Verification Based on Local States *

Michaela Huhn[1] and Peter Niebert[1] and Frank Wallner[2]

[1] Institut für Informatik, Universität Hildesheim,
Postfach 101363, D-31113 Hildesheim, Germany,
{huhn,niebert}@informatik.uni-hildesheim.de

[2] Institut für Informatik, Technische Universität München
D-80290 München, Germany
wallnerf@informatik.tu-muenchen.de

Abstract. Net unfoldings are a well-known partial order semantics for Petri nets. Here we show that they are well suited to act as models for branching-time logics interpreted on *local* states. Such local logics (in particular a *distributed μ-calculus*) can be used to express properties from the point of view of one component in a distributed system. Local logics often allow for more efficient verification procedures because – in contrast to interleaving branching-time logics – they do not refer to the entire space of *global* states. We reduce verification of local properties to standard model checking algorithms known for interleaving branching-time logics. The key is to extract a finite (usually small), local transition system bisimilar to the complete unfolding. The construction is based on the finite prefix of a net unfolding defined by McMillan.

1 Introduction

Model checking is one of the most successful approaches to formal, automated verification of distributed systems. Model checking algorithms decide whether a finite state system meets its specification given in terms of a temporal logic formula. One of the causes of the state explosion problem limiting this approach is the representation of concurrency as interleaving. Recently proposed partial order methods [Pel93, GW91, Val91] avoid the exploration of the entire state space for model checking by reductions according to the *partial order semantics* of the system, where certain interleaving properties are preserved.

Instead of reducing the interleaving model, verification can also be done directly on the partially ordered object: Net unfoldings[3] [NPW80, Eng91] provide a partial order branching time semantics for Petri nets. McMillan [McM92] has shown how to use net unfoldings for *efficient* deadlock detection and reachability analysis of finite-state Petri nets. He described the construction of a "finite prefix" of the (usually infinite) unfolding containing every reachable global state.

* This work was partially supported by: the SFB 342, Teilprojekt A3 of the DFG; and by the Human Capital and Mobility Cooperation Network "EXPRESS" (Expressivity of Languages for Concurrency).

[3] Also known as (branching) non-sequential processes.

It was already observed by Esparza in [Esp94] that the McMillan prefix can be used for model checking S4 (the modal logic based on the reachability relation of the global state space without *next*-modalities).

We show in this paper that a slight modification of McMillan's construction is a very adequate basis for more expressive branching time logics interpreted on *local states* and that for model checking such logics algorithms known from corresponding interleaving logics can be used. Here we understand a local (prime) configuration as the representation of the *view* of a single component onto the system, taking into account that the individual components have only partial information on a system's global state. Local logics allow to express partial order properties of distributed systems in a natural way, while the expression of properties, that refer to a certain interleaving of concurrent events, is neither impossible nor desired. For the linear time case, such logics have been investigated by Thiagarajan in [Thi94, Thi95] and Niebert [Nie95], local branching time logics were introduced in [LT87, LRT92].

We consider systems – described in terms of Petri nets – composed of sequential, nondeterministic subsystems, which synchronously communicate by means of common actions. As a logic we propose a *distributed μ-calculus*, interpreted solely at the local states of the contributing components. The basic operator is an indexed modality $\langle a \rangle_J$ meaning "*next* a for the components $i \in J$". Using fixpoints, local CTL-operators (cf. Sec.3) or the knowledge operator $\Box_i$ from [LRT92] can be encoded. Thus, the distributed μ-calculus serves as a powerful low-level logic, in which other local branching time logics can be expressed.

Besides considerations of its practical use for specification, the proposed logic is designed (i.e. restricted) in order to stay feasible for automatic verification. For good reasons we do not address the theoretical question of the overall expressiveness of the logic: the reference logics for the comparison with μ-calculi are monadic second order logics, but the monadic second order logic of net unfoldings (or prime event structures) can be shown to have a highly undecidable model-checking problem even for 1-safe Petri nets.

The distributed μ-calculus corresponds directly to the sequential μ-calculus [Koz83] interpreted on the *local configurations* of the system's unfolding. Since the (local) state space of the unfolding is in general infinite, our aim is to extract a *bisimilar, finite-state* representation of the unfolding. Such a representation can be immediately used by proved interleaving model checkers [CS93, CES86], yielding efficient automated verification.

We show that for any local configuration of the system's unfolding we find a bisimilar local configuration in the finite prefix – no matter whether we take McMillan's original definition or the improved prefix construction given in [ERV96]. Thus the local configurations within the finite prefix can serve as the *state space* for the desired finite representation. But the proof does not indicate how to determine the *transitions* needed for the finite bisimilar representation without exploring the complete unfolding. The major problem to solve is to determine those transitions (leading to the direct local successors) that are not already present in the finite prefix but which must exist because the local states of the

prefix serve as representatives also for local states in the unfolding far beyond the finite prefix. We show how to find all direct local successors without extending the prefix any further.[4]

Since the resulting local transition system does not contain more states than the prefix contains events, the input for model checkers can be dramatically smaller than the transition system of the global state space. Nevertheless, during the construction of the local transition system we sometimes have to inspect *global* configurations contained in the prefix. Complexity considerations show that the representation of the algorithm given in Sec. 5 can be improved such that it never exceeds the costs of building the global state space times the number of transitions of the original net – which is at the same time the worst case bound of the size of the resulting transition system.

The paper is structured as follows. In Section 2 we introduce basic definitions of our models. In Section 3 we introduce the distributed μ-calculus and its formal semantics, and illustrate its use in specification with examples. In Sections 4 and 5 we show how to use the finite prefix for constructing a finite local transition system on which conventional model checkers apply.

2 Distributed nets and their unfoldings

We begin with the indispensable basic definitions and the class of Petri nets that serve as our system models. For further details on nets, cf. [Rei85].

Petri nets. Let P and T be disjoint, finite sets of *places* and *transitions*, generically called *nodes*. A *net* is a triple $N = (P, T, F)$ with a *flow relation* $F \subseteq (P \times T) \cup (T \times P)$, which we identify with its characteristic function on the set $(P \times T) \cup (T \times P)$. The *preset* ${}^\bullet x$ and the *postset* $x^\bullet$ of the node x are defined as ${}^\bullet x := \{y \in P \cup T \mid F(y,x) = 1\}$ and $x^\bullet := \{y \in P \cup T \mid F(x,y) = 1\}$. The preset (postset) of a set X of nodes is given by the union of the presets (postsets) of all nodes in X. We assume ${}^\bullet x \cup x^\bullet \neq \emptyset$ for every node x.

