Electronic Notes in Theoretical Computer Science 68 No. 4 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume68.html 16 pages

Using Assumptions to Distribute CTL Model Checking

Luboš Brim [a, 1,2], Jitka Crhová [a, 1,3], Karen Yorav [b, 4]

[a] *Faculty of Informatics, Masaryk University, Brno, Czech Republic*
[b] *Technion, Haifa, Israel and CMU, Pittsburg, U.S.A*

Abstract

In this work we discuss the problem of performing distributed CTL model checking by splitting the given state space into several "partial state spaces". The partial state space is modelled as a Kripke structure with border states. Each computer involved in the distributed computation owns a partial state space and performs a model checking algorithm on this incomplete structure. To be able to proceed, the border states are augmented by assumptions about the truth of formulas and the computers exchange assumptions about relevant states as they compute more precise information. In the paper we give the basic definitions and present the distributed algorithm.

1 Introduction

The main aim in exploiting a distributed environment for model checking is to extend the applicability of model checking algorithms to larger and more complex systems. Many "sequential" approaches have been proposed to deal with large state spaces, e.g. partial-order methods, symbolic verification, abstractions, and partial state space reasoning. Often these approaches do not suffice – time or space resources can still significantly limit the practical applicability. A parallel super computer, grid or a network of computers can provide extra resources needed to fight more realistic verification problems. Here we consider a cheap variant – a network of workstations that communicate via message passing.

[1] Research supported by the Grant Agency of Czech Republic grant No. 201/00/1023 and by Ministry of Education grant FRVŠ No. B598/G4/2002
[2] Email: brim@fi.muni.cz
[3] Email: xcrhova@fi.muni.cz
[4] Email: laster@cs.technion.co.il

©2002 Published by Elsevier Science B. V.

The important feature of algorithms running in a distributed environment is to solve the given task by distributing the data among the participating workstations with as small amount of coordination as possible. One of the main issues in distributing model checking algorithms is how to partition the state space (data) among the individual computers called here *network nodes*. There are two aspects that significantly influence the overall effectiveness of model-checking in the distributed environment: *locality* and (spatial) *balance* of the state space partition. Locality means that most of the state's descendants are assigned to the same node as the parent state, thus reducing communication and cooperation overhead. Balance means that each network node is assigned approximately the same number of states, thus achieving good speed-up.

The main idea of many distributed algorithms is similar: the state graph is partitioned among the network nodes, i.e., each network node owns a subset of the state space. The differences are in the way the state space is partitioned (*partition function*). Probabilistic techniques to partition the state space have been used e.g. in [10,14,1], and a technique which exploits some structural properties derived from the verified formula has been proposed in [2].

The model checking algorithm running on each network node has thus access only to a part of the entire system. Depending on the type of the algorithm it communicates with other nodes to achieve the required (global) result.

Laster (Yorav) and Grumberg [9,16] have developed an approach to model checking of software which uses modularity. Their notion of a module differs from that used in modular model checking as understood for example in [7,8,3,13]. A module here is not a part of a whole system that runs in parallel with other modules (i.e. that contributes to the whole system in a multiplicative way), but a subset of a state space that originates from splitting the whole system in an additive way. It is defined by following the syntactical structure of the program. This notion of module has also been used in [11], where the system is splitted according to the semantics of the program.

Besides this partition, the authors in [9,16] have also defined the notion of an assumption function that represents partial knowledge about truth of formulas provided by the rest of the system (by other parts).

In this contribution we propose a technique that explores the possibility to extend the approach of Laster and Grumberg to partitions not necessarily resulting from the syntactical structure of the program, allowing thus a distributed model checking. Furthermore, we have modified the model checking algorithm in such a way that it can be run in a distributed environment.

The main ideas are similar to the ideas introduced by Laster and Grumberg. Once the system is partitioned, the Kripke structure on each network node can contain states that represent "border" states, which are those states that in fact belong to some other network node. Whenever the model checking algorithm reaches a border state it uses information provided by other

network nodes about the truth of formulas in that state – assumptions. As the assumptions can change, a re-computation is necessary in general. There are several scenarios how to reduce the amount of required re-computations. In all cases we have also to take into account the associated communication complexity.

2 CTL Semantics under Assumptions

Our aim is to perform a model checking algorithm on a cluster of n workstations, called *(network) nodes*. In addition to the sequential case a *partition function* f is used to partition the state space among the nodes. After partitioning the state space, each node owns a part of the original state space. For each state s the value $f(s)$ is the identifier of the node the state belongs to. For simplicity we use natural numbers to identify nodes.

We model the state space owned by one network node as a Kripke structure with *border states*. Intuitively, border states are states that in fact belong to other nodes and within the Kripke structure they represent the missing parts of the state space.

Definition 2.1 A *Kripke structure* is a tuple $M = (S, R, I)$ where S is a finite set of *states*, $R \subseteq S \times S$ is a *transition relation* and $I \subseteq S$ is a set of *initial states*. The set of *border states* is $border(M) = \{s \in S \mid \neg\exists s'.(s, s') \in R\}$.

A Kripke structure M is called *total* if $border(M) = \emptyset$. We suppose that the whole system under consideration is modelled as a *total* Kripke structure M with the set I of initial states containing one *initial state* $\hat{s}$. Once the given system is partitioned, the resulting Kripke structures $K_1, \ldots, K_n$ do not need to be total. In section 3 we describe a particular technique of transforming M into the parts $K_1, \ldots, K_n$. Kripke structures resulting from the given Kripke structure by partitioning it are called *fragments*. A fragment M_1 of M is a substructure of M satisfying the property that every state in M_1 has either no successor in M_1 or it has exactly the same successors as in M.

A path π in a Kripke structure M from a state s_0 is a sequence $\pi = s_0 s_1 \ldots$ such that $\forall i \geq 0 : s_i \in S$ and $(s_i, s_{i+1}) \in R$. A maximal path is a paths that is either infinite or ends in a border state. For a maximal path we denote by $|\pi|$ the length of the path. In case the path is infinite we put $|\pi| = \infty$.

Definition 2.2 A Kripke structure $M_1 = (S_1, R_1, I_1)$ is a *fragment* of a Kripke structure $M = (S, R, I)$ iff

(i) $S_1 \subseteq S$,

(ii) $R_1 \subseteq R$

(iii) $I_1 = I \cap S_1$

(iv) $\forall (s_1, s_2) \in R$: if $s_1 \in S_1$, then either $(s_1, s_2) \in R_1$ or $s_1 \in border(M_1)$.

