathcal{C}$, and $(\mu_1, \mu_2) \in D(\pi)$. We explain below the relation between $\mathcal{D}$ and $\mathcal{D}'$ as well as the one between $\sigma$ and $\sigma'$.

Formally, the variables used in quotienting our logic are pairs $(X, \rho) \in \mathcal{X} \times \mathcal{C}$. Formulae of the form $X$ are simply rewritten as $X/\rho = (X, \rho)$, where the $X/\rho$ on the left-hand side denotes the transformation (as in Section 4) and the pair on the right-hand side is the variable in our adapted logic. When there is no risk of ambiguity, we simply use the notation $X/\rho$ to represent *the variable* $(X, \rho)$.

*Transformation of declarations* Generating the transformed declaration $\mathcal{D}'$ from a declaration $\mathcal{D}$ is done as follows:

$$\mathcal{D}'(X/\rho) = \mathcal{D}(X)/\rho. \tag{5}$$

Note that the rewritten formula on the right-hand side may introduce more variables which obtain their values in $\mathcal{D}'$ in the same manner.

*Transformation of environments* The function $\Phi$ maps environments over $\mathcal{X}$ to environments over $\mathcal{X} \times \mathcal{C}$ thus:

$$\begin{aligned}\sigma'(X/(q, \mu_2)) &= \Phi(\sigma)(X/(q, \mu_2)) \\ &= \{(p, \mu_1) \mid (p \parallel q, \pi) \in \sigma(X) \\ &\quad \text{for some } \pi \text{ with } (\mu_1, \mu_2) \in D(\pi)\}.\end{aligned}$$

Our order of business now is to show that if $\sigma$ is the least (respectively, largest) model for a declaration $\mathcal{D}$, then $\sigma'$ is the least (respectively, largest) model for $\mathcal{D}'$ and vice versa. In particular, we show that there is a bijection relating models of $\mathcal{D}$ and models of $\mathcal{D}'$, based on the mapping $\Phi$. First we define its inverse. Consider the function $\Psi$, which maps an environment over $\mathcal{X} \times \mathcal{C}$ to one over $\mathcal{X}$.

$$\Psi(\sigma')(X) = \{(p \parallel q, \pi) \mid \forall (\mu_1, \mu_2) \in D(\pi) : (p, \mu_1) \in \sigma'(X/(q, \mu_2))\}$$

It is not hard to see that $\Phi$ and $\Psi$ are both monotonic.

We now use the model transformation functions $\Phi$ and $\Psi$ to prove an extended version of Theorem 1.

**Theorem 2.** *Let $p, q$ be CCS processes, $(p \parallel q, \pi) \in \mathcal{C}^*(p \parallel q)$. For a formula $\varphi \in HML^*_{\leftarrow,\mathcal{X}}$ and an environment $\sigma$, we have*

$$(p \parallel q, \pi) \models_\sigma \varphi \;\Leftrightarrow \forall (\mu_1, \mu_2) \in D(\pi) : (p, \mu_1) \models_{\Phi(\sigma)} \varphi/(q, \mu_2). \tag{6}$$

*Conversely, for an environment $\sigma'$,*

$$(p \parallel q, \pi) \models_{\Psi(\sigma')} \varphi \;\Leftrightarrow \forall (\mu_1, \mu_2) \in D(\pi) : (p, \mu_1) \models_{\sigma'} \varphi/(q, \mu_2). \tag{7}$$

We can now show that the functions $\Phi$ and $\Psi$ are inverses of each other.

**Lemma 2.** *$\Psi \circ \Phi = \mathrm{id}_{\mathcal{E}_{\mathcal{X}}}$ and $\Phi \circ \Psi = \mathrm{id}_{\mathcal{E}_{\mathcal{X}\times\mathcal{C}}}$.*

This means that $\Phi$ is a bijection between the collections of environments over the variable spaces $\mathcal{X}$ and $\mathcal{X} \times \mathcal{C}$, and $\Psi$ is its inverse. The last theorem of this section establishes soundness of the decompositional reasoning for $HML^{*}_{\leftarrow,\mathcal{X}}$ by showing that $\Phi$ and $\Psi$ preserve models of $\mathcal{D}$ and $\mathcal{D}'$, respectively.

**Theorem 3.** *Let $\mathcal{D}$ be a declaration over $\mathcal{X}$, and let $\mathcal{D}'$ be its companion declaration over $\mathcal{X} \times \mathcal{C}$ defined by (5). If $\sigma$ is a model for $\mathcal{D}$, then $\Phi(\sigma)$ is a model for $\mathcal{D}'$. Moreover, if $\sigma'$ is a model for $\mathcal{D}'$, then $\Psi(\sigma')$ is a model for $\mathcal{D}$.*

