Efficient Algorithm for Reachability Checking in Modeling

Alexander Letichevsky[1], Olexander Letychevskyi[1], and Vladimir Peschanenko[2]

[1] Glushkov Institute of Cybernetics of NAS of Ukraine, 40 Glushkova ave., Kyiv, Ukraine
let@cyfra.net, lit@iss.org.ua
2 Kherson State University, 27, 40 Rokiv Zhovtnya str, Kherson, Ukraine
vladimirius@gmail.com

Abstract. The problem of reachability of the states of transition systems is considered hereby. The notions of partial unfolding and permutability of two operators (including the notion of statically permutable operators) are presented. New algorithm for reachability problem in terms of insertion modeling is described. An example of application of such algorithm is considered.

Keywords: insertion modeling, reachability, verification, interleaving, introduction

Key Terms: MathematicalModel, Process, Research.

1 Introduction

The verification of models of multiagent distributed systems and models of parallel computation usually need symbolic modeling and high level of abstraction. These models are highly nondeterministic because of symbolic nature of models and use of parallel composition adds a new level of non-determinism by interleaving. The main problem of verification is the problem of combinatorial explosion of states of the model. A state of model checking includes a lot of attributes and processes. The total number of states could be very large even if the number of processes is finite and all of the attributes are taken by finite number of values. Main source of combinatorial explosion of states are the number of processes and interleaving between them, nondeterministic behavior of them etc. Usually the systems are parallel and the number of their states grows exponentially with the number of processes. Our experience of verification of industrial systems shows that the total number of states is more than 21000. Obviously, model checking by naive enumeration of states is not feasible. Devoted to solving the problem many different technologies: developed methods for the partial order to reduce interleaving, used methods for determining the symmetry when verifying the equivalence of states, studied the information dependence to phase of verification components, applied techniques of abstraction, factorization, approximation, symbolic modeling etc. The standard model checking algorithms work only when the set of states reachable from the given initial state is

finite. Insertion modeling is a symbolic modeling with infinite number of states. There are various model checking techniques for infinite-state systems, but they are less developed than finite-state techniques and tend to place stronger limitations on the kind of systems and/or the properties that can be model checked. One of such techniques is presented in [1].

In Petri net theory there is well known the McMillan algorithm of unfolding that in many cases helps the exponential decreasing of interleaving in system analyses [2]. The book [3] generalizes this technique to the finite automata networks. So, the paper presents the new algorithm for reachability problem in terms of insertion modeling [4,5,6] for models with infinite number of states. The algorithm combines the ideas of economic unfolding of McMillan with on-line reachability checking using some general assumptions about the nature of information dependencies in the states of distributed system expressed in terms of permutability of actions.

So, the paper is devoted to the solution of a problem of reduce interleaving in insertion models with infinite number of states.

The algebra of behavior is presented in section Behavior Algebras, the verification environments and corresponded insertion function, predicate transformer are considered in section Verification Environments. The normal form of behavior is defined in section Behaviors Over Basis *B*. The problem of reachability of the states is described in section Verification. The notion of partial unfolding is examined in section Partial Unfolding. The optimization of partial unfolding by statically permutable operators is reviewed in section Static Permutability Property. The algorithm for reducing of interleaving for transitional systems is presented in section Algorithm of Reducing Interleaving. The example which demonstrates a good result of using the partial unfolding algorithm is presented in section Example of Application.

2 Behavior Algebras

Behavior algebra [5] is a kind of process algebra and it is used to express the behavior of agents (transition systems) considered up to bisimilarity or trace equivalence. To make economic unfolding we need to distinguish sequential and parallel behaviors. So we consider the following modification of the notion of behavior algebra. It is a multisorted algebra with three components: the algebra of *actions*, the algebra of *sequential behaviors*, and the algebra of *parallel behaviors*.