In this paper we consider a state based branching time temporal logic CTL.

Definition 2.3 The language of CTL is defined by the following abstract syntax:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{AX}\varphi \mid \mathbf{EX}\varphi \mid \mathbf{A}(\varphi_1 \,\mathbf{U}\, \varphi_2) \mid \mathbf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2)$$

where p ranges over *atomic propositions* taken from a set AP.

Let φ be a CTL formula. We denote by $cl(\varphi)$ the set of all subformulas of φ and by $tcl(\varphi)$ the set of all subformulas of φ of the form $\mathbf{EX}\varphi$,$\mathbf{AX}\varphi$,$\mathbf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2)$, or $\mathbf{A}(\varphi_1 \,\mathbf{U}\, \varphi_2)$.

To define the semantics of CTL formulas over Kripke structures with border states we need to adapt the standard semantic definition. CTL is usually interpreted over total structures, while our structures are typically non-total. Furthermore, we need to define the notion of the truth under assumptions associated with border states. Here we use a modification of the notion of the truth under assumptions as defined in [9].

Definition 2.4 An *assumption function* for a Kripke structure $M = (S, R, I)$ and a CTL formula ψ is a partial function $\mathcal{A} : S \times cl(\psi) \to Bool$.

We use the notation $\mathcal{A}(s, \varphi) = \perp$ to say that the value of $\mathcal{A}(s, \varphi)$ is undefined. By $\mathcal{A}_\perp$ we denote the assumption function which is undefined for all inputs. Intuitively, $\mathcal{A}(s, \varphi) = \mathbf{true}$ if we can assume that φ holds in the state s, $\mathcal{A}(s, \varphi) = \mathbf{false}$ if we can assume that φ does not hold in the state s, and $\mathcal{A}(s, \varphi) = \perp$ if we cannot assume anything. Let us denote by AS^ψ_M the set of all assumption functions for the Kripke structure M and the formula ψ.

Definition 2.5 Let $M = (S, R, I)$ be a Kripke structure, $\mathcal{L} : AP \to 2^S$ a *valuation* assigning to each atomic proposition a set of states, and ψ a formula. We define the function $\mathcal{C}_M : AS^\psi_M \to AS^\psi_M$. For $\mathcal{A}_{in} \in AS^\psi_M$ let $\mathcal{A} = \mathcal{C}_M(\mathcal{A}_{in})$. Then $\mathcal{A}$ is defined inductively as follows:

(i) Propositional operators ($\varphi \notin tcl(\psi)$):

- $\mathcal{A}(s, p) = \begin{cases} \mathbf{true} & \text{if } s \in \mathcal{L}(p) \\ \mathbf{false} & \text{otherwise} \end{cases}$
- $\mathcal{A}(s, \varphi_1 \wedge \varphi_2) = \begin{cases} \mathbf{true} & \text{if } \mathcal{A}(s, \varphi_1) = \mathbf{true} \text{ and } \mathcal{A}(s, \varphi_2) = \mathbf{true} \\ \mathbf{false} & \text{if } \mathcal{A}(s, \varphi_1) = \mathbf{false} \text{ or } \mathcal{A}(s, \varphi_2) = \mathbf{false} \\ \perp & \text{otherwise} \end{cases}$
- $\mathcal{A}(s, \neg\varphi_1) = \begin{cases} \mathbf{true} & \text{if } \mathcal{A}(s, \varphi_1) = \mathbf{false} \\ \mathbf{false} & \text{if } \mathcal{A}(s, \varphi_1) = \mathbf{true} \\ \perp & \text{otherwise} \end{cases}$

(ii) Temporal operators ($\varphi \in tcl(\psi)$):
a. if $s \in border(M)$ then $\mathcal{A}(s, \varphi) = \mathcal{A}_{in}(s, \varphi)$

b. if $s \notin \mathit{border}(M)$ then $\mathcal{A}(s,\varphi)$ is defined as follows:

- $\mathcal{A}(s,\mathbf{AX}\varphi_1) = \begin{cases} \mathbf{true} & \text{if } \forall s' \in S : (s,s') \in R \Rightarrow \mathcal{A}(s',\varphi_1) = \mathbf{true} \\ \mathbf{false} & \text{if } \exists s' \in S : (s,s') \in R \wedge \mathcal{A}(s',\varphi_1) = \mathbf{false} \\ \bot & \text{otherwise} \end{cases}$

- $\mathcal{A}(s,\mathbf{EX}\varphi_1) = \begin{cases} \mathbf{true} & \text{if } \exists s' \in S : (s,s') \in R \wedge \mathcal{A}(s',\varphi_1) = \mathbf{true} \\ \mathbf{false} & \text{if } \forall s' \in S : (s,s') \in R \Rightarrow \mathcal{A}(s',\varphi_1) = \mathbf{false} \\ \bot & \text{otherwise} \end{cases}$

- $\mathcal{A}(s,\mathbf{A}(\varphi_1 \,\mathbf{U}\, \varphi_2)) = \begin{cases} \mathbf{true} & \text{if for all paths } \pi = s_0 s_1 s_2 \ldots \text{ with } s = s_0 \\ & \text{there exists an index } x < |\pi| \text{ such that:} \\ & [\text{either } \mathcal{A}(s_x,\varphi_2) = \mathbf{true} \text{ or } (s_x \in \mathit{border}(M) \\ & \qquad \text{and } \mathcal{A}_{in}(s_x, \mathbf{A}(\varphi_1 \,\mathbf{U}\, \varphi_2)) = \mathbf{true})], \\ & \text{and } \forall y : 0 \le y < x : \ \mathcal{A}(s_y,\varphi_1) = \mathbf{true} \\ \mathbf{false} & \text{if there exists a path } \pi = s_0 s_1 s_2 \ldots \text{ with} \\ & s = s_0 \text{ such that either } \exists x < |\pi| \text{ such that} \\ & (\mathcal{A}(s_x,\varphi_1) = \mathbf{false} \text{ and} \\ & \qquad \forall y \le x : \ \mathcal{A}(s_y,\varphi_2) = \mathbf{false}) \\ & \text{or } \forall x < |\pi| : (\mathcal{A}(s_x,\varphi_2) = \mathbf{false} \ \text{ and} \\ & (|\pi| = \infty \text{ or } \mathcal{A}_{in}(s_{|\pi|-1}, \mathbf{A}(\varphi_1 \,\mathbf{U}\, \varphi_2)) = \mathbf{false})) \\ \bot & \text{otherwise} \end{cases}$