A *marking* of a net is a mapping $P \to \mathbb{N}$. We call $\Sigma = (N, M_0)$ a *net system* with initial marking M_0 if N is a net and M_0 a marking of N. A marking M *enables* the transition t if $M(p) \geq 1$ for each $p \in {}^\bullet t$. In this case the transition can *occur*, leading to the new marking M', given by $M'(p) = M(p) + F(t,p) - F(p,t)$ for every place p. We denote this occurrence by $M \xrightarrow{t} M'$. If there exists a chain $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} M_n$ then the sequence $\sigma = t_1 t_2 \ldots t_n$ is called *occurrence sequence* and we write $M_0 \xrightarrow{\sigma} M_n$. M is called a *reachable marking of* Σ if there exists an occurrence sequence σ, such that $M_0 \xrightarrow{\sigma} M$. Two transitions t_1, t_2 are *concurrently enabled* in M if M enables t_1, and t_2 is enabled in M', where $M'(p) = M(p) - F(p, t_1)$ for each p. A system is called *sequential* if no reachable marking concurrently enables two transitions.

[4] Since a direct local successor in one component may require an enormous number of causal predecessors in another component, it is not clear in advance when a further extension is sufficient to decide on the existence of a local successor.

We will exclusively regard *1-safe systems*, in which every reachable marking map each place to 0 or 1, and thus can be identified with the set of places it maps to 1, i.e., $M \subseteq P$ for every reachable marking M. Safe net systems can be seen as a synchronization of several finite automata. In the following we will exploit this compositional view by introducing the notion of locations.

Distributed net systems. Let $\{\Sigma_i = (P_i, T_i, F_i, M_i^0) \mid i \in I\}$ be a family of 1-safe, sequential net systems with pairwise disjoint sets P_i of places, indexed by a finite set I of *locations*. The *distributed net system* $\Sigma_I = (N_I, M_0)$ is the union of the *subsystems* Σ_i:

$$P = \bigcup_{i \in I} P_i\,, \quad T = \bigcup_{i \in I} T_i\,, \quad F = \bigcup_{i \in I} F_i\,, \quad M_0 = \bigcup_{i \in I} M_i^0\,.$$

Clearly, Σ_I is again 1-safe. The intended interpretation of such a system is a collection of sequential, non-deterministic processes with communication capabilities, namely the common transitions. We understand the common execution of a joint transition as a communication event. The *location* $loc(x)$ of a node x is defined by $loc(x) := \{i \in I \mid x \in P_i \cup T_i\}$. A simple distributed net system consisting of two subsystems is depicted in Fig. 1.

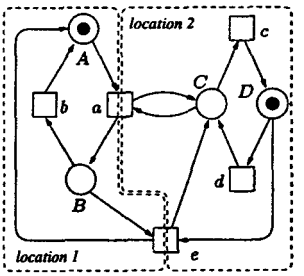


Fig. 1. Distributed net

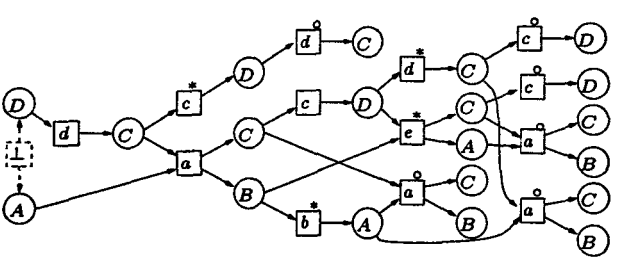


Fig. 2. Branching process

Net unfoldings. In order to define a partial order semantics of the behaviour of a distributed net system, we consider *net unfoldings*, also known as *branching processes*. They contain information about both concurrency and conflict.

Two nodes x_1, x_2 of a net (P, T, F) are *in conflict*, denoted $x_1 \# x_2$, if there exist two distinct transitions t_1, t_2 such that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, and $(t_1, x_1), (t_2, x_2)$ belong to the reflexive and transitive closure of F. If $x \# x$, we say x *is in self-conflict*. An *occurrence net* [NPW80] is a net $N = (B, E, F)$ such that (1) for every $b \in B$, $|{}^\bullet b| \leq 1$, (2) the irreflexive transitive closure $\prec$ of F is well-founded and acyclic, i.e., for every node $x \in B \cup E$, the set $\{y \in B \cup E \mid y \prec x\}$ is finite and does not contain x, and (3) no element $e \in E$ is in self-conflict. The reflexive closure $\preceq$ of $\prec$ determines a partial order, called *causal relation*. In occurrence nets we speak of *conditions* and *events* instead of places and

transitions, respectively. $Min(N)$ denotes the minimal elements of N w.r.t. $\preceq$, and $Max(X)$ the causally maximal elements of the set X of nodes.

Given two nets N_1, N_2, the mapping $h : P_1 \cup T_1 \to P_2 \cup T_2$ is called a *homomorphism* if $h(P_1) \subseteq P_2, h(T_1) \subseteq T_2$, and for every $t \in T_1$ the restriction of h to ${}^\bullet t$, denoted $h|_{{}^\bullet t}$, is a bijection between ${}^\bullet t$ and ${}^\bullet h(t)$, and similar for $h|_{t^\bullet}$.

A *branching process* [Eng91] of a net system $\Sigma = (N, M_0)$ is a pair $\beta = (N', \pi)$ where $N' = (B, E, F)$ is an occurrence net and $\pi : N' \to N$ is a homomorphism, such that the restriction of π to $Min(N')$ is a bijection between $Min(N')$ and M_0 and additionally for all $e_1, e_2 \in E$: if $\pi(e_1) = \pi(e_2)$ and ${}^\bullet e_1 = {}^\bullet e_2$ then $e_1 = e_2$. Loosely speaking, we unfold the net N to an occurrence net N', obeying the rules determined by the conditions for π, and labelling each node x of N' with the corresponding node $\pi(x)$ of N. Referring to distributed net systems, the location $loc(x)$ of a node x of N' is given by $loc(x) = loc(\pi(x))$. By E_J we denote the set of *J-events*, i.e., $E_J := \{e \in E \mid J \subseteq loc(e)\}$. For singleton locations $J = \{i\}$ we abbreviate $E_{\{i\}}$ by E_i.

Given two distinct branching processes β_1, β_2 of Σ, we say that β_1 and β_2 are *isomorphic* if there exists a bijective homomorphism $h : N_1 \to N_2$, such that the composition $\pi_2 \circ h$ equals π_1. If h is injective, such that $h|_{Min(N_1)}$ is a bijection between $Min(N_1)$ and $Min(N_2)$, and furtheron $B_1 \subseteq B_2$ and $E_1 \subseteq E_2$, we call β_1 a *prefix* of β_2. Notice that a prefix is uniquely determined by its set of events *or* its set of conditions. In [Eng91] it is shown that a net system has a unique maximal branching process up to isomorphism, which we call the *unfolding* of Σ, and denote by Unf. Fig. 2 shows a prefix of the infinite unfolding of the net system drawn in Fig. 1.

Configurations and Cuts. A *configuration* C of an occurrence net is a causally downward-closed, conflict-free set of events, i.e., for each $e \in C$: if $e' \preceq e$ then $e' \in C$, and for all $e, e' \in C : \neg(e \# e')$.

