Put your Model Checker on Diet: Verification 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, very suited to act as models for branching-time logics interpreted on *local* states. We demonstrate how these 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. Thus – in contrast to interleaving branching time logics – in general they do not refer to the entire space of *global* states. We show that verification of local properties can be done by applying 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 unfolding. The construction is based on the finite prefix of a net unfolding defined by McMillan.

1 Introduction

One of the causes of the state explosion problem limiting verification of finite state systems 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.

We show in this paper that McMillan's construction is a very adequate basis for model checking branching time logics interpreted on *local states*. Here we

[⋆] 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.

understand a local state 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. Such 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 impossible. For the linear time case, such logics have been investigated by Thiagarajan in [Thi94, Thi95], local branching time logics were introduced in [LT87, LRT92].

We consider systems – described in terms of Petri nets – composed of sequential, non-deterministic 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. We demonstrate the use of the logic for specification on a multi-party network protocol.

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. This representation immediately can be used by proved interleaving model checkers [CS93, CES86], yielding efficient automated verification.

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).

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]. But the proof does not indicate *which* event *within* the prefix can serve as a representative for an event lying *outside* the prefix. The major problem to solve is to make the proof constructive, i.e. to determine a bisimilar representative for those events outside the prefix that are *direct* local successors of events within the prefix – without extending the prefix until these events are present. In particular, we have to find out whether such a successor exists at all.[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. 4 can be improved such that it never exceeds the costs of building the global state space.

[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.

The paper is structured as follows. In Section 2 we introduce basic definitions of the models we will use. In Section 3 we introduce the distributed μ-calculus and its formal semantics, and illustrate them with examples. In Section 4 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*. The elements of $P \cup T$ are 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 $N = (P, T, F)$ is a net and M_0 a marking of N. A marking M *enables* the transition t if for each $p \in P$, $F(p,t) \leq M(p)$. 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* at M if M enables t_1, and t_2 is enabled at M', where $M'(p) = M(p) - F(p, t_1)$ for each p.

We will exclusively regard *1-safe systems*, in which all reachable markings map each place to 0 or 1. Each marking can thus be identified with the set of places it maps to 1, and so $M \subseteq P$ for every reachable marking M.

A system is called *sequential* if every reachable marking concurrently enables exactly one transition.

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 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 $loc(x) := \{i \in I \mid x \in P_i \cup T_i\}$.

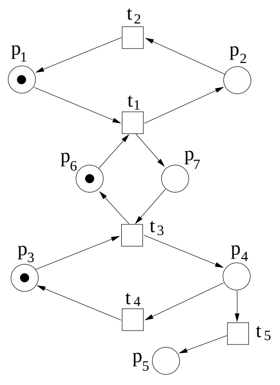


Fig. 1. Distributed net

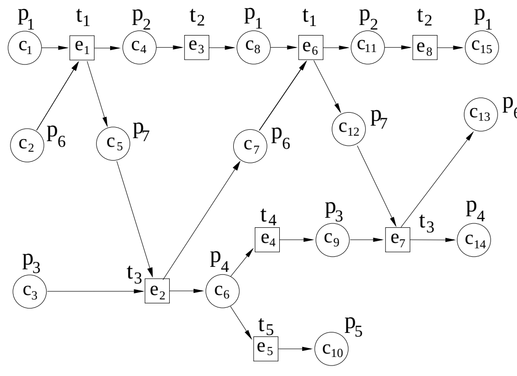


Fig. 2. Branching process

Fig. 1 shows a distributed net system consisting of three subsystems. The upper net can be seen as a producer, the lower as a consumer, and in between there is a single buffer cell, communicating with the producer and the consumer.