The algebra of sequential behaviors has operations of prefixing: *<action> . <sequential behavior>* and one internal operation of nondeterministic choice *(()+())* which is associative commutative and idempotent operation with neutral element 0. We also consider the constant behavior Δ (successful termination) which is the common element of the algebra of sequential and the algebra of parallel behaviors. The operations of action algebra will be considered later.

The algebra of parallel behaviors has the parallel composition *()||()* of sequential behaviors as the main binary operation. It is associative commutative (not idempotent) with the neutral element Δ. It also has the prefixing operation and nondeterministic choice. The algebra of sequential behaviors is implicitly included to

the algebra of parallel behaviors by identity $u = u \,||\, \Delta$ (parallel composition with one component). The unfolding of parallel composition by interleaving will be considered only after inserting the agents formed by parallel composition into the environment.

3 Verification Environments

These environments $E = E(U, P, B)$ are defined by the following parameters: the set of *conditional expressions U*, the set of *operators P*, and the set of *basic behaviors B*. The set of conditions and the set of operators are used to define actions (it is a union of these two sets). The set of basic behaviors are used to define the behaviors of agents inserted into environment in the way which will be explained later. We also suppose that some logic language (first order or temporal) called as basic language is fixed to define the states of environment and checking conditions for verification. The conditional expressions also belong to this language.

The state of environment is represented as $E[u]$, where *E* is a statement of basic language and *u* is a parallel composition of sequential behaviors of agents inserted into environment. We suppose that operators are divided into the set of conditional and unconditional operators. Conditional operator has the form $\alpha \to a$ where α is a condition and *a* is an unconditional operator. Unconditional operator *a* is identified with conditional operator $1 \to a$. The associative product *()*()* and the function $pt : U \times P \to U$ (predicate transformer) are defined by the set of actions so that the following identities are valid:

$$pt(\alpha, \beta \to a) = pt(\alpha \wedge \beta \to a)$$
$$pt(pt(\alpha, a), b) = pt(\alpha, a * b)$$
$$(\alpha \to a) * (\beta \to b) = pt(pt(\alpha, a) \wedge \beta, b)$$
$$\alpha * \beta = \alpha \wedge \beta$$

Here α and β are conditions, *a* and *b* are unconditional operators.

Predicate transformer is supposed to be monotonic: $\alpha \to \beta \Rightarrow pt(\alpha, a) \to pt(\beta, a)$. In general case, the *pt* function is defined by some concrete syntax. One of the examples of such pair *(syntax, pt)* could be found [7].

Example. Basic language is a first order language. Conditions are formulae over simple attributes - the symbols that change their values when system changes its state. Formally they are considered as functional symbols with arity 0. Unconditional operators are assignments (parallel assignments, sequences of assignments and if-then-else operators, loops with finite number of repetitions etc.). As usually in this case

$$pt(\alpha(x), (x_1 := t_1(x), x_2 := t_2(x), \ldots)) = \exists z(\alpha(z) \wedge (x_1 = t_1(z) \wedge x_2 = t_2(z) \wedge \ldots))$$

Actually this is the strongest postcondition for precondition α.

Insertion function is defined by the following identities and rules.

1. $E[u, v] = E[u \,||\, v]$, *u,v* are agents with sequential behavior (see sec. 1).

Identities for conditions.

2. $E[\alpha.u + v] = E[v]$, if $(E \wedge \alpha) = 0$.

3. $E[\alpha.\beta.u + v] = E[\alpha \wedge \beta.u + v]$, if $(E \wedge \alpha) \neq 0$ (merging conditions).

4. $E[\alpha.\beta \to a.u + v] = E[\alpha \wedge \beta \to a.u + v]$, if $(E \wedge \alpha \wedge \beta) \neq 0$. Special cases of these identities are obtained when v=0 or $\beta = 1$ as special cases.

5. $E[\alpha.\varepsilon] = E \wedge \alpha[\varepsilon]$, if $(E \wedge \alpha) = 0$.

Identities for operators.

6. $E[a.u + v] = E[v]$, if $pt(E,a) = 0$.