If $Max(C)$ is a singleton, say $\{e\}$, we speak of the *local configuration of e* and denote it by $\downarrow e$. It is given by the set of all the preceding events, i.e., $\downarrow e = \{e' \in E \mid e' \preceq e\}$. As usual, we identify each finite configuration C with the state of the system that is reached after all the events in C have occurred. A local configuration then defines a *local state*. The set of local configurations of a branching process β is denoted by $\mathcal{C}_{loc}(\beta)$. In order to simplify the handling, we introduce a virtual event symbol $\perp$ that can be seen as initial event with an empty preset and $Min(N)$ as postset. $\downarrow\perp$ then denotes the empty configuration. We extend the set of events of Unf to $E_\perp := E \cup \{\perp\}$ and set $loc(\perp) = I$.

In distributed systems, we define the *i-view* $\downarrow^i C$ of a configuration C as

$$\downarrow^i C := \{e \in C \mid \exists e' \in (C \cap E_i) : e \preceq e'\}$$

The i-view is a configuration: the empty configuration if $C \cap E_i = \emptyset$, and the local configuration of the (unique) maximal i-event in C, otherwise. This follows from the sequentiality of the subsystems. Thus, $\downarrow^i C$ can be understood as the most recent local state of the subsystem $i \in I$ that the whole system is aware of in the global state C. The i-view of the local configuration $\downarrow e$ is written as $\downarrow^i e$.

Two nodes of an occurrence net are *concurrent* if they are neither in conflict nor causally related. A set B' of conditions of an occurrence net is called a *co-set* if any two elements of B' are concurrent. A co-set is called a *cut* if it is a maximal co-set w.r.t. set inclusion. There is a tight interrelation between finite configurations and cuts: the set of conditions

$$Cut(C) = (Min(N) \cup C^\bullet) \setminus {}^\bullet C$$

where C is a finite configuration, is a cut. The corresponding set of places $\pi(Cut(C))$ is a reachable marking, denoted by $\mathcal{M}(C)$ and called *final state of* C. Notice that for *every* reachable marking M of the system there exist a (not necessarily unique) finite configuration with final state M. Configurations are called *$\mathcal{M}$-equivalent*, denoted by $C =_{\mathcal{M}} C'$, if their final state is equal. Two $\mathcal{M}$-equivalent configurations C, C' have a similar "future", i.e., there exists an isomorphism between the part of *Unf* that lies behind C and that one behind C'. Formally, if $C =_{\mathcal{M}} C'$ then $\beta(C)$ is isomorphic to $\beta(C')$, where $\beta(C) := \{x \in B \cup E \mid \exists b \in Cut(C).\ b \preceq x \ \wedge \ \forall y \in C.\ \neg(x \# y)\}$.

Assume two $\mathcal{M}$-equivalent local configurations $\downarrow e, \downarrow e'$ with $|\downarrow e| < |\downarrow e'|$. The branching process $\beta(\downarrow e)$ can be seen as $\beta(\downarrow e')$ "shifted backward". Any configuration C' containing e' thus can be shifted backward to an $\mathcal{M}$-equivalent configuration C containing e.

In [Esp94] this idea was formalized as follows: let $\mathcal{I}_e^{e'}$ denote the isomorphism from $\beta(\downarrow e')$ to $\beta(\downarrow e)$, and C be a configuration of *Unf*. The *(e', e)-shift of* C, denoted $shift_{(e',e)}(C)$, is defined by

$$shift_{(e',e)}(C) := \begin{cases} C & \text{if } e' \notin C \\ \downarrow e \cup \mathcal{I}_e^{e'}(C \setminus \downarrow e') & \text{if } e' \in C \end{cases}$$

Local successor relation. Let Act be a *distributed alphabet* of actions, i.e., $Act = \bigcup_{i \in I} Act_i$ where the Act_i are not necessarily disjoint. We speak of the location of an action, defined as $loc(a) := \{i \mid a \in Act_i\}$. Assume a mapping l from the transitions (and, via π, also from the events) to the actions that respects the distribution of the alphabet: $l(t) = a$ implies $loc(t) = loc(a)$.

Given two configurations C, C' we call C' an *a-successor* of C, written as $C \xrightarrow{a} C'$, if $C' = C \uplus \{e\}$ for some event e mapped to the action a.

This relation works fine for global configurations, but when considering *local* configurations it turns out to be too restrictive. Intuitively, we want to speak of the *local a-successor* of the local state $\downarrow e$, if for some locations that participated in e the next possible action is an a, ignoring the fact that some other locations possibly have to do some preparing steps until a is enabled. By parameterizing the successor relation with sets of locations, we will determine which of the locations may do those preparing steps, and for which locations the a is the immediate next action.

Let C_1, C_2 be configurations, a an action, and J a non-empty set of locations such that $loc(a) \cap J \neq \emptyset$. We call C_2 a *J-local a-successor* of C_1, written as

$C_1 \xrightarrow{a}_J C_2$, if there exists a configuration $C_1' \supseteq C_1$ such that $\downarrow^i C_1 = \downarrow^i C_1'$ for all $i \in J$, and $C_1' \xrightarrow{a} C_2$.

If $J = \{i\}$ is a singleton, we write $\longrightarrow_i$. Note that $\longrightarrow_i$ captures the local transition relation in an adequate way, i.e., $\longrightarrow_i$ in the unfolding of Σ_I corresponds to the $\longrightarrow$ relation in the unfolding of Σ_i.

3 The distributed μ-calculus

In this section we define the syntax and semantics of a version of the μ-calculus [Koz83] that is adequate to describe *local* properties of the components of a distributed system. More precisely, the formulae of the logic are interpreted over the local configurations of the unfolding of a distributed net system. The logic is adapted from a similar linear time logic for Mazurkiewicz traces [Nie95]. We will indicate how the local approach can be used for the specification and verification of distributed systems, and show that our logic naturally can be transferred to the conventional framework of global states.

Syntax. Let (N_I, M_0) be a distributed net system, $Unf = (N', \pi)$ its unfolding, and $l : T \to Act$ a labelling of the transitions of N_I with *actions* taken from the alphabet Act. We identify the corresponding labelling of events with l, i.e., $l(e) = l(\pi(e))$ for e in Unf. The abstract syntax of the logic is given by

$$\varphi \ ::= \ p \mid \neg p \mid x \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid [a]_J \, \varphi \mid \langle a \rangle_J \, \varphi \mid \mu x.\varphi \mid \nu x.\varphi$$

where the atomic propositions p range over the set P of places of the distributed net, x over a set $\mathcal{V}$ of propositional variables, a over Act, and J over $2^I \setminus \emptyset$. For the modal operators $[a]_J$ and $\langle a \rangle_J$, we assume $J \cap loc(a) \neq \emptyset$. The intended meaning of $\langle a \rangle_J \, \varphi$ is that there exists a *next* local state $\downarrow e$ such that $l(e) = a$ and no event of any of the locations in J will happen before e. The operators μ and ν bind the variables. A formula that does not contain any free variable is *closed*. We use the basic propositions **true** and **false** as abbreviations for $\nu y.y$ and $\mu y.y$, respectively, and define $\langle - \rangle_J \, \varphi := \bigvee_{a \in Act} \langle a \rangle_J \, \varphi$ and $[-]_J \, \varphi := \bigwedge_{a \in Act} [a]_J \, \varphi$.