Branching processes. 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 $N = (B, E, F)$ where the irreflexive transitive closure of the flow relation F is well-founded and acyclic (and thus a (strict) partial order which we denote by $\prec$). Furtheron $|{}^\bullet b| \leq 1$ for every $b \in B$, and no transition is in self-conflict. The reflexive closure of $\prec$ determines a partial order, called *causal relation* and written as $\preceq$. In occurrence nets we speak of *conditions* and *events* instead of places and transitions, respectively. $\mathit{Min}(N)$ denotes the minimal elements of N with respect to $\preceq$, and $\mathit{Max}(X)$ the causally maximal elements of the set X of nodes.

Given two nets $N_i = (P_i, T_i, F_i)$ for $i \in \{1, 2\}$, the mapping $h : P_1 \cup T_1 \to P_2 \cup T_2$ is called *net homomorphism* if it embeds P_1 into P_2 and T_1 into T_2 as well, and if 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 analogue 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 net homomorphism, such that the restriction of π to $\mathit{Min}(N')$ is a bijection between $\mathit{Min}(N')$ 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 $\mathit{loc}(x)$ of a node x of N' is given by $\mathit{loc}(x) = \mathit{loc}(\pi(x))$. By E_J we denote the set of J*-events*, i.e., $E_J := \{e \in E \mid J \subseteq \mathit{loc}(e)\}$.

Given two distinct branching processes $\beta_1 = (N_1, \pi_1)$ and $\beta_2 = (N_2, \pi_2)$ of Σ, we say that β_1 and β_2 are *isomorphic* if there exists a bijective homomorphism

$h : N_1 \rightarrow N_2$, 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 $B_1 \subseteq B_2$ 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 of 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., $\downarrow e = \{e' \in E \,|\, e' \preceq e\}$. 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 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 the configuration* C. Notice that for *every* reachable marking M of the system there exist a (not necessarily unique) finite configuration C of the corresponding branching process, such that $\mathcal{M}(C) = M$. We call C and C' $\mathcal{M}$*-equivalent*, denoted $C =_\mathcal{M} C'$, if $\mathcal{M}(C) = \mathcal{M}(C')$.

An elementary property of two $\mathcal{M}$-equivalent configurations C, C' is that their "future" is equal, i.e., there exists an isomorphism between the part of *Unf* that lies behind of C and that one behind C', formally,

$$\beta(C) \text{ is isomorphic to } \beta(C') \quad \text{if} \quad C =_\mathcal{M} C',$$

where $\beta(C) := \{x \in B \cup E \mid \exists b \in Cut(C).\ b \preceq x \ \wedge\ \forall y \in Cut(C).\ \neg(x \# y)\}$.

In the case of $\downarrow e =_{\mathcal{M}} \downarrow e'$ and $|\downarrow e| < |\downarrow e'|$, the branching process $\beta(\downarrow e)$ can be seen as $\beta(\downarrow e')$ "shifted backward". Any configuration C' containing $\downarrow e'$ thus can be shifted backward to an $\mathcal{M}$-equivalent configuration C containing $\downarrow e$.

In [Esp94] this idea was formalised 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 $\mathit{shift}_{(e',e)}(C)$, is defined by

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

Local successor relation. For $J \subseteq I$, we call the configuration C' a J-*successor* of the configuration C, written as $C \xrightarrow{J} C'$, if C' is a state that lies in the future of C, such that on the path from C to C' all the components i in J execute exactly one event e, and nothing happens after e. Formally, $C \xrightarrow{J} C'$ iff

$$C \subseteq C' \quad \text{and} \quad \exists e \in Max(C').\, \forall i \in J.\ (C' \setminus C) \cap E_{\{i\}} = \{e\}.$$

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

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

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 $2^I \setminus \emptyset$. 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 $\langle \text{-} \rangle_J\, \varphi := \bigvee_{a \in Act} \langle a \rangle_J\, \varphi$ and $[\text{-}]_J\, \varphi := \bigwedge_{a \in Act} [a]_J\, \varphi$ where $a \in Act$.