- $\mathcal{A}(s,\mathbf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2)) = \begin{cases} \mathbf{true} & \text{if there exists a path } \pi = s_0 s_1 s_2 \ldots \text{ with} \\ & s = s_0 \text{ such that } \exists x < |\pi| \text{ such that} \\ & (\text{either } \mathcal{A}(s_x,\varphi_2) = \mathbf{true} \\ & \text{or } (s_x \in \mathit{border}(M) \text{ and} \\ & \qquad \mathcal{A}(s_x, \mathbf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2)) = \mathbf{true})), \\ & \text{and } \forall 0 \le y < x : \ \mathcal{A}(s_y,\varphi_1) = \mathbf{true} \\ \mathbf{false} & \text{if for all paths } \pi = s_0 s_1 s_2 \ldots \text{ with } s = s_0 \\ & \text{either } \exists x < |\pi| \text{ such that} \\ & (\mathcal{A}(s_x,\varphi_1) = \mathbf{false} \text{ and} \\ & \qquad \forall y \le x : \ \mathcal{A}(s_y,\varphi_2) = \mathbf{false}) \\ & \text{or } \forall x < |\pi| : (\mathcal{A}(s_x,\varphi_2) = \mathbf{false} \ \text{ and} \\ & (|\pi| = \infty \text{ or } \mathcal{A}_{in}(s_{|\pi|-1}, \mathbf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2)) = \mathbf{false})) \\ \bot & \text{otherwise} \end{cases}$

For a given assumption function $\mathcal{A}$ we define the standard notion of truth $M, s \models_{\mathcal{A}} \psi$ as $\mathcal{C}_M(\mathcal{A})(s,\psi)$. The truth of a formula in a state is thus relative to given assumptions.

Notice that a value of an assumption function $\mathcal{A}_{in}(s,\varphi)$ for a state $s \notin \mathit{border}(M)$ does not influence the value $\mathcal{C}_M(\mathcal{A}_{in})$. Hence, the truth under

assumptions relates to the standard notion of the truth over total Kripke structures in the following way.

Proposition 2.6 *For any* total *Kripke structure M, valuation $\mathcal{L}$, CTL formula ψ and an assumption function $\mathcal{A}_{in} \in AS_M^\psi$*

$$M, s \models \psi \text{ iff } \mathcal{C}_M(\mathcal{A}_{in})(s, \psi) = \textbf{true}$$

Notice that the truth of a formula in a state $s \notin border(M)$ depends on the assumtpion function.

An important feature of the the semantic function $\mathcal{C}_M$ is that assumptions are preserved for fragments.

Definition 2.7 Let $M = (S, R, I)$ be a Kripke structure, $\mathcal{A}_{in}, \mathcal{A} \in AS_M^\psi$, ψ a CTL formula. We say that $\mathcal{A}$ is *correct* for a state $s \in S$ and a formula $\varphi \in cl(\psi)$ (w.r.t. M and $\mathcal{A}_{in}$) iff

$$\mathcal{A}(s, \varphi) = \mathcal{C}_M(\mathcal{A}_{in})(s, \varphi)$$

We say that $\mathcal{A}$ is *correct* for a state $s \in S$ (w.r.t. M and $\mathcal{A}_{in}$) if for every $\varphi \in cl(\psi)$ it is correct for s and φ.

Lemma 2.8 *Let $M = (S, R, I)$ be a Kripke structure, $M_1 = (S_1, R_1, I_1)$ its fragment, and $\mathcal{A}_{in}, \mathcal{A}_1 \in AS_M^\psi$. If $\mathcal{A}_1$ is correct for every $s \in border(M_1)$ (w.r.t. M and $\mathcal{A}_{in}$) then $\mathcal{C}_{M_1}(\mathcal{A}_1)$ is correct for every $s \in S_1$.*

For the proof of the Lemma and all the other proofs we refere to the full version of the paper [4].

3 Distributed CTL Model Checking Algorithm

In this section we describe the algorithm for distributed CTL model checking. The algorithm first partitions the given state space (Kripke structure) among the participating network nodes. This can be done locally (on the fly) or globally depending on the type of the basic model checking algorithm performed on the network nodes.

Definition 3.1 Let $M = (S, R, I)$ be a Kripke structure, $T \subseteq S$. We define the Kripke structure $Fragment_M(T) = (S_T, R_T, I_T)$ as follows:

(i) $S_T = \{s \in S \mid s \in T \vee \exists s' \in T \text{ s.t. } (s', s) \in R\}$

(ii) $R_T = \{(s_1, s_2) \in R \mid s_1 \in T, s_2 \in S_T\}$

(iii) $I_T = \{s \in S_T \mid s \in I\}$

The states from T are called *original*, the states from $S_T \setminus T$ are called *subsequent* in $Fragment_M(T)$.

The structure $Fragment_M(T)$ contains the states from T and all its (immediate) successors, and all transitions *from* states in T. Initial states are those which are initial in M.