7. $E[a.u] = a.pt(E,a)[u \,||\, \varphi(a,E)]$, if $pt(E,a) \neq 0$, $\varphi(a,E)$ is a parallel composition of sequential behaviors (generating some new parallel branches). If $\varphi(a,E) = \Delta$, then $u \,||\, \varphi(a,E) = u \,||\, \Delta = u$ and u remains unchanged.

Nondeterministic choice.

8. $E[a.u + a.v + w] = E[a.(u + v) = w]$. The use of left distributivity means that environment considers behavior expressions up to trace equivalence. It also means that a system uses delayed (angelic) choice.

9. $E[u + \Delta] = E[u] + E[\Delta]$. The states $E[0]$ and $E[\Delta]$ are called *terminal states of environment*. Formally the states of the form $E[0]$ are equivalent to 0, and states of a form $E[\Delta]$ are equivalent to Δ (if the $E[\Delta] = E[\Delta] + \Delta$ is added). But from the point of view of verification it is useful to distinguish syntactically different terminal states.

Parallel behaviors.

10. $E[u] = E[v] \Rightarrow E[u \,||\, w] = E[v \,||\, w]$. Therefore all identities for conditions and operators can be applied within the parallel composition. A component $a_1.u_1 + \ldots + a_n.u_n$ of parallel composition is called *degenerated relative to the state E*, if for all operators $a_i.pt(E,a_i) = 0$ and for all conditions α_i it is true that $(E \wedge \alpha_i) = 0$. Each component that is degenerated relatively to the state E is equivalent to 0 relatively to this state.

11. $E[u] + F[v] = F[v]$, if parallel composition u contains degenerated component relative to E. So all states of environment with degenerated components are equivalent to 0.

12. $E[u + \Delta \,||\, v] = E[u \,||\, v] + E[v]$.

13. $E[a_1.u_1 + a_2.u_2 + \ldots] = E[a_1.u_1 \,||\, v] + E[a_2.u_2 \,||\, v] + \ldots$, if all actions a_i are different, if a_i is a condition then u_i is terminal constant, and v does not contain components degenerated relatively to the state E. The state of environment $E[u]$ is called *dead lock state*, if there are no transitions from this state, but u is not successful termination. If there is at least one degenerated component in parallel composition, then corresponding state is a dead lock state and all dead lock states are equivalent to 0, but it is useful to distinguish them as well as terminal constants. The rules (9), (12), and (13) are called *unfolding of nondeterministic choice*.

14. $E[a_1.u_1 \,||\, \ldots \,||\, a_n.u_n] = \sum_{i=1}^{n} a_i.(\ldots \,||\, a_{i-1}.u_{i-1} \,||\, u_i \,||\, a_{i+1}.u_{i+1} \,||\, \ldots)$, if all components of parallel composition are non-degenerated. This relation is called *unfolding of a parallel composition*. This is a complete unfolding and the main result

of this paper shows that it is not needed to make the complete unfolding at each step of verification. Let be $u = a_1.u_1 \| \ldots \| a_n.u_n$,

$$unfold(u,i) = a_i.(\ldots \| a_{i-1}.u_{i-1} \| u_i \| a_{i+1}.u_{i+1} \| \ldots)$$

The identity (14) can be rewritten as

14a. $E[a_1.u_1 \| \ldots \| a_n.u_n] = \sum_{i=1}^{n} unfold(u,i)$.

Environment does not distinguish trace equivalent behaviors and consequently bisimilar states of environment are trace equivalent. The identity (14) defines the main transition rule for the system:

$$E[u] \xrightarrow{a_i} E'[u'],$$

if *u* is a parallel composition with non-degenerated components and $E'[u']$ is defined by the identity (7).