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 $v : \mathcal{V} \longrightarrow 2^{\mathcal{C}_{loc}(Unf)}$ 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:

$$
\begin{array}{rlrl}
[\![p]\!]_v &= \{\downarrow e \mid \ p \in \mathcal{M}(\downarrow e)\} & [\![\neg p]\!]_v &= \{\downarrow e \mid \ p \notin \mathcal{M}(\downarrow e)\} \\
[\![\varphi \wedge \psi]\!]_v &= [\![\varphi]\!]_v \cap [\![\psi]\!]_v & [\![\varphi \vee \psi]\!]_v &= [\![\varphi]\!]_v \cup [\![\psi]\!]_v \\
[\![\nu x.\varphi]\!]_v &= \bigcup\{A \mid A \subseteq [\![\varphi]\!]_{v[x:=A]}\} & [\![\mu x.\varphi]\!]_v &= \bigcap\{A \mid [\![\varphi]\!]_{v[x:=A]} \subseteq A\}
\end{array}
$$

$$
\begin{array}{rl}
[\![\langle a\rangle_J\, \varphi]\!]_v &= \{\downarrow e \ \mid \ \exists e' \in E_J \,.\, l(e') = a \ \text{ and } \downarrow e \xrightarrow{J} \downarrow e' \ \text{ and } \downarrow e' \in [\![\varphi]\!]_v\} \\
[\![[a]_J\, \varphi]\!]_v &= \{\downarrow e \ \mid \ \forall e' \in E_J \,.\ \text{if } l(e') = a \ \text{ and } \downarrow e \xrightarrow{J} \downarrow e' \ \text{ then } \downarrow e' \in [\![\varphi]\!]_v\}
\end{array}
$$

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 $\downarrow\bot$ 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. The assertion 'eventually φ will hold in i' is given by $\mu x.\varphi \vee ([\text{-}]_i\, x \wedge \langle\text{-}\rangle_i\, \mathsf{true})$, and '$\varphi$ 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.

The more interesting properties, of course, are expressed by formulae referring to distinct subsystems. Let Ψ be the above mentioned 'φ always holds in i'. Then $\mu x.[\text{-}]_j\, x \vee [\text{-}]_i\, \Psi$ means that component j will eventually communicate with i, and afterwards φ holds indefinitely in i.

If $J = \{i, j, k\}$ then the formula $\nu y.[\text{-}]_i\, y \wedge (p \rightarrow \langle a\rangle_J\, \mathsf{true})$ describes that whenever p holds in i then i's next a-action may be a synchronization with j and k, where a also for j and k is the next step.

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:

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

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 line of events along which φ holds until ψ is satisfied.

Example. To give a flavour of the usage of our logic, and in order to show that it is indeed reasonable to argue in terms of local properties, we inspect the following *echo-algorithm* [Wal95].

Assume a (strongly connected) network N consisting of a set of agents Ag that includes *initiator* A_0. Each agent A_i communicates exclusively with her direct neighbours $N_i = \{A_{i_1}, \ldots, A_{i_n}\} \subseteq Ag$, and each agent (but the initiator) behaves the same way, as shown in Fig. 3. 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 $terminated_0$, she wants to be sure that every agent in the network has executed her local computation.

The first transition of an agent A_i is to receive a wake-up from one of her neighbours $A_k \in N_i$, and simultaneously to send a wake-up to her other neighbours $A_j \in N_i \setminus \{A_k\}$, firing $wkup_i(k)$, and changing her state to $start_i$. Then she executes her local computation. Afterwards she awaits the accept-signal from those of her neighbours which she waked up. Simultaneously she sends the accept-signal to the one neighbour from whom she had received the wake-up ($acpt_i(k)$). Eventually she will spontaneously return to her initial state $sleeping_i$.

For every pair (A_i, A_j) of neighbours there exist channels (i, j) and (j, i), that are initially all empty, and synchronise with the wake-up resp. accept transitions of the agents. To simplify the formulae we label each transition $wkup_i(j)$ by $wkup$ and each $acpt_i(j)$ by $acpt$.