proc *Basic Node Algorithm*
{Let $cl(\psi) = \{\varphi_1, \ldots, \varphi_z\}$ such that $\varphi_i \in cl(\varphi_j) \Rightarrow i \leq j$;
for $i := 1$ **to** z **step** 1 **do**
begin
case φ_i **of**
• $\varphi_i = p, p \in AP$:
forall $s \in S$ **do**
if $\varphi \in \mathcal{L}(s)$ **then** $\mathcal{A}(s, \varphi) :=$ **true** **else** $\mathcal{A}(s, \varphi) :=$ **false** **od**
• $\varphi_i = \varphi_1 \wedge \varphi_2$:
forall $s \in S$ **do**
if $\mathcal{A}(s, \varphi_1) =$ **true** and $\mathcal{A}(s, \varphi_2) =$ **true** **then** $\mathcal{A}(s, \varphi_i) :=$ **true**;
if $\mathcal{A}(s, \varphi_1) =$ **false** or $\mathcal{A}(s, \varphi_2) =$ **false** **then** $\mathcal{A}(s, \varphi_i) :=$ **false** **od**
• $\varphi_i = \neg\varphi_1$:
forall $s \in S$ **do** **if** $\mathcal{A}(s, \varphi_1) \neq \perp$ **then** $\mathcal{A}(s, \varphi_i) := \neg\mathcal{A}(s, \varphi_1)$ **od**
• $\varphi_i \in tcl(\psi)$:
forall $s \in border(M)$ **do** $\mathcal{A}(s, \varphi_i) := \mathcal{A}_{in}(s, \varphi_i)$ **od**;
case φ_i **of**
• $\varphi_i = \mathbf{EX}\varphi_1$:
forall $s \in S \setminus border(M)$ **do**
if $\exists s' \in S : (s_j, s') \in R$ and $\mathcal{A}(s', \varphi_1) =$ **true** **then** $\mathcal{A}(s_j, \varphi_i) :=$ **true**;
if $\forall s' \in S : (s, s') \in R \Rightarrow \mathcal{A}(s', \varphi_1) =$ **false** **then** $\mathcal{A}(s, \varphi_i) :=$ **false** **od**
• $\varphi_i = \mathbf{AX}\varphi_1$:
forall $s \in S \setminus border(M)$ **do**
if $\exists s' \in S : (s, s') \in R$ and $\mathcal{A}(s', \varphi_1) =$ **false** **then** $\mathcal{A}(s, \varphi_i) :=$ **false**;
if $\forall s' \in S : (s, s') \in R \Rightarrow \mathcal{A}(s', \varphi_1) =$ **true** **then** $\mathcal{A}(s, \varphi_i) :=$ **true** **od**
• $\varphi_i = \mathbf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2)$:
forall $s \in S$ **do** **if** $\mathcal{A}(s, \varphi_2) =$ **true** **then** $\mathcal{A}(s, \varphi_i) :=$ **true** **od**;
while $\exists s \in S : \mathcal{A}(s, \varphi_i) \neq$ **true** and $\mathcal{A}(s, \varphi_1) =$ **true** and
$(\exists s' \in S : (s, s') \in R$ and $\mathcal{A}(s', \varphi_i) =$ **true**) **do** $\mathcal{A}(s, \varphi_i) :=$ **true** **od**
forall $s \in S$ **do** **if** $\mathcal{A}(s, \varphi_i) = \perp$ and $\mathcal{A}(s, \varphi_2) =$ **false**
then $\mathcal{A}(s, \varphi_i) :=$ **false** **od**;
while $\exists s \in S : \mathcal{A}(s, \varphi_i) =$ **false** and $\mathcal{A}(s, \varphi_1) \neq$ **false** and
$(\exists s' \in S : (s, s') \in R$ and $\mathcal{A}(s', \varphi_i) \neq$ **false**) **do** $\mathcal{A}(s, \varphi_i) := \perp$ **od**
• $\varphi_i = \mathbf{A}(\varphi_1 \,\mathbf{U}\, \varphi_2)$:
forall $s \in S$ **do** **if** $\mathcal{A}(s, \varphi_2) =$ **true** **then** $\mathcal{A}(s, \varphi_i) :=$ **true** **od**;
while $\exists s \in S : \mathcal{A}(s, \varphi_i) \neq$ **true** and $\mathcal{A}(s, \varphi_1) =$ **true** and
$(\forall s' \in S : (s, s') \in R \Rightarrow \mathcal{A}(s', \varphi_i) =$ **true**) **do** $\mathcal{A}(s, \varphi_i) :=$ **true** **od**;
forall $s \in S$ **do** **if** $\mathcal{A}(s, \varphi_i) = \perp$ and $\mathcal{A}(s, \varphi_2) =$ **false**
then $\mathcal{A}(s, \varphi_i) :=$ **false** **od**;
while $\exists s \in S : \mathcal{A}(s, \varphi_i) =$ **false** and $\mathcal{A}(s, \varphi_1) \neq$ **false** and
$(\forall s' \in S : (s, s') \in R \Rightarrow \mathcal{A}(s', \varphi_i) \neq$ **false**) **do** $\mathcal{A}(s, \varphi_i) := \perp$ **od**
esac
esac
end **od**
end

Fig. 1. Modified model checking algorithm – "Node Algorithm"

Lemma 3.2 *Let $M = (S, R, I)$ be a Kripke structure, $T \subseteq S$. Then $Fragment_M(T)$ is a fragment of M.*

The result of splitting the given state space is a collection of fragments called a *partitioning*.

Definition 3.3 Let $M = (S, R, I)$ be a Kripke structure and $f : S \to \{1, \ldots, n\}$ a total function (partition function). A *partitioning* of M under f is a tuple $K_M^f = (K_1, \ldots, K_n)$ such that $\forall i \in \{1, \ldots, n\} : K_i = Fragment_M(\{s \in S \mid f(s) = i\})$.

Figure 2 shows an example of a system and its partitioning for a partition function $f : \{s_1, \ldots, s_6\} \to \{1, 2, 3\}$, $f(s_1) = f(s_2) = 1$, $f(s_3) = f(s_4) = 2$, $f(s_5) = f(s_6) = 3$. Border states are marked with dotted circles.

In model checking we are interested in answering the question whether $M, s \models \psi$. Due to Proposition 2.6 this is equivalently expressed as $\mathcal{C}_M(\mathcal{A}_\perp)(s, \psi) = \mathbf{true}$. Therefore we can answer the model checking question by computing the assumption function $mc(\mathcal{A}_\perp)$ and return its value on the input (s, ψ). To be able to distribute the computatiton of $mc(\mathcal{A}_\perp)$, we (iteratively) compute assumption functions that are defined on parts of the system M only. We exploit Lemma 2.8 that ensures us that results of these assumption functions equal those of assumption function $mc(\mathcal{A}_\perp)$.

Let us fix a total Kripke structure $M = (S, R, I)$, a CTL formula ψ, and a (partition) function $f : S \to \{1, \ldots, n\}$ as inputs of the algorithm. Moreover, let us denote by $K_M^f = (K_1, \ldots, K_n)$ the corresponding partitioning and let $K_i = (S_i, R_i, I_i)$ for all $i \in \{1, \ldots, n\}$.