Theorem 3 allows us to use decompositional reasoning for $HML^{*}_{\leftarrow,\mathcal{X}}$. Assume, for example, that we want to find the least model for a declaration $\mathcal{D}$. We start by constructing the declaration $\mathcal{D}'$ defined by (5). Next, we find the least model $\sigma'_{\min}$ of $\mathcal{D}'$ using standard fixed-point computations. (See, e.g., [4] for a textbook presentation.) We claim that $\Psi(\sigma'_{\min})$ is the least model of the declaration $\mathcal{D}$. Indeed, let $\sigma$ be any model of $\mathcal{D}$. Then, by the above theorem, $\Phi(\sigma)$ is a model of $\mathcal{D}'$ and thus $\sigma'_{\min} \subseteq \Phi(\sigma)$ holds, where $\subseteq$ is lifted pointwise to environments. Then the monotonicity of $\Psi$ and Lemma 2 ensure that $\Psi(\sigma'_{\min}) \subseteq \Psi(\Phi(\sigma)) = \sigma$. To conclude, note that $\Psi(\sigma'_{\min})$ is a model of $\mathcal{D}$ by the above theorem.

## 6 Extensions and further related work

In this paper, we have developed techniques that allow us to apply decompositional reasoning for history-based computations over CCS and Hennessy-Milner logic with past modalities. Moreover, we extended the decomposition theorem to a recursive extension of that logic. The contribution of this paper can thus be summarized as follows. For each modal formula $\varphi$ (in the $\mu$-calculus with past) and each parallel computation $\pi$, in order to check whether $(p \parallel q, \pi) \models_{\sigma} \varphi$, it is sufficient to check $(p, \mu_1) \models_{\Phi(\sigma)} \varphi/(q, \mu_2)$, where $(\mu_1, \mu_2)$ is a decomposition of $\pi$ and $\varphi/(q, \mu_2)$ is the quotient of $\varphi$ with respect to the component $(q, \mu_2)$. (The implication holds in the other direction, as well; however, the application of this theorem is expected in the aforementioned direction.) In the presentation of the decomposition of computations that is at the heart of our approach, we rely on some specific properties of CCS at the syntactic level, namely to detect which rule of the parallel operator was applied. By tagging a transition with its proof [11, 15], or even just with the last rule used in the proof, we could eliminate this restriction and extend our approach to other languages involving parallel composition. Another possibility is to construct a rule format that guarantees the properties we use at a more general level, inspired by the work of [18]. However, all our results apply without change to CCS parallel composition over (possibly infinite) synchronization trees.

In this work we have only considered contexts built using parallel composition. However, decompositionality results have been shown for the more general setting of *process contexts* [31] and for rule formats [10, 18]. In that work, one considers, for example, a unary context $C[\cdot]$ (a process term with a *hole*) and a process $p$ with which to instantiate the context. A property of the instantiated context $C[p]$ can then be transformed into an equivalent property of $p$, where the transformation depends on $C$. As the state space explosion of model-checking problems is often due to the use of the parallel construct, we consider our approach a useful first step towards a full decomposition result for more general contexts. In general, the decomposition of computations will be more complex for general contexts.

The initial motivation for this work was the application of epistemic logic to behavioural models, following the lines of [14]. We therefore plan to extend our results to logics that include epistemic operators, reasoning about the knowledge of agents observing a running system. This work depends somewhat on the results presented in Section 5.

As we already mentioned in the introduction, there is by now a substantial body of work on temporal and modal logics with past operators. A small sample is given by the papers [21, 27, 39]. Of particular relevance for our work in this paper is the result in [27] to the effect that Hennessy-Milner logic with past modalities can be translated into ordinary Hennessy-Milner logic. That result, however, is only proved for the version of the logic without recursion and does not directly yield a quotienting construction for the logics we consider in this paper.

## References

1. L. Aceto, A. Birgisson, A. Ingolfsdottir, and M.R. Mousavi. Decompositional reasoning about the history of parallel processes. Technical Report CSR-10-17, TU/Eindhoven, 2010.
2. L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen. The power of reachability testing for timed automata. *TCS*, 300(1–3):411–475, 2003.
3. L. Aceto and A. Ingolfsdottir. Testing Hennessy-Milner logic with recursion. In *FoSSaCS'99*, vol. 1578 of *LNCS*, pp. 41–55. Springer, 1999.
4. L. Aceto, A. Ingolfsdottir, K. G. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification.* Cambridge, 2007.
5. H. R. Andersen. Partial model checking (extended abstract). In *LICS'95*, pp. 398–407. IEEE CS, 1995.
6. H. R. Andersen, C. Stirling, and G. Winskel. A compositional proof system for the modal mu-calculus. In *LICS'94*, pp. 144–153. IEEE CS, 1994.
7. A. Arnold, A. Vincent and I. Walukiewicz. Games for synthesis of controllers with partial observation. *TCS*, 303(1):7–34, 2003.
8. J.C.M. Baeten, T. Basten, and M. A. Reniers. *Process Algebra.* Cambrdige, 2009.
9. S. Basu and R. Kumar. Quotient-based control synthesis for non-deterministic plants with mu-calculus specifications. In *IEEE Conference on Decision and Control 2006*, pp. 5463–5468. IEEE, 2006.
10. B. Bloom, W. Fokkink, and R. J. van Glabbeek. Precongruence formats for decorated trace semantics. *ACM Trans. Comput. Log.*, 5(1):26–78, 2004.
11. G. Boudol and I. Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informaticae*, 11(4):433–452, 1988.