One of the most interesting system characteristics that is expressible as a local property is that the protocol achieves synchronisation of all agents: the initiator cannot be terminated before all agents are in the state *accepted*.

$$\Sigma \models \mathsf{AG}_0(terminated_0 \rightarrow \textstyle\bigwedge_{i\geq 1} accepted_i)$$

Another property of interest guarantees that no agent A_i can be in $accepted_i$ if any of her neighbours still is sleeping, expressed by

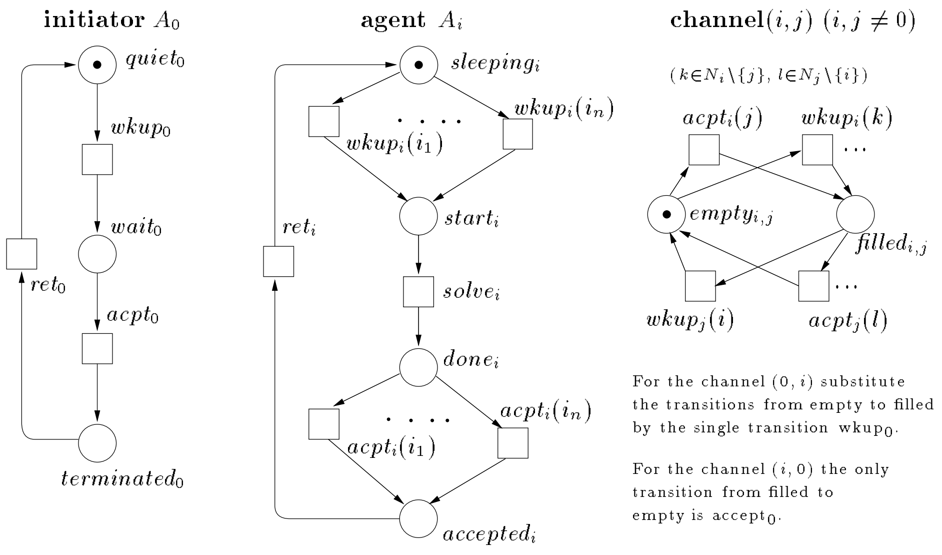
$$\Sigma \models \bigwedge_{i\geq 1} \mathsf{AG}_i(accepted_i \rightarrow \textstyle\bigwedge_{j\in N_i} \neg sleeping_j)$$

A third property is that in the first round of the protocol after the initial $wkup$ each agent A_i will wake up in her next step.[5]

$$\Sigma \models [wkup]_0 \bigwedge_{i\leq 1} \langle wkup\rangle_i\, \mathsf{true} \wedge [\text{-}wkup]_i\, \mathsf{false}^6$$

[5] In a later round of the protocol all agents wake up only in their *second* step after the initialising $wkup_0$ because in the local view of the initiator the others have to execute their restart action ret first.

[6] $[\text{-}a]_i\, \varphi \equiv \bigwedge_{b\in Act_i - \{a\}} [b]_i\, \varphi$, i.e. $[\text{-}a]_i\, \mathsf{false}$ says 'nothing than a must occur next in i'.



For the channel $(0,i)$ substitute the transitions from empty to filled by the single transition wkup_0.

For the channel $(i,0)$ the only transition from filled to empty is accept_0.

Fig. 3. Components for the echo-algorithm.