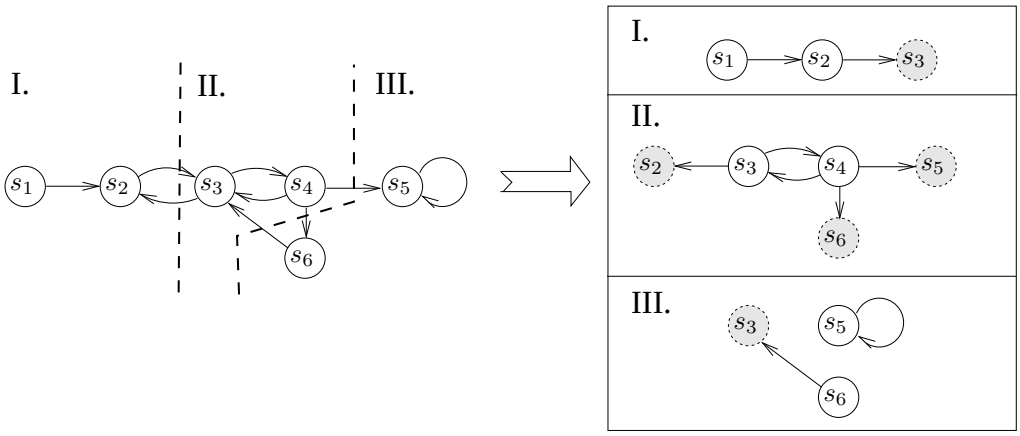

Fig. 2. Fragments

The distributed algorithm uses a procedure (*node algorithm*) for computing the function $\mathcal{C}_{K_i}$ on each fragment K_i. We consider a modification of an explicit state CTL model checking algorithm (see [6]), but other model checking algorithms can be adapted as well, in particular symbolic algorithms. Intuitively, the node algorithm performs standard model checking, but is able to cope with "undefined values" as well. Moreover, it computes both the positive

and negative results, i.e., if a state s has a successor in which φ is true, it can be concluded both that s satisfies $\mathbf{EX}\varphi$ and that it does not satisfy $\mathbf{AX}\neg\varphi$, even when the validity of φ in other successors of s is undefined yet. The pseudocode of the explicit state *node algorithm* is given in Figure 1.

The main idea of the distributed algorithm is the following. Each fragment K_i is managed by a separate process P_i. These processes are running in parallel on each network node.

Each process P_i initializes the assumption function $\mathcal{A}_i$ to the undefined assumption function $\mathcal{A}_\perp$. After initialization it computes (using the node algorithm) the function $\mathcal{C}_M(\mathcal{A}_i)$. Then it sends the results to each process P that may be interested in them (i.e., it sends the part of the assumption function for P's border states) and receives similar information from the other processes. These steps are repeated until a *fixpoint is reached* ("global" stabilization occurs), i.e. until no new information can be computed.

After stabilization there still may remain a state s and a formula φ for which $\mathcal{A}_i(s,\varphi) = \perp$. This can happen in the case of the $\mathbf{U}$ operator.

A possible situation is exemplified in Figure 3.

The state space has three states $S = \{s_1, s_2, s_3\}$ equally distributed on the three network nodes. Suppose the valuation is such that $\mathcal{L}(p) = S$ and $\mathcal{L}(q) = \emptyset$. If we want to model check the formula $\varphi = \mathbf{A}(p\mathbf{U}q)$ then each node algorithm reaches fixpoint with value of φ being undefined in the border state.

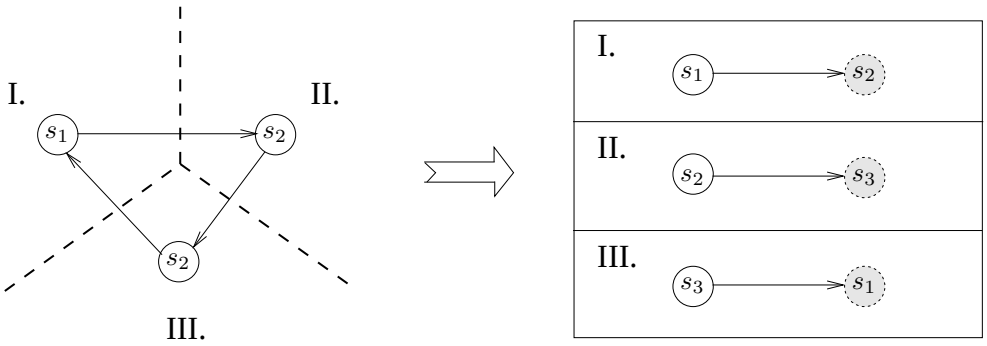

Fig. 3. Undefined assumptions

However, if the truth of all subformulas of φ has already been computed in all states in all nodes, then from the fact that the fixpoint has been reached we can conclude that φ does not hold in s. Therefore all processes extrapolate this information and continue to compute. In our example this results in the answer that φ does not hold in any state of S.

The described computation is repeated until the information we are searching for is fully computed. The main idea of the distributed algorithm is summarized in Figure 4.

We now elaborate the distributed algorithm so as to be able to argue about its correctness. The detailed pseudocode is given in Figure 5.

Notice that there are two main stages in the execution of algorithm. In

```
proc Distributed Algorithm (input: total KS M, ψ, f; output: A_{f(ŝ)}(ŝ, ψ))
  Split M into K_i;
  forall i ∈ {1, ..., n} do in parallel {for all K_i}
    Take the initial assumption function;
    repeat
        repeat
            Compute all you can;
            Send relevant information to other nodes;
            Receive relevant information from other nodes;
        until all processes reach fixpoint;
        Extrapolate additional information;
    until all is computed;
    Return result for the initial state ŝ;
  od
end
```

Fig. 4. Main Idea of the Distributed Algorithm

the first stage the processes repeatedly compute information about truth of formulas and send and receive computed information to and from other processes, respectively. This stage finishes when a fixpoint is reached. Then the second stage is performed, when each process extrapolates information, using the fact that the fixpoint has been reached. These two stages are performed repeatedly until the information we search for is computed. Let us denote the beginning of the first and second stage point I and II, respectively, as marked in the algorithm. The algorithm is at point II exactly when the fixpoint is reached. There is no synchronization on the beginning of the first stage, but without loss of generality we can assume that all processes start the first stage at the same time.

For each state and each formula we want to say if its value has already been computed or not. We consider a value for a state and a formula computed if an appropriate value of $\mathcal{A}_i$ has already been defined for some i. After reaching the fixpoint at the end of stage I it can be necessary to resolve undefined values to be able to continue in the computation, i.e. to consider as computed a value for some state and some formula. We choose a pair for which the formula is the "lowest" subformula.

Let us denote by *Def* the set of all tuples from $S \times cl(\psi)$ that have already been computed in this sense, and *Undef* its complement. Formally,

$$Def = \{(s, \varphi) \in S \times cl(\psi) \mid \exists i \in \{1, \ldots, n\} : \mathcal{A}_i(s, \varphi) \neq \bot\}$$

Undef is the complement of *Def* in $S \times cl(\psi)$.

Now, let us define an ordering $\leq$ on $S \times cl(\psi)$. It formalizes the notion of a tuple that is minimal in *Undef*.