We only allow negation of atomic propositions. However, the logic is closed under negation, because every operator has its *dual*, and negations can be drawn inside down to the atomic propositions.

Semantics. The semantics of a formula φ of our logic is a set of local configurations (satisfying it), and is written as $[\![\varphi]\!]_v^{Unf} \subseteq \mathcal{C}_{loc}(Unf)$, where Unf is the unfolding under consideration and $v : \mathcal{V} \to 2^{\mathcal{C}_{loc}(Unf)}$ is a valuation function for the variables. Since Unf is clear from the context, we omit this superscript, and if also v is understood, we simply write $[\![\varphi]\!]$. For $\downarrow e \in [\![\varphi]\!]$ we also write $\downarrow e \models \varphi$. We inductively define the semantics according to the following rules:

$$[\![p]\!]_v = \{\downarrow e \mid p \in \mathcal{M}(\downarrow e)\} \qquad [\![\nu x.\varphi]\!]_v = \bigcup \{A \mid A \subseteq [\![\varphi]\!]_{v[x:=A]}\}$$

$$[\![\neg p]\!]_v = \{\downarrow e \mid p \notin \mathcal{M}(\downarrow e)\} \qquad [\![\mu x.\varphi]\!]_v = \bigcap \{A \mid [\![\varphi]\!]_{v[x:=A]} \subseteq A\}$$

$$[\![\varphi \wedge \psi]\!]_v = [\![\varphi]\!]_v \cap [\![\psi]\!]_v \qquad [\![\, [a]_J \, \varphi \,]\!]_v = \{\downarrow e \mid \forall e' \in E \,.\, \text{if } \downarrow e \xrightarrow{a}_J \downarrow e' \text{ then } \downarrow e' \in [\![\varphi]\!]_v\}$$

$$[\![\varphi \vee \psi]\!]_v = [\![\varphi]\!]_v \cup [\![\psi]\!]_v \qquad [\![\, \langle a \rangle_J \, \varphi \,]\!]_v = \{\downarrow e \mid \exists e' \in E \,.\, \downarrow e \xrightarrow{a}_J \downarrow e' \text{ and } \downarrow e' \in [\![\varphi]\!]_v\}$$

where $v[y := A](y) = A$, and for $z \neq y$ we have $v[y := A](z) = v(z)$. We say that system Σ satisfies the formula φ, denoted by $\Sigma \models \varphi$, if the empty configuration $\downarrow\!\perp$ belongs to $[\![\varphi]\!]$.

Note that a local state $\downarrow e$ may satisfy an atomic proposition p that does not belong to the location of e. Thus, the proposed logic allows to express properties corresponding to the local view that one component has onto other components.

We briefly comment on the assertions expressible by the proposed language. Single-located formulae are simply formulae of the standard μ-calculus, interpreted on the corresponding subsystem. For instance, $\Psi = \nu x.\varphi \wedge [\text{-}]_i\, x$ means that on every path of the i-component φ holds at every local state – 'φ always holds in i'. If we substitute $[\text{-}]_i\, x$ by $([\text{-}]_i\, x \wedge \langle\text{-}\rangle_i\, \mathsf{true})$ in Ψ, we additionally express that the mentioned path is of infinite length since for every local state of i there must exist a successor. 'φ holds in i infinitely often' can be formalized as $\nu y.\mu x.(\varphi \vee [\text{-}]_i\, x) \wedge [\text{-}]_i\, y \wedge \langle\text{-}\rangle_i\, \mathsf{true}$. Notice, however, that this formula may hold even if there exist global runs in which the i-component only executes a finite number of events. It actually states that if i executes infinitely many events in the future then it will satisfy φ infinitely often.

It is useful to translate a local logic reminding of CTL [CES86] to our logic. Localised variants of the two next operators, EX_J and AX_J are already part of the syntax, namely $\langle\text{-}\rangle_J$ and $[\text{-}]_J$. The set of locations specifies, for which components this event is a next step. Similarly we now define the until-operators of CTL with locations:

$$\mathsf{E}(\varphi \mathsf{U}_J\, \psi) := \mu y.\psi \vee (\varphi \wedge \langle\text{-}\rangle_J\, y)$$
$$\mathsf{A}(\varphi \mathsf{U}_J\, \psi) := \mu y.\psi \vee (\varphi \wedge [\text{-}]_J\, y \wedge \langle\text{-}\rangle_J\, \mathsf{true}).$$

Other CTL-like operators, such as $\mathsf{AG}_J, \mathsf{AF}_J, \mathsf{EG}_J, \mathsf{EF}_J$ can in turn be defined using the until-operators in the standard way. $\mathsf{E}(\varphi \mathsf{U}_J\, \psi)$ specifies a J-local chain of events along which φ holds until ψ is satisfied.

The more interesting properties, of course, are expressed by formulae referring to distinct subsystems. If $J = \{i, j, k\}$ then $\nu y.[\text{-}]_i\, y \wedge (p \rightarrow \langle a \rangle_J\, \mathsf{true})$ describes that whenever p holds in i then i's next a-action may be a synchronization with j and k, which is also for j and k the next step. Another example referring to several components can be found in the appendix.

It is also possible to refer to conflicts in the causal future of local configurations: $\bigvee_{i \in I}(\langle\text{-}\rangle_i\, p \wedge \langle\text{-}\rangle_i\, \neg p)$ states that there are two next events in conflict which can be distinguished by p. Nevertheless, it is not possible to express that there are two *identically labelled*, but conflicting events if their future cannot be distinguished with the distributed μ-calculus.

As a further example we specify properties of the echo-algorithm as defined in [Wal95] in the distributed μ-calculus. Assume a (strongly connected) network consisting of a set of agents Ag including *initiator* A_0. Each agent A_i communicates exclusively with her direct neighbours, and each agent (but the initiator) behaves identically. At any time the initiator wants to flood the whole network with a *wake-up* signal, each agent – after receiving a wake-up – executes a local computation and sends back an *accept* signal afterwards. Whenever the initiator

reaches state *terminated*, she wants to be sure that every agent in the network has executed her local computation: $\Sigma \models \mathsf{AG}_0(\mathit{terminated} \rightarrow \bigwedge_{i \geq 1} \mathit{accepted}_i)$.

Furthermore, no agent shall have finished her local computation, when any of her neighbours is still sleeping: $\Sigma \models \bigwedge_{i \geq 1} \mathsf{AG}_i(\mathit{accepted}_i \rightarrow \bigwedge_{j \in N_i} \neg \mathit{sleeping}_j)$.

4 Transition systems semantics

Now we want to show that the unfolding can be understood as a local transition system $\mathcal{T}_{Unf}$ with transitions labelled by indexed actions a_J, $J \subseteq I$, and with the *local* configurations of *Unf* as set of states. It will be immediate that on $\mathcal{T}_{Unf}$ the distributed μ-calculus corresponds to the standard μ-calculus over the modified action alphabet $\widetilde{Act} = \{a_J \mid a \in Act, J \subseteq I\}$.