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

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

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 $[\![\phi]\!]_v^{\mathcal{T}} \subseteq S$, where v is the valuation function for the variables. We write $s \models_{\mathcal{T}} \phi$ if $s \in [\![\phi]\!]_v$. The semantics is defined inductively by:

$$\begin{array}{rlrl}
[\![p]\!]_v &= \{s \mid\ p \in \mathrm{I}(s)\} & [\![\neg p]\!]_v &= \{s \mid\ p \notin \mathrm{I}(s)\} \\
[\![\phi \wedge \chi]\!]_v &= [\![\phi]\!]_v \cap [\![\chi]\!]_v & [\![\phi \vee \chi]\!]_v &= [\![\phi]\!]_v \cup [\![\chi]\!]_v \\
[\![\nu x.\phi]\!]_v &= \bigcup\{A \mid A \subseteq [\![\phi]\!]_{v[x:=A]}\} & [\![\mu x.\phi]\!]_v &= \bigcap\{A \mid [\![\phi]\!]_{v[x:=A]} \subseteq A\}
\end{array}$$

$$[\![\langle a \rangle \phi]\!]_v = \{s \mid \exists s' \in S \,.\, s \xrightarrow{a} s' \text{ and } s' \in [\![\phi]\!]_v\}$$

$$[\![[a]\phi]\!]_v = \{s \mid \forall s' \in S \,.\, \text{ if } s \xrightarrow{a} s' \text{ then } s' \in [\![\phi]\!]_v\}$$

It is well-known that the distinguishing power of the μ-calculus is limited to standard bisimulation: A relation $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{T}$ 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

$$s \sim s' \;\text{ implies }\; s \models_{\mathcal{T}} \phi \;\Leftrightarrow\; s' \models_{\mathcal{T}} \phi \;\text{ for all closed } \mu\text{-calculus formulae } \phi.$$

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 $\mathcal{T}_{Unf} = \langle \mathcal{C}_{loc}(Unf), \downarrow\perp, \rightarrow, \widetilde{Act}, P, \mathrm{I}\rangle$ where $\downarrow e \xrightarrow{a_J} \downarrow e'$ iff $\downarrow e \xrightarrow{J} \downarrow e'$ and $l(e') = a$, and the interpretation of propositions $\mathrm{I}(\downarrow e) = \mathcal{M}(\downarrow e)$ for all $\downarrow e$.

Proposition 1. *Let $\downarrow e_1, \downarrow e_2 \in \mathcal{T}_{Unf}$. Then $\downarrow e_1 \sim \downarrow e_2$ iff $\downarrow e_1 =_{\mathcal{M}} \downarrow e_2$.*

By definition, if $\downarrow e_1 \neq_{\mathcal{M}} \downarrow e_2$, they are not bisimilar. Conversely, if $\downarrow e_1 =_{\mathcal{M}} \downarrow e_2$ then, based on the isomorphism $\mathcal{I}^{e_2}_{e_1}$, we define a relation $\mathcal{R}$ by setting $\downarrow e'_1 \,\mathcal{R}\, \downarrow e'_2$ iff $\downarrow e_1 \subseteq \downarrow e'_1$, $\downarrow e_2 \subseteq \downarrow e'_2$ and $\mathcal{I}^{e_2}_{e_1}(e'_1) = e'_2$. It is easily checked, that in fact $\mathcal{R}$ is a bisimulation and of course $\downarrow e_1 \,\mathcal{R}\, \downarrow e_2$. $\square$

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 $\downarrow e \in \mathcal{C}_{loc}(Unf)$.*

4 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 this prefix to obtain a finite transition system with local states, which is bisimilar to the unfolding.

Let $\mathit{Unf} = (N', \pi)$ be the unfolding of a net system, where $N' = (B, E, F)$. 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, formally:

$$\exists\, e' \in E_\perp : |{\downarrow} e'| < |{\downarrow} e| \text{ and } {\downarrow} e' =_{\mathcal{M}} {\downarrow} e.$$

Notice that in general for each cut-off event e there may be several corresponding events e' with the above property. In the sequel, we fix one of them and refer to it as $\mathit{corr}(e)$. 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.

The prefix *Fin* is usually 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. 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 partial orders instead. In the prefix generated by the refined algorithm, if e and e' are two different non-cut-off events, then the markings $\mathcal{M}({\downarrow} e)$ and $\mathcal{M}({\downarrow} e')$ are also different. Therefore, the number of non-cut-off events never exceeds the number of reachable states, and so the prefix can never be larger than the state space (up to a small constant).