Definition 3.4 Let $s_1, s_2 \in S, \varphi_1, \varphi_2 \in cl(\psi)$. Then

$$(s_1, \varphi_1) \leq (s_2, \varphi_2) \Leftrightarrow \varphi_1 \in cl(\varphi_2)$$

```
1  proc
2    Split M into K_i;
3    forall i ∈ {1,...,n} do in parallel
4      {Process P_i}
5      A_i := A_⊥;
6      repeat
7      {stage I:}
8      repeat
9        A'_i := C_{K_i}(A_i)
10       forall φ ∈ tcl(ψ), s ∈ S_i : s is original in K_i and subsequent in K_j do
11         if A'_i(s,φ) ≠⊥  and  A_i(s,φ) =⊥
12           then  send A'_i(s,φ) to the process P_j od;
13       forall received A_j(s,φ) do A'_i(s,φ) := A_j(s,φ)  od;
14       A_i := A'_i
15     until  all processes reach fixpoint;
16     {stage II:}
17     forall φ ∈ {E(φ1 U φ2), A(φ1 U φ2)}, s ∈ border(K_i) do
18       if (s,φ) is minimal in Undef  then A_i(s,φ) = false  od
19     until A_i(s,φ) ≠⊥,  ∀φ ∈ cl(ψ), ∀s ∈ S_i
20   od;
21   return A_{f(ŝ)}(ŝ,ψ)
22 end
```

Fig. 5. Distributed algorithm for computing $\mathcal{C}_M$

The fact that a fixpoint has been reached cannot be detected locally. However, by employing an additional communication between computers we are able to determine it.

Additional communication between processes is also needed to find out what tuples (s,φ) are minimal in *Undef* (line 18). Suppose a fixpoint has been reached. Each process P_i computes a set $\mathit{LocalyMinimal}_i$ of tuples that are minimal in the set for which $\mathcal{A}_i$ is undefined. When finished, it sends the set $\mathit{LocalyMinimalFormulas}_i = \{\varphi \in \{\mathbf{A}(\varphi_1 \mathbf{U} \varphi_2), \mathbf{E}(\varphi_1 \mathbf{U} \varphi_2)\} \mid \exists s \in S_i : (s,\varphi) \in \mathit{LocalyMinimal}_i\}$ to every other process and receives similar information from other processes. Using this information, each process is able to determine what tuples from $\mathit{LocalyMinimal}_i$ are minimal in *Undef*. (Notice that if a tuple is not in $\mathit{LocalyMinimal}_i$, then it cannot be minimal in *Undef*).

To improve the performance of the algorithm, we can make it stop exactly at the moment when $\mathcal{A}_{f(\hat{s})}(\hat{s},\varphi)$ is computed, i.e., there is no need to reach a fixpoint if we already have computed the desired information earlier.

4 Correctness of the Algorithm

In this section we state the correctness of the distributed algorithm, i.e., we show that the algorithm always halts and returns the value $\mathcal{A}_{f(\hat{s})}(\hat{s},\psi)$ which equals to $\mathcal{C}_M(\mathcal{A}_\perp)(\hat{s},\psi)$. We present the proof of the key lemma only, for other

proofs we refer to the full version of the paper [4].

The first lemma states that after the first stage has been completed all the so far computed assumptions are correct.

Lemma 4.1 *Assume the computation is at point I (line 7) and* $\forall i \in \{1, \ldots, n\}$, $s \in S, \varphi \in cl(\psi)$ *the following property holds:*

$$\mathcal{A}_i(s, \varphi) \neq \bot \text{ implies } \mathcal{A}_i(s, \varphi) \text{ is correct (w.r.t. } M \text{ and } \mathcal{A}_\bot)$$

Then this property holds at point II (line 16) as well.

The next Lemma expresses the property that assumptions with defined values assign truth values in a uniform way.

Lemma 4.2 *Assume the algorithm is at point II and* $\forall i \in \{1, \ldots, n\}, \bar{s} \in S, \bar{\varphi} \in cl(\psi)$ *it holds that* $\mathcal{A}_i(\bar{s}, \bar{\varphi}) \neq \bot$ *implies* $\mathcal{A}_i(\bar{s}, \bar{\varphi})$ *is correct (w.r.t.* M *and* $\mathcal{A}_\bot$*). Let* $s \in S, \varphi \in cl(\psi)$ *s.t.* $(s, \varphi) \in Def$*. Then either*

$$\forall i \in \{1, \ldots, n\} : s \in K_i \;\Rightarrow\; \mathcal{A}_i(s, \varphi) = \mathbf{true}$$

or

$$\forall i \in \{1, \ldots, n\} : s \in K_i \;\Rightarrow\; \mathcal{A}_i(s, \varphi) = \mathbf{false}$$

Lemma 4.3 states the key idea of the distributed algorithm. After reaching fixpoint in the distributed computation, there still may exists a tuple in *Undef* meaning that for some state and some formula the assumption value is still undefined. This is the case of formulas containing the **U** operator. In the node algorithm, when stating that an **U**-formula does not hold, the greatest fixpoint is computed. But to compute the greatest fixpoint properly it is necessary to explore the entire state space, which is not possible in the distributed environment (the node algorithm has access to a part of the entire system only). On the other hand, when stating that an **U**-formula holds, the least fixpoint is computed, and it is possible to perform such a computation only on a part of the state space iteratively in a manner the distributed algorithm works. Therefore, if there is a tuple in *Undef* such that it is minimal in this set when reaching fixpoint of the distributed computation, the formula does not hold in the state in the entire system. We give a full proof of the claim.

Lemma 4.3 *Assume the computation of the algorithm is at point II and* $\forall i \in \{1, \ldots, n\}, \bar{s} \in S, \bar{\varphi} \in cl(\psi)$ *it holds that* $\mathcal{A}_i(\bar{s}, \bar{\varphi}) \neq \bot$ *implies* $\mathcal{A}_i(\bar{s}, \bar{\varphi})$ *is correct (w.r.t.* M *and* $\mathcal{A}_\bot$*). Assume also that there exists* $s \in S, \varphi \in cl(\psi)$ *such that* (s, φ) *is minimal in Undef. Then* $\mathcal{C}_M(\mathcal{A}_\bot)(s, \varphi) = \mathbf{false}$*.*

Proof. We split the proof into several parts.