μ-calculus and bisimulation. The syntax of the μ-calculus [Koz83] is given by

$$\varphi \ ::= \ p \mid \neg p \mid x \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu x.\varphi \mid \nu x.\varphi$$

where $p \in P$, $x \in \mathcal{V}$, and $a \in Act_{\mathcal{T}}$. The semantics of the μ-calculus is defined over transition systems $\mathcal{T} = \langle S, s_0, \rightarrow, Act_{\mathcal{T}}, P, \mathrm{I} \rangle$ where S is a set of states, $Act_{\mathcal{T}}$ an action alphabet, $s_0 \in S$ the initial state, $\rightarrow \subseteq S \times Act_{\mathcal{T}} \times S$ the transition relation, and $\mathrm{I} : S \rightarrow \mathbf{2}^P$ an interpretation mapping the states onto the propositions. As usual, we write $s \xrightarrow{a} s'$ if $(s, a, s') \in \rightarrow$.

The semantics of a μ-calculus formula φ over a given transition system $\mathcal{T}$ is denoted by $[\![\varphi]\!]^{\mathcal{T}}_v \subseteq S$, where v is the valuation function for the variables. We write $s \models_{\mathcal{T}} \varphi$ if $s \in [\![\varphi]\!]_v$. The semantics is defined inductively by:

$$\begin{array}{ll}
[\![p]\!]_v = \{s \mid p \in \mathrm{I}(s)\} & [\![\nu x.\varphi]\!]_v = \bigcup\{A \mid A \subseteq [\![\varphi]\!]_{v[x:=A]}\} \\
[\![\neg p]\!]_v = \{s \mid p \notin \mathrm{I}(s)\} & [\![\mu x.\varphi]\!]_v = \bigcap\{A \mid [\![\varphi]\!]_{v[x:=A]} \subseteq A\} \\
[\![\varphi \wedge \chi]\!]_v = [\![\varphi]\!]_v \cap [\![\chi]\!]_v & [\![\langle a \rangle \varphi]\!]_v = \{s \mid \exists s' \in S.\ s \xrightarrow{a} s' \text{ and } s' \in [\![\varphi]\!]_v\} \\
[\![\varphi \vee \chi]\!]_v = [\![\varphi]\!]_v \cup [\![\chi]\!]_v & [\![[a]\varphi]\!]_v = \{s \mid \forall s' \in S.\ \text{if } s \xrightarrow{a} s' \text{ then } s' \in [\![\varphi]\!]_v\}
\end{array}$$

It is well-known that the distinguishing power of the μ-calculus is limited to standard bisimulation: A relation $\mathcal{R} \subseteq S \times S$ is called a *bisimulation* iff for any $s \mathcal{R} s'$ it holds that $\mathrm{I}(s) = \mathrm{I}(s')$ and for all $a \in Act_{\mathcal{T}}$

- if $s \xrightarrow{a} s_1$, then there exists s'_1 with $s' \xrightarrow{a} s'_1$ and $s_1 \mathcal{R} s'_1$, and dually
- if $s' \xrightarrow{a} s'_1$, then there exists s_1 with $s \xrightarrow{a} s_1$ and $s_1 \mathcal{R} s'_1$.

Two states s and s' are called *bisimilar*, denoted $s \sim s'$, iff there exists a bisimulation $\mathcal{R}$ with $s \mathcal{R} s'$. We also write $\mathcal{T} \sim \mathcal{T}'$ if for the initial states $s_0 \sim s'_0$. It was shown by Milner [Mil89] (see also [Sti92]) that $s \sim s'$ implies $s \models_{\mathcal{T}} \varphi \Leftrightarrow s' \models_{\mathcal{T}} \varphi$ for all closed μ-calculus formulae φ.

The local transition system $\mathcal{T}_{Unf}$. Let *Unf* be the unfolding of a distributed net system Σ. Then the local transition system extracted from *Unf* is given by $\mathcal{T}_{Unf} = \langle \mathcal{C}_{loc}(\mathit{Unf}), \downarrow\!\perp, \rightarrow, \widetilde{Act}, P, \mathrm{I} \rangle$ where $\downarrow\!e \xrightarrow{a_J} \downarrow\!e'$ iff $\downarrow\!e \xrightarrow{a}_J \downarrow\!e'$, and the interpretation of propositions $\mathrm{I}(\downarrow\!e) = \mathcal{M}(\downarrow\!e)$ for all $\downarrow\!e$.
Two events e_1, e_2 are *$\mathcal{M}$-loc-equivalent* iff $\downarrow\!e_1 =_{\mathcal{M}} \downarrow\!e_2$ and $\mathit{loc}(e_1) = \mathit{loc}(e_2)$.

Proposition 1. *Let* $\downarrow e_1, \downarrow e_2 \in \mathcal{T}_{Unf}$. *If* e_1 *and* e_2 *are* $\mathcal{M}$*-loc-equivalent then* $\downarrow e_1 \sim \downarrow e_2$.

Proof. Let $\mathcal{I}$ be the isomorphism from $\beta(\downarrow e_1)$ to $\beta(\downarrow e_2)$, induced by $\mathcal{M}$-equivalence of $\downarrow e_1$ and $\downarrow e_2$. Clearly, $loc(f) = loc(\mathcal{I}(f))$, and $f \preceq g$ iff $\mathcal{I}(f) \preceq \mathcal{I}(g)$ for all events $f, g \in \beta(\downarrow e_1)$. If furtheron $e_1 \prec f$ and $e_2 \prec \mathcal{I}(f)$, the events f and $\mathcal{I}(f)$ again are $\mathcal{M}$-equivalent, and thus $=_{\mathcal{M}}$ would be a bisimulation. However, it does not necessarily hold that $e_1 \prec f$ iff $e_2 \prec \mathcal{I}(f)$.

The additional *loc*-condition now preserves the desired causality: Let us call e' a *direct successor* of e, iff $e^\bullet \cap {}^\bullet e' \neq \emptyset$. For all e_1, e_1' it holds that if e_1' is a direct successor of e_1 then $loc(e_1) \cap loc(e_1') \neq \emptyset$. Consequently, if e_1' is a direct successor of e_1 then $\mathcal{I}(e_1') \in \beta(\downarrow e_2)$ is a direct successor of e_2 iff $loc(e_2) \cap loc(\mathcal{I}(e_1')) \neq \emptyset$. Thus the set of direct successors is preserved under $\mathcal{M}$-*loc*-equivalence. Since every J-local successor of an event e_1 is a direct successor of e_1 or the J-local successor of a direct successor of e_1, and since every direct successor of e_1 is $\mathcal{M}$-*loc*-equivalent to the corresponding direct successor of e_2, indeed $\mathcal{M}$-*loc*-equivalence is a bisimulation. □

Let φ be a formula of the distributed μ-calculus. Then $\tilde{\varphi}$ denotes the formula where each occurrence of $\langle a \rangle_J$ is substituted by $\langle a_J \rangle$, and similarly $[a]_J$ by $[a_J]$.