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 McMillan's construction in fact guarantees that for each local configuration ${\downarrow} e$ in *Unf* there exists an $\mathcal{M}$-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 ${\downarrow} \mathit{corr}(e)$ in *Fin* which is minimal w.r.t. the size of $|{\downarrow} \mathit{corr}(e)|$. (In case of using the improved prefix [ERV96], $\mathit{corr}(e)$ 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 ${\downarrow} \mathit{corr}(e), {\downarrow} \mathit{corr}(e')$ in *Fin*, we can (imaginarily) "bend" all the transitions ${\downarrow} e \xrightarrow{a_J} {\downarrow} e'$ in $\mathcal{T}_{Unf}$ to transitions ${\downarrow} \mathit{corr}(e) \xrightarrow{a_J} {\downarrow} \mathit{corr}(e')$ in *Fin* (possibly merging infinitely many transitions into one). Since $\mathit{corr}(e)$ is unique and minimal, afterwards all local states ${\downarrow} e$ reachable from ${\downarrow} \perp$ 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. $\mathcal{T}_{Fin} \sim \mathcal{T}_{Unf}$.

Moreover, we have in fact obtained the *minimal* transition system bisimilar to $\mathcal{T}_{Unf}$ (which is determined uniquely up to isomorphism). By Proposition 2 we obtain as corollary:

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

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 $\mathcal{T}_{Fin}$. Moreover, it is guaranteed that $\mathcal{T}_{Fin}$ is not bigger than the global state space of the distributed net system – and often much smaller.

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 any event e in Fin to a unique non-cut-off e_0, such that $\downarrow e =_{\mathcal{M}} \downarrow e'$ implies $corr(e) = corr(e') = e_0$. The codomain of *corr* is called $E_{Rep} \subset E_{Fin}$, the set of representative events. Note that $\{\downarrow e \mid e \in E_{Rep}\}$ forms the state space of $\mathcal{T}_{Fin}$.
- a function *shift**, which maps any configuration $C = C_1$ of *Unf* containing some cut-off to a configuration $shift^*(C) = C' = C_n$ not containing a cut-off, hence being present in Fin. This function works by repeatedly applying $C_{i+1} := shift_{(e_i, corr(e_i))}(C_i)$ 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 $\beta(shift^*(C))$, which is the composition of the isomorphisms $\mathcal{I}^{e_i}_{corr(e_i)}$ induced by the chosen cut-off events. Moreover, $shift^*(\downarrow e)$ is strictly smaller than $\downarrow e$ (in the underlying order) for any $e \in \beta(C)$, and hence for any e, for which $C \xrightarrow{J} \downarrow e$.

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

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

$prefix\_successors(C, J, a) = \{\downarrow corr(e) \mid e \in E_{Fin} \wedge l(e) = a \wedge C \xrightarrow{J} \downarrow e\}$

$inheritable\_extension(C, e, J, a) = (\forall i \in J.\ (\downarrow e \setminus C) \cap E_i = \emptyset)$
(* predicate ensuring, that joining $\downarrow e$ to $C$ adds no $i$-events for $i \in J$ *)

$compatible\_cutoffs(C) = \{e \mid e$ is cut-off and $\downarrow e \cup C$ 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 $\emptyset$; fi (* we have closed a cycle *)
   result := $prefix\_successors(C, J, a)$; (* directly accessible successors *)
   v.pathmark:=true; (* put vertex on path *)
   for $e_c \in compatible\_cutoffs(C)$ do
       (* find successors outside the prefix behind $e_c$ *)
       if $inheritable\_extension(C, e_c, J, a)$ then
          result := result $\cup$ $successors(shift^*(C \cup \downarrow e_c), J, a)$;
       fi
   od ;
   v.pathmark:=false; (* take vertex from path *)
   return result;
}