- We first show that the formula must be an **U**-formula, that is $\varphi = \mathbf{A}(\varphi_1 \,\mathbf{U}\, \varphi_2)$ or $\varphi = \mathbf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2)$. We prove the claim by contradiction. Suppose $\varphi \neq \mathbf{A}(\varphi_1 \,\mathbf{U}\, \varphi_2)$ and $\varphi \neq \mathbf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2)$. We show that in case there must exist $i \in \{1, \ldots, n\}$ such that the function $\mathcal{A}_i(s, \varphi)$ can be computed, which is a contradiction to the assumption of reaching fixpoint.

Notice that $\forall s' \in S, \forall \xi \in cl(\varphi), \xi \neq \varphi$ it holds that $(s', \xi) \in Def$ as (s, φ) is minimal in $Undef$.

Let $i = f(s)$, meaning s is original in K_i. Because M is total we have $s \notin border(K_i)$.

- Let $\varphi = p$. Then it can be computed trivially.
- Let $\varphi = \neg\varphi_1$ or $\varphi = \varphi_1 \wedge \varphi_2$. Then $\mathcal{C}_{K_i}(\mathcal{A}_i)(s, \varphi)$ can be computed because $\mathcal{A}_i(s, \varphi_j)$ is defined for $j = 1, 2$.
- Let $\varphi = \mathbf{EX}\varphi_1$ or $\varphi = \mathbf{AX}\varphi_1$. Let $s_1, \ldots, s_t$ are all successors of s in M. As s is original in K_i we have $s_1, \ldots, s_t \in K_i$. Furthermore, $\mathcal{A}_i(s_j, \varphi_1)$ is defined for all $j \in \{1, \ldots, t\}$, and it follows that $\mathcal{C}_{K_i}(\mathcal{A}_i)(s, \varphi)$ can be computed.

- We now show that if $\varphi = \mathbf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2)$ and $\mathcal{C}_M(\mathcal{A}_\perp)(s, \varphi) = \mathbf{true}$ then there exists $s' \in S$ such that $(s', \varphi) \in Undef$ can be computed. As we assume that the fixpoint has been reached we conclude that $\mathcal{C}_M(\mathcal{A}_\perp)(s, \varphi) = \mathbf{false}$.

  From the fact that M is total we have that $\mathcal{C}_M(\mathcal{A}_\perp)(s, \varphi) = \mathbf{true}$ iff there exists a path $\pi = s_0 s_1 s_2 \ldots$ with $s = s_0$ and $x < |\pi|$: such that $\mathcal{C}_M(\mathcal{A}_{in})(s_x, \varphi_2) = \mathbf{true}$ and $\forall 0 \leq y < x :\ \mathcal{C}_M(\mathcal{A}_{in})(s_y, \varphi_1) = \mathbf{true}$.

  Let k be the greatest number such that $k \leq x$ and $(s_k, \varphi) \in Undef$. Such a number exists for we have assumed that $(s_0, \varphi) \in Undef$ and certainly $0 \leq x$. Let $i = f(s_k)$ is the identification of the process for which s_k is original.

  Suppose $k = x$. From the fact that (s, φ) is minimal in $Undef$ we have that $(s_k, \varphi_2) \in Def$. From Lemma 4.2 we conclude that $\mathcal{A}_i(s_k, \varphi_2) \neq \perp$ and from the assumption that the already computed values are correct we finally have that $\mathcal{A}_i(s_k, \varphi_2) = \mathbf{true}$. This allows to compute that $\mathcal{C}_{K_i}(\mathcal{A}_i)(s_k, \varphi) = \mathbf{true}$.

  Suppose $k < x$. We know that $s_{k+1} \in K_i$ and $(s_k, s_{k+1}) \in K_i$, for s_k is original in K_i. Moreover, $\mathcal{A}_i(s_{k+1}, \varphi) \neq \perp$ (from the maximality of k) and $\mathcal{A}_i(s_{k+1}, \varphi) = \mathbf{true}$ (from the assumption of correctness of computed values). Again, it allows to compute that $\mathcal{C}_{K_i}\mathcal{A}_i(s_k, \varphi) = \mathbf{true}$.

  In both cases we have a contradiction, therefore we can conclude that $\mathcal{C}_M(\mathcal{A}_\perp)(s, \varphi) = \mathbf{false}$.

- Now let $\varphi = \mathbf{A}(\varphi_1 \,\mathbf{U}\, \varphi_2)$. We will follow similar ideas as in the previous case. Let us assume that $\mathcal{C}_M(\mathcal{A}_\perp)(s, \varphi) = \mathbf{true}$ and we will show a contradiction with reaching a fixpoint.

  Using the totality of M we have that $\mathcal{C}_M(\mathcal{A}_\perp)(s, \varphi) = \mathbf{true}$ iff for all paths $\pi = s_0 s_1 s_2 \ldots$ with $s = s_0$ there exists $x < |\pi|$ such that $\mathcal{C}_M(\mathcal{A}_\perp)(s_x, \varphi_2) = \mathbf{true}$ and $\forall 0 \leq y < x :\ \mathcal{C}_M(\mathcal{A}_\perp)(s_y, \varphi_1) = \mathbf{true}$.

  Recall the standard sequential model checking algorithm (see for example [6]) of computing universal until using fixpoint on M. The states that satisfies $\mathbf{A}(\varphi_1 \,\mathbf{U}\, \varphi_2)$ are kept in a set H. First, all states that satisfies φ_2 are added to H. Then, repeatedly, a state is added to H if all its successors are already in H, until a fixpoint is reached. Let us choose a sequence $t_0, \ldots, t_p$

of states that would have been added to H by the algorithm such that
- $s = t_0$,
- $\forall a, b \in \{0, \ldots, p\}$, if $a < b$ then t_b would have been added to H before t_a,
- If a state would have been added to H before s, then it equals to t_c for some $c \in \{1, \ldots, p\}$.

Let $k \in \{0, \ldots, p\}$ is the greatest number so that $(t_k, \varphi) \in Undef$. Such a number exists, for we have assumed that $(t_0, \varphi) \in Undef$. Let $i = f(t_k)$ is the identification of the process for which t_k is original.

Let $r_1, \ldots, r_q$ are all successors of t_k in M. In K_i the state t_k has the same successors $r_1, \ldots, r_q$. As t_k would have been added to H by the sequential algorithm on M either $\mathcal{C}_M(\mathcal{A}_\perp)(t_k, \varphi_2) = \mathbf{true}$ or all successors $r_1, \ldots, r_q$ would have been added to H before t_k.