Proposition 2. $\downarrow e \models \varphi$ *iff* $\downarrow e \models_{\mathcal{T}} \tilde{\varphi}$ *for any* $\downarrow e \in C_{loc}(Unf)$.

5 Model checking

In this section we develop the technical tools required to achieve efficient verification techniques for the logic. In fact we will not give an algorithm for the model checking procedure itself. Rather we give a construction, which *reduces* the model checking problem for the distributed μ-calculus to a suitable input for well understood algorithms known from sequential model checking like [CES86, CS93].

As a first step, we will show that there exists a *finite* transition system $\mathcal{T}_{Fin}$ bisimilar to the usually infinite system $\mathcal{T}_{Unf}$. This finite system $\mathcal{T}_{Fin}$ can be defined over the set of local configurations of the *complete finite prefix* introduced by McMillan [McM92]. Secondly, we give an algorithm for constructing $\mathcal{T}_{Fin}$.

The finite prefix. In [McM92], McMillan showed how to construct a finite prefix of the unfolding of a finite-state net system in which every reachable marking is represented by some cut. We will use a slight modification of this prefix to obtain a finite transition system with local states, bisimilar to $\mathcal{T}_{Unf}$.

Let $Unf = (N', \pi)$ be the unfolding of a net system. A *cut-off event* is an event $e \in E_\perp$ whose local configuration's final state coincides with the final state of a smaller local configuration with the same location, formally:

$$\exists\, e' \in E_\perp : |\downarrow e'| < |\downarrow e| \text{ and } e, e' \text{ are } \mathcal{M}\text{-}loc\text{-equivalent.}$$

In McMillan's standard definition $\mathcal{M}$-equivalence suffices.[5] Notice that in general for each cut-off event e there may be several corresponding $\mathcal{M}$-*loc*-equivalent events e'. In the sequel, we fix one of them and refer to it as $corr(e)$. The prefix Fin is then defined as the unique prefix of Unf with $E_{Fin} \subseteq E_\perp$ as set of events, where E_{Fin} is characterised by

$$e \in E_{Fin} \quad \text{iff} \quad \text{no event } e' \prec e \text{ is a cut-off event.}$$

It is easy to prove that Fin is finite for net systems with finitely many reachable markings. Usually, the prefix Fin is much smaller than the state space of the system. However, it can also be larger. In [ERV96] it is shown how to improve McMillan's construction such that the finite prefix never exceeds the size of the full state space (up to a constant). The main idea is to determine cut-off events not by comparing the size of the local configurations of events (which does not produce any cut-off event when the sizes are equal), but other well-founded partial orders instead. In the prefix generated by the refined algorithm, if e and e' are two different non-cut-off events, then they are not $\mathcal{M}$-*loc*-equivalent.

The number of location sets occurring at events can grossly be bounded by the number of transitions in the original net. Therefore, the number of non-cut-off events never exceeds the number of reachable states times the number of transitions of the original net, and so the prefix can never be substantially larger than the state space.

The finite, local transition system $\mathcal{T}_{Fin}$. Now we show that there exists a finite transition system $\mathcal{T}_{Fin} \sim \mathcal{T}_{Unf}$, such that the states of $\mathcal{T}_{Fin}$ are at most the local configurations of the finite prefix.

Observe that the modified McMillan construction in fact guarantees that for each local configuration $\downarrow e$ in Unf there exists an $\mathcal{M}$-*loc*-equivalent corresponding configuration $\downarrow e'$ in Fin, i.e., $\downarrow e \sim \downarrow e'$ in $\mathcal{T}_{Unf}$, and $e' \in E_{Fin}$. The only reason for $e \notin E_{Fin}$ can be that e supersedes a cut-off belonging to Fin and therefore itself is a cut-off. By induction it is possible to find a corresponding event for e within E_{Fin}.

For the following, we select for each equivalence class of bisimilar configurations $\downarrow e$ in Unf a unique representative $\downarrow corr(e)$ in Fin which is minimal w.r.t. the size of $|\downarrow corr(e)|$. (In case of using the improved prefix [ERV96], $corr(e)$ is selected from the non-cut-off events in E_{Fin} and thus uniquely determined.)

If we have two bisimilar states $\downarrow e_1 \sim \downarrow e_2$ in $\mathcal{T}_{Unf}$ we can replace each transition $\downarrow e \xrightarrow{a_J} \downarrow e_1$ by $\downarrow e \xrightarrow{a_J} \downarrow e_2$ for any source state $\downarrow e$ and obtain a transition system $\mathcal{T}'_{Unf}$ bisimilar with $\mathcal{T}_{Unf}$ on all states and with the same state space.

Since we have selected for all local configurations $\downarrow e$, $\downarrow e'$ bisimilar representatives $\downarrow corr(e)$ and $\downarrow corr(e')$ in Fin, we can (imaginarily) "bend" all the transitions $\downarrow e \xrightarrow{a_J} \downarrow e'$ in $\mathcal{T}_{Unf}$ to transitions $\downarrow corr(e) \xrightarrow{a_J} \downarrow corr(e')$ in Fin (possibly merging infinitely many transitions into one). Since $corr(e)$ is unique and

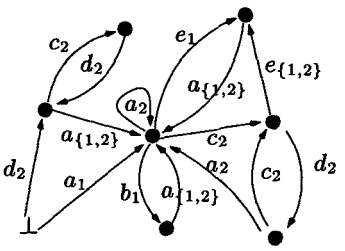
[5] The cut-off events from the unfolding of the distributed net system of Fig. 1 are tagged by "o" in Fig. 2 whereas the cut-offs due to McMillan's original definition are tagged by "*".

minimal, afterwards all local states $\downarrow e$ reachable from $\downarrow\bot$ via transitions are non-cut-offs. Now we discard all cut-off events (whether contained in *Fin* or not). We call the resulting transition system $\mathcal{T}_{Fin}$. Observe that $\mathcal{T}_{Fin}$ is even smaller than *Fin* itself, since cut-offs are discarded. We obtain:

Proposition 3. $\mathcal{T}_{Fin} \sim \mathcal{T}_{Unf}$.

Theorem 4. *For any closed formula φ of the distributed μ-calculus it holds that $\downarrow\bot \models \varphi$ iff $\downarrow\bot \models_{\mathcal{T}_{Fin}} \tilde{\varphi}$.*

Theorem 4 is an immediate consequence of Proposition 2 and Proposition 3. Thus we can reduce the model checking problem of the distributed μ-calculus for some distributed net system to the model-checking problem of the standard μ-calculus over $\mathcal{T}_{Fin}$. Observe that $\mathcal{T}_{Fin}$ is not bigger than the global state space (i.e. the product of the local state spaces) times the number of transitions of the distributed net system – and often much smaller.



The figure depicts $\mathcal{T}_{Fin}$ of the prefix drawn in Fig. 2.