proc $Compute\mathcal{T}_{Fin}$;
{  InitializeTransitionSystem(ts,$Fin$) (* extract state space from $Fin$ *)
   for $e \in E_{Rep}, a \in Act, \emptyset \neq J \subseteq I$ do
       for $\downarrow e' \in successors(\downarrow e, J, a)$ do
           add transition $\downarrow e \xrightarrow{a_J} \downarrow e'$
       od
   od
}
```

Fig. 4. The conceptual algorithm to compute $\mathcal{T}_{Fin}$

The procedure works as follows. Assume there exists at least one e' anywhere in *Unf* with $C \xrightarrow{J} \downarrow e'$; then there are two possibilities:

- 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 $prefix\_successors(C, J, a)$.
- 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 $(C \cup \downarrow e_c) \xrightarrow{J} \downarrow e'$. In this case we inherit the transition $C \xrightarrow{J} \downarrow e'$.

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-successors of C_c are J-successors of C (which is determined by *inheritable_extension*(C, e_c, J, a)) we want to search for the *successors*(C_c, J, a). But if any e' with $l(e') = a$ and $C_c \xrightarrow{J} \downarrow e'$ 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 depthfirst 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 an e' with $l(e') = a$ and $(C \xrightarrow{J} \downarrow e')$ exists. Then we choose from all these suitable J-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 stricly smaller e'_{min}. Thus in case we hit a configuration C twice, when searching for a-labelled J-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 a-labelled J-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 figure 4 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 (C, J, a) is explored *at most once*, essentially by storing intermediate results with the vertices in the hashtable. 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 transition system, but no larger.

However it can very well be, that the number of *visited* global states remains small compared to the number of *all* global states existing. Only practical experiments can give an answer here.

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.

5 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 this new logic 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 we believe that 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.

The scenario we presented is not limited to our logic. In fact, it can be used for any logic, which is based on local states, as they are present in $\mathcal{T}_{Fin}$. For instance [Pel93, Thi95] have proposed linear time logics, where the formulae are boolean combinations of linear time formulae that refer to a single location. The subclass of formulae, that are *conjunctions* of purely local formulae, can be checked with a standard linear time model checker on $\mathcal{T}_{Fin}$. There are many examples known, where *Fin* is much smaller than the interleaving based reduced state spaces used e.g. in [Pel93].

We plan to implement a prototype of our proposed model checking system within the PEP environment [Be94].

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

[Be94] E. Best and H. Fleischhack (eds.). PEP: Programming environment based on nets. Technical report, University of Hildesheim, 1994.

[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(2):244–263, April 1986.

[CS93] R. Cleaveland and B. Steffen. A linear time model-checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2:121- 147, 1993.

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

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

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

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

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[LRT92] K. Lodaya, R. Ramanujam, and P.S. Thiagarajan. Temporal logics for communicating sequential agents: I. *Int. Journal of Foundations of Computer Science*, 3(2):117–159, 1992.

[LT87] K. Lodaya and P.S. Thiagarajan. A modal logic for a subclass of event structures. In T. Ottmann, editor, *Automata, Languages and Programming*, volume 267 of *LNCS*, pages 290–303. Springer, 1987.

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

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

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

[NPW80] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains. *Theoretical Computer Science*, 13(1):85–108, 1980.

[Pel93] D. Peled. All from one, one for all: on model checking using representatives. In *International Conference on Computer Aided Verification (CAV)*, volume 697 of *LNCS*, pages 409–423, Elounda, Greece, 1993.

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

[Sti92] C. Stirling. Modal and temporal logics. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

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

[Thi95] P.S. Thiagarajan. A Trace Consistent Subset of PTL. In I. Lee and S.A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *LNCS*, pages 438–452, Philadelphia, P.A., USA, 1995. Springer.

[Val91] A. Valmari. Stubborn Sets for Reduced State Space Generation. In G.Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *LNCS*, pages 491–515. Springer, 1991.

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