Let $\mathcal{C}_M(\mathcal{A}_\perp)(t_k, \varphi_2) = \mathbf{true}$. From the fact that $(t_k, \varphi_2) \in Def$ (follows from minimality of (s, φ) in $Undef$), Lemma 4.2 and the assumption of correctness of already computed values we have that $\mathcal{A}_i(t_k, \varphi_2) = \mathbf{true}$. This allows to compute that $\mathcal{C}_{K_i}(\mathcal{A}_i)(t_k, \varphi) = \mathbf{true}$.

In the second case, the fact that $r_1, \ldots, r_q$ would have been added to H before t_k means that every r_a, $a \in \{1, \ldots, q\}$ is contained between states $t_{k+1}, \ldots, t_p$, implying that $(r_a, \varphi) \in Def$ (we chose k to be the maximal number such that $(t_k, \varphi) \in Undef$). Lemma 4.2 and the assumption of correctness of already computed values gives that $\mathcal{A}_i(r_a, \varphi) = \mathbf{true}$ for all $a \in \{1, \ldots, q\}$. Hence, it can be computed that $\mathcal{C}_{K_i}(\mathcal{A}_i)(t_k, \varphi) = \mathbf{true}$.

Again, in both cases we have a contradiction, therefore we can conclude that $\mathcal{C}_M(\mathcal{A}_\perp)(s, \varphi) = \mathbf{false}$. ■

We conclude by the theorem which states the correctness of the distributed algorithm.

Theorem 4.4 *Let K_M^f be a partitioning of a total Kripke structure M according to the function f and ψ a formula. Then the distributed algorithm (Figure 5) returns the value which equals to $\mathcal{C}_M(\mathcal{A}_\perp)(\hat{s}, \psi)$.*

As a state in the initial system can be duplicated into several states in the distributed environment, the size of the state space may enlarge. It is shown below that the sum of the number of states of every node structure is at most equal to number of states plus number of transitions in the initial structure. In practice it may be much less – it depends on the partition function and the number of network nodes used.

Lemma 4.5 *Let $M = (S, R, I)$ be a Kripke structure, f a partition function. Furthermore, let $K_M^f = (K_1, \ldots, K_n)$ is the partitioning of M w.r.t. f, $K_i = (S_i, R_i, I_i)$. Then*

$$\sum_{i=1}^{n} |S_i| \;\leq\; |S| + |R|$$

5 Conclusions and Related Work

In this work we have considered a technique that uses assumptions about missing parts of the state space to perform CTL model checking in a distributed environment. We have developed the necessary theoretical background and described the distributed algorithm. The experimental version of the algorithm is currently being implemented.

One of the points that would certainly deserve at least some comments is how to chose the partitioning so as to minimize communications. For example if M is the model of a program we could choose to partition according to its structure (as done in [9]). If it is a hardware system the wise partitioning is probably according to a few bits that are known to change rarely. We expect to elaborate more possibilities in the future.

This work is to the best of our knowledge the first algorithm that uses a modular approach to distribute model checking. Closest to our work is the modular model checking approach by Yorav and Grumberg. In fact, the basic idea of the assumption function as defined here has been developed in their work. An other approach that utilises a decomposition of the system into parts (modules, fragments) is that by Burkart and Steffen [12]. They present a model checking algorithm for pushdown processes and consider the semantics of "fragments" which are interpreted as "incomplete portions" of the process. Another work where assumption functions have been considered is the model checking algorithm for the logic EF and CTL and pushdown processes developed by Walukiewicz [15]. Finally, in [5] the authors have used 3-valued logic (with $\perp$ representing "don't know if property is true or false") to reason about Kripke structures with partial labelling (called partial state space).

For the future work, our first goal is to perform an experimental evaluation. In particular we would like to find out how the performance is influenced by various types of partition function. We also intend to consider other logics and model checking algorithms in place of the "node algorithm".

References

[1] Barnat, J., L. Brim and J. Stříbrná, *Distributed LTL Model-Checking in SPIN*, in: M. B. Dwyer, editor, *Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN'01)*, LNCS **2057** (2001), pp. 217–234.

[2] Barnat, J., L. Brim and I. Černá, *Property Driven Distribution of Nested DFS*, in: *VCL'02, Pittsburgh, USA*, 2002.

[3] Berezin, S., S. Campos and E. M. Clarke, *Compositional reasoning in model checking*, in: *COMPOS*, 1997, pp. 81–102.

[4] Brim, L., J. Crhová and K. Yorav, *Using Assumptions to Distribute CTL Model Checking*, Technical Report FIMU-RS-2002-02, Faculty of Informatics, Masaryk

University Brno (2002).

[5] Bruns, G. and P. Godefroid, *Model checking partial state spaces with 3-valued temporal logics*, in: *Proc. 11th International Computer Aided Verification Conference*, 1999, pp. 274–287.

[6] Clarke, E. M., O. Grumberg and D. A. Peled, "Model Checking," The MIT Press, Cambridge, Massachusetts, 1999.

[7] Kupferman, O. and M. Vardi, *An automata-theoretic approach to modular model checking*, ACMTOPLAS: ACM Transactions on Programming Languages and Systems **22** (2000).

[8] Kupferman, O. and M. Y. Vardi, *Modular model checking*, in: *COMPOS*, 1997, pp. 381–401.

[9] Laster, K. and O. Grumberg, *Modular model checking of software*, in: *TACAS'98*, LNCS **1384**, 1998, pp. 20–35.

[10] Lerda, F. and R. Sisto, *Distributed-memory model checking with SPIN*, in: *Proceedings of the 6th International SPIN Workshop on Model Checking of Software (SPIN'99)*, LNCS **1680** (1999), pp. 22–39.

[11] Pierre-Alain Masson, J. J., Hassan Mountassir, *Modular verification for a class of PLTL properties*, LNCS **1945** (2000), pp. 398–419.

[12] Steffen, B. and O. Burkart, *Pushdown processes: Parallel composition and model checking*, in: *CONCUR'94*, LNCS **836** (1994), pp. 98–113.

[13] Tsay, Y.-K., *Compositional verification in linear-time temporal logic*, in: *FoSSaCS 2000*, 2000, pp. 344–358.

[14] U.Stern and D. L. Dill, *Parallelizing the murφ verifier*, in: O. Grumberg, editor, *Proceedings of Computer Aided Verification (CAV '97)*, LNCS **1254** (1997), pp. 256–267.

[15] Walukiewicz, I., *Model checking CTL properties of pushdown systems*, in: *Foundations of Software Technology and Theoretical Computer Science*, LNCS **1974**, 2000, pp. 127–138.

[16] Yorav, K., "Exploiting Syntactic Structure for Automatic Verification," Ph.D. thesis, Technion, Haifa, Israel (2000).