An algorithm to compute $\mathcal{T}_{Fin}$. By now we know that $\mathcal{T}_{Fin}$ exists, the question remains, how we can compute it. We propose an algorithm that takes *Fin* as input and moreover uses the structural information, which the algorithm computing *Fin* has built up:

- a function *corr* mapping all $\mathcal{M}$-*loc*-equivalent events onto a unique representative non-cut-off event. The codomain of corr is called $E_{Rep} \subset E_{Fin}$, the set of representative events. The state space of $\mathcal{T}_{Fin}$ is formed by the local configurations of these representatives.
- a function *shift**, which maps any configuration $C = C_1$ of *Unf* containing some cut-off to a configuration $\mathit{shift}^*(C) = C' = C_n$ not containing a cut-off, hence being present in *Fin*. This function works by repeatedly applying $C_{i+1} := \mathit{shift}_{(e_i, corr(e_i))}(C_i)$ with e_i being a cut-off in *Fin* contained in C_i. *shift** terminates, because the sequence $C_1, C_2, ..$ decreases in the underlying (well-founded) order (e.g. contains less and less events in the case of the McMillan order). Obviously this function implies the existence of an isomorphism $\mathcal{I}$ between $\beta(C)$ and $\beta(\mathit{shift}^*(C))$, which is the composition of the isomorphisms $\mathcal{I}^{e_i}_{corr(e_i)}$ induced by the chosen cut-off events. Moreover, $\mathit{shift}^*(\downarrow e)$ is strictly smaller than $\downarrow e$ (in the underlying order) for any $e \in \beta(C)$, and hence for any e, for which $C \xrightarrow{a}_J \downarrow e$.

The most important part of the algorithm is the recursive procedure *successors* which, when called *from the top level* with a triple $(\downarrow e, J, a)$, returns the a_J-successors for $\downarrow e$ in $\mathcal{T}_{Fin}$. More generally, *successors* performs depth-first search through triples (C, J, a), where C is an arbitrary, not necessarily local configuration not containing a cut-off, J is a non-empty subset of locations, and a is

```
type Vertex = { C: Configuration; J: LocationSet; a: ActionLabel;
                pathmark: bool ; (* for depth first search *) }

prefix_successors(C, J, a) = {↓corr(e) | e ∈ E_Fin ∧ C -a->_J ↓e}

inheritable_extension(C, e, J, a) = (∀i ∈ J. (↓e \ C) ∩ E_i = ∅)
(* predicate ensuring, that joining ↓e to C adds no i-events for i ∈ J *)

compatible_cutoffs(C) = {e | e is cut-off and ↓e ∪ C is a configuration in Fin}

proc successors(C, J, a): ConfigurationSet;
{ var result: ConfigurationSet; (* result accumulator for the current vertex *)
  Vertex v := findvertex(C,J,a); (* lookup in hash table, if not found then *)
                                 (* create new vertex with pathmark = false *)
  if v.pathmark then return ∅; fi (* we have closed a cycle *)
  result := prefix_successors(C, J, a); (* directly accessible successors *)
  v.pathmark:=true; (* put vertex on path *)
  for e_c ∈ compatible_cutoffs(C) do
      (* find successors outside the prefix behind e_c *)
      if inheritable_extension(C, e_c, J, a) then
        result := result ∪ successors(shift*(C ∪ ↓e_c), J, a);
      fi
  od ;
  v.pathmark:=false; (* take vertex from path *)
  return result;
}

proc ComputeT_Fin;
{ InitializeTransitionSystem(ts,Fin) (* extract state space from Fin *)
  for e ∈ E_Rep, a ∈ Act, ∅ ≠ J ⊆ I do
      for ↓e' ∈ successors(↓e,J,a) do
          add transition ↓e -a_J->↓e'
      od
  od
}
```

Fig. 3. The conceptual algorithm to compute $\mathcal{T}_{Fin}$

an action. It determines the subset of events in E_{Rep} that represent the J-local a-successors of C. Formally, $e \in successors(C, J, a)$ iff there exists $\downarrow e'$ in Unf, which is $\mathcal{M}$-loc-equivalent to $\downarrow e$, and $C \xrightarrow{a}_J \downarrow e'$.

The procedure works as follows. Assume there exists at least one e' anywhere in Unf with $C \xrightarrow{a}_J \downarrow e'$; then there are two possibilities:
(1) One of these e' lies in the prefix. This is easy to determine. The corresponding event $corr(e') \in E_{Rep}$ is given back by $prefix\_successors(C, J, a)$.
(2) There exist such events e', but none of them lies in the prefix. The reason for $e' \notin E_{Fin}$ is the existence of a cut-off $e_c \in E_{Fin}$, such that $e_c \preceq e'$. So we can

do a case analysis over the *compatible* cut-offs. A cut-off e_c is compatible with C if it is not in conflict with C, i.e., $\downarrow e_c \cup C$ is a configuration in *Fin*. If there is a compatible e_c, such that $(\downarrow e_c \setminus C) \cap E_i = \emptyset$ for all $i \in J$ then for at least one of them, we have $(C \cup \downarrow e_c) \xrightarrow{a}_J \downarrow e'$. In this case we inherit the transition $C \xrightarrow{a}_J \downarrow e'$.

In the second case, we loop over all compatible cut-offs e_c looking at the configuration $C_c := C \cup \downarrow e_c$. If the J-local a-successors of C_c are J-local a-successors of C (determined by *inheritable_extension*(C, e_c, J, a)) we want to search for the *successors*(C_c, J, a). But if any J-local a-successor e' of C_c exists, then there also exists a bisimilar e'' for $C^* := \mathit{shift}^*(C_c)$ (by the isomorphism), where moreover $\downarrow e''$ is smaller than $\downarrow e'$. So *successors* is recursively called with (C^*, J, a). Note that C^* contains no cut-off.

Hence we apply depth-first search with respect to triples (C, J, a). Cycles may occur (if we hit a triple (C, J, a) with pathmark= true), at which we break off to ensure termination. Note that the search space is limited by the fact that C is represented in *Fin* and does not contain cut-offs.

It remains to show that the termination is correct: Assume a J-local a-successor e' of C exists. Then we choose from all these suitable successors a minimal one named $e_{\min}$. Whenever a configuration $(C \cup \downarrow e_c)$ is shifted with shift^* to obtain a configuration C' for the next call of *successors*, also $e_{\min}$ is shifted to a strictly smaller $e'_{\min}$. Thus in case we hit a configuration C twice, when searching for J-local a-successors, $e_{\min}$ is mapped by the various shift^*s to a strictly smaller event $e^*_{\min}$ which contradicts the minimality of $e_{\min}$. Thus whenever a configuration is investigated a second time for J-local a-successors, we know that there cannot be one.

The main procedure *Compute*$\mathcal{T}_{Fin}$ thus only has to loop about all possible triples $(\downarrow e, J, a)$ with $e \in E_{Rep}$ to check for transitions $\downarrow e \xrightarrow{a_J} \downarrow e'$ in $\mathcal{T}_{Fin}$ and to insert the results of *successors*. Concluding the above discussion, we obtain:

Theorem 5. *The algorithm Compute*$\mathcal{T}_{Fin}$ *computes* $\mathcal{T}_{Fin}$.