12. F. Cassez and F. Laroussinie. Model-checking for hybrid systems by quotienting and constraints solving. In *CAV'00*, vol. 1855 of *LNCS*, pp. 373–388. Springer, 2000.
13. V. Danos and J. Krivine. Reversible communicating systems. In *CONCUR'04*, vol. 3170 of *LNCS*, pp. 292–307. Springer, 2004.
14. F. Dechesne, M. Mousavi, and S. Orzan. Operational and epistemic approaches to protocol analysis: Bridging the gap. In *LPAR'07*, vol. 4790 of *LNCS*, pp. 226–241. Springer, 2007.
15. P. Degano and C. Priami. Proved trees. In *ICALP'92*, vol. 623 of *LNCS*, pp. 629–640. Springer, 1992.
16. R. De Nicola, U. Montanari, and F. W. Vaandrager. Back and forth bisimulations. In *CONCUR 1990*, vol. 458 of *LNCS*, pp. 152–165. Springer, 1990.
17. R. De Nicola and F. W. Vaandrager. Three logics for branching bisimulation. *JACM*, 42(2):458–487, 1995.
18. W. Fokkink, R. J. van Glabbeek, and P. de Wind. Compositionality of Hennessy-Milner logic by structural operational semantics. *TCS*, 354(3):421–440, 2006.
19. D. Giannakopoulou, C. S. Pasareanu, and H. Barringer. Component verification with automatically generated assumptions. *Automated Software Engineering*, 12(3):297–320, 2005.
20. J. Y. Halpern and K. R. O'Neill. Anonymity and information hiding in multiagent systems. *Journal of Computer Security*, 13(3):483–512, 2005.
21. M. Hennessy and C. Stirling. The power of the future perfect in program logics. *I & C*, 67(1-3):23–52, 1985.
22. T. A. Henzinger, O. Kupferman, and S. Qadeer. From pre-historic to post-modern symbolic model checking. *Formal Methods in System Design*, 23(3):303–327, 2003.
23. A. Ingólfsdóttir, J. C. Godskesen, and M. Zeeberg. Fra Hennessy-Milner logik til CCS-processer. Technical report, Aalborg Universitetscenter, 1987.
24. D. Kozen. Results on the propositional mu-calculus. *TCS*, 27:333–354, 1983.
25. F. Laroussinie and K. G. Larsen. Compositional model checking of real time systems. In *CONCUR'95*, vol. 962 of *LNCS*, pp. 27–41. Springer, 1995.
26. F. Laroussinie and K. G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *FORTE'98*, vol. 135 of *IFIP Conference Proceedings*, pp. 439–456. Kluwer, 1998.
27. F. Laroussinie, S. Pinchinat, and P. Schnoebelen. Translations between modal logics of reactive systems. *TCS*, 140(1):53–71, 1995.
28. F. Laroussinie and P. Schnoebelen. Specification in CTL+past for verification in CTL. *I & C*, 156(1):236–263, 2000.
29. K. G. Larsen. *Context-dependent bisimulation between processes.* PhD thesis, University of Edinburgh, 1986.
30. K. G. Larsen. Proof systems for satisfiability in Hennessy–Milner logic with recursion. *TCS*, 72(2–3):265–288, 1990.
31. K. G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. *Journal of Logic and Computation*, 1(6):761–795, 1991.
32. O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Logic of Programs*, vol. 193 of *LNCS*, pp. 196–218. Springer, 1985.
33. M. Nielsen. Reasoning about the past. In *MFCS98*, pp. 117–128, Springer, 1998.
34. I. C. C. Phillips and I. Ulidowski. Reversing algebraic process calculi. *JLAP*, 73(1–2):70–96, 2007.
35. J.-B. Raclet. Residual for component specifications. *Electr. Notes Theor. Comput. Sci.*, 215:93–110, 2008.

36. A. K. Simpson. Sequent calculi for process verification: Hennessy-Milner logic for an arbitrary GSOS. *JLAP*, 60-61:287–322, 2004.
37. C. Stirling. A complete compositional model proof system for a subset of CCS. In *ICALP'85*, vol. 194 of *LNCS*, pp. 475–486. Springer, 1985.
38. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
39. M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP'98*, vol. 1443 of *LNCS*, pp. 628–641. Springer, 1998.
40. G. Winskel. Synchronization trees. *TCS*, 34:33–82, 1984.
41. G. Winskel. A complete proof system for SCCS with modal assertions. *Fundamenta Informaticae*, IX:401–420, 1986.
42. G. Xie and Z. Dang. Testing systems of concurrent black-boxes—an automata-theoretic and decompositional approach. In *FATES'05*, vol. 3997 of *LNCS*, pp. 170–186. Springer, 2006.