Note that at top level, *successors* is only called with local configurations C, but the extension of C with cut-offs requires that we can also handle some global configurations. Further note that we present the algorithm in Fig. 3 with emphasis on understandability, not efficiency: many vertices (C, J, a) will be explored very often, leading to an unsatisfying runtime. However it is very easy to modify the algorithm so that every vertex is explored *at most once*, essentially by storing intermediate results with the vertices in the hash-table. Then the runtime of the algorithm is proportional to the size of the search space. Since we have to deal with some global configurations, in principle the search space can grow to the size of the global state space times the number of the transitions of the original net, but no larger.

Experiments suggest that in many cases the number of *visited* global states will remain small compared to the number of *all* global states existing.

Heuristic improvements. Apart of the improvements mentioned above, the algorithm also allows for several *heuristic improvements* to save unnecessary

computation. For instance, it is impossible that a state $\downarrow e$ has any a_J-successor if the J-places in $\mathcal{M}(\downarrow e)$ are not contained in ${}^\bullet t$ for any a-labelled transition t of the original net, and thus $successors(\downarrow e, J, a)$ need not to be called. Moreover, the algorithm can be combined with *on-the-fly* algorithms (sometimes called *local model checking*), by only calling *successors*, when the model checker needs to find the a_J-successors of some state.

6 Conclusion

We introduced a distributed version of the μ-calculus and showed its use in describing branching time properties of distributed algorithms based on local states. We reduced the model checking problem for the distributed μ-calculus to the well-investigated model checking problem of sequential logics over transition systems.

How expensive is all this? The *computation* of $\mathcal{T}_{Fin}$ can be as costly as generating the global state space (although often it will be much cheaper), the *resulting system* $\mathcal{T}_{Fin}$ is typically much smaller than the global transition system. The transformation of the formulae is for free. So the cost of computing $\mathcal{T}_{Fin}$ does not affect the runtime of the standard model checker in the next phase. It is necessary to investigate really meaningful examples, to give a concise answer on the benefits of the proposed approach. First experiments with a prototype implementation indicate that $\mathcal{T}_{Fin}$ and the number of global states visited during its computation indeed are very small.

Independently, Penczek [Pen97] suggested a model checker for a future fragment of his event structure logic DESL. Instead of using net unfoldings, Penczek relies on partial order methods in the generation of a finite representation of the event structure. The causal future operators of DESL as considered in [Pen97] can easily be treated by the algorithm we proposed here by changing the modalities and the successor relation accordingly. At the price of the restriction to free-choice systems, in [Pen97] also an *immediate conflict* operator is handled.

Acknowledgment. We thank P.S. Thiagarajan for discussions on location based logics. Burkhard Graves has helped our understanding of the subtleties of *Fin*. Special thanks to Javier Esparza, whose contribution to this work in its initial phase was very important.

References

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla, *Automatic verification of finite-state concurrent systems using temporal logic specifications*, ACM Transactions on Programming Languages and Systems **8** (1986), no. 2, 244–263.

[CS93] Rance Cleaveland and Bernhard Steffen, *A linear time model-checking algorithm for the alternation-free modal mu-calculus*, Formal Methods in system Design **2** (1993), 121– 147.

[Eng91] J. Engelfriet, *Branching processes of Petri nets*, Acta Informatica **28** (1991), 575–591.

[ERV96] J. Esparza, S. Römer, and W. Vogler, *An Improvement of McMillan's Unfolding Algorithm*, Tools and Algorithms for the Construction and Analysis of Systems TACAS '96 (Passau, Germany) (T. Margaria and B. Steffen, eds.), LNCS, vol. 1055, Springer, 1996, pp. 87–106.

[Esp94] J. Esparza, *Model checking using net unfoldings*, Science of Computer Programming **23** (1994), 151–195.

[GW91] P. Godefroid and P. Wolper, *A Partial Approach to Model Checking*, Proceedings of the 6th IEEE Symposium on Logic in Computer Science (Amsterdam), July 1991, pp. 406–415.

[HNW96] M. Huhn, P. Niebert, and F. Wallner, *Put your model checker on diet: verification on local states*, Technical Report TUM-I9642, Technische Universität München, December 1996.

[Koz83] Dexter Kozen, *Results on the propositional μ-calculus*, TCS **27** (1983), 333–354.

[LRT92] K. Lodaya, R. Ramanujam, and P.S. Thiagarajan, *Temporal logics for communicating sequential agents: I*, Int. Journal of Foundations of Computer Science **3** (1992), no. 2, 117–159.

[LT87] K. Lodaya and P.S. Thiagarajan, *A modal logic for a subclass of event structures*, Automata, Languages and Programming (T. Ottmann, ed.), LNCS, vol. 267, Springer, 1987, pp. 290–303.

[McM92] K.L. McMillan, *Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits*, Proceedings of the 4th Workshop on Computer Aided Verification (Montreal), 1992, pp. 164–174.

[Mil89] R. Milner, *Communication and concurrency*, Prentice Hall, 1989.

[MT89] M. Mukund and P.S. Thiagarajan, *An axiomatization of event structures*, Foundations of Software Technology and Theoretical Computer Science (C.E. Veni Madhavan, ed.), LNCS, vol. 405, Springer, 1989, pp. 143–160.

[Nie95] Peter Niebert, *A ν-calculus with local views for systems of sequential agents*, MFCS, LNCS, vol. 969, 1995.

[NPW80] M. Nielsen, G. Plotkin, and G. Winskel, *Petri nets, event structures and domains*, Theoretical Computer Science **13** (1980), no. 1, 85–108.

[Pel93] Doron Peled, *All from one, one for all: on model checking using representatives*, Computer Aided Verification CAV, LNCS, 1993.

[Pen97] W. Penczek, *Model-checking for a subclass of event structures*, TACAS, 97, *to appear.*

[Rei85] W. Reisig, *Petri Nets*, EATCS Monographs on Theoretical Computer Science, vol. 4, Springer, 1985.

[Sti92] Colin Stirling, *Modal and temporal logics*, Handbook of Logic in Computer Science (S. Abramsky, D. Gabbay, and T. Maibaum, eds.), Oxford University Press, 1992.

[Thi94] P.S. Thiagarajan, *A Trace Based Extension of PTL*, Proceedings of the 9th IEEE Symposium on Logic in Computer Science, 1994.

[Thi95] P.S. Thiagarajan, *A Trace Consistent Subset of PTL*, Proceedings of CONCUR '95 (Philadelphia, P.A., USA) (I. Lee and S.A. Smolka, eds.), LNCS, vol. 962, Springer, 1995, pp. 438–452.

[Val91] A. Valmari, *Stubborn Sets for Reduced State Space Generation*, Advances in Petri Nets 1990 (G.Rozenberg, ed.), LNCS, vol. 483, 1991, pp. 491–515.

[Wal95] Rolf Walter, *Petrinetzmodelle verteilter Algorithmen – Beweistechnik und Intuition*, Ph.D. thesis, Humboldt-Universität zu Berlin, Institut für Informatik, 1995, edition VERSAL, W. Reisig (Hrsg.), Dieter Bertz Verlag.