4 Behaviors Over Basis B

The set of symbols is given for the set *B* of behavior basis. These symbols are called *basic sequential behaviors*. The expression of the algebra of sequential behaviors constructed from these symbols and termination constants are called *sequential behavior over basis B*. Suppose that for each symbol $v \in B$ an equation of the form $v = F_v(v_1, v_2, \ldots)$ is given with sequential behavior over basis *B* as a right hand side. This equation is called the *definition of a basic behavior v*. The application of this definition (the substitution of the right hand side for the left hand one) is called the *unfolding of this behavior*. System of basic behaviors is called non-degenerated if each path in the tree representation of the expression $v = F_v(v_1, v_2, \ldots)$ contains at least one operator.

Normal form of sequential behavior is an expression of the form $a_1.u_1 + a_2.u_2 + \ldots + a_n.u_n + \varepsilon$ where $u_1, u_2, \ldots$ are sequential behaviors. If a_i is a condition, then u_i is a termination constant, $n \geq 0$, and all actions are different (not equivalent with respect to the environment *E*), because of delayed (angelic) choice (see sec. 2).

Each sequential behavior u over non-degenerated basis in a state $E[u]$ *can be reduced to a normal form v equivalent to u with respect to E.*

Parallel behavior over B is a parallel composition of sequential behaviors over *B*.

Normal form of parallel behavior is a nondeterministic sum of behaviors of the form $a_1.u_1 + a_2.u_2 + \ldots$, where $u_1, u_2, \ldots$ are sequential behaviors over *B*, $a_1, a_2, \ldots$ - operators or conditions at what if a_i is a condition, then u_i is a termination constant.

Normal form of environment state is a term of a form $\sum_{i \in I} a_i.E_i[u_i] + \sum_{j \in J} \Delta$ or 0. *Each environment state with non-degenerated system of basic behaviors is trace equivalent to some normal form.*

5 Verification

A property ξ of environment state is called to be *correct* if it does not distinguish equivalent states. A property ξ of environment state is monotonic if $E \to E' \Rightarrow \xi(E[u]) \to \xi(E'[u])$.

5.1 Verification problem

For a given set Ξ of correct and monotonic checked properties, defined on the set of environment states, the set of initial states defines which properties are reachable(not reachable) from the initial states for a finite number of steps or a number of steps bounded by some constant. It is supposed that the set of checked properties contains the property of a state to be dead lock and to be a state of successful termination.

The simplest verification algorithm is exhaustive unfolding of initial states up to saturation or depletion of a given number of steps. It uses the next formulae of unfolding: $\sum_{i=1}^{n} E[unfold(u,i)]$. The checked properties are checked in the process of unfolding and the states satisfying checked properties are collected. More economic algorithm can be constructed using the following *partial unfolding algorithm*.

6 Partial Unfolding

Two operators a and a' are called *permutable with respect to the state E*, if $E[a * a'] = E[a' * a](a \xleftrightarrow{E} a')$. Let the state of environment $E[u] = E[a_1.u_1 \parallel \ldots \parallel a_n.u_n]$ be given. Select a component $s = a_i.u_i$ and construct for this component the set $nonp(E,s)$ of those components $a_j.u_j, i \neq j$ such that a_i and a_j are not permutable with respect to the current state *E*. Obtain

$punfold(E,u,i) = A(i) + B(E,i) + C(E,i)$

$A(i) = a_i.(\ldots \parallel a_{i-1}.u_{i-1} \parallel u_i \parallel a_{i+1}.u_{i+1} \parallel \ldots)$

$B(E,i) = \sum_{i \neq j \wedge (a_i,a_j) \in nonp(E,s)} a_j.(\ldots \parallel a_{j-1}.u_{j-1} \parallel u_j \parallel a_{j+1}.u_{j+1} \parallel \ldots)$

$C(E,i) =$

$= \sum_{k \neq i \wedge (a_k,a_i) \notin nonp(E,s) \wedge a_k \xleftrightarrow{E} a_w} a_k.(\ldots \parallel a_{k-1}.u_{k-1} \parallel a_k \ldots a_w.u_k \parallel a_{k+1}.u_{k+1} \parallel \ldots)$

An expression $E[punfold(E,u,i)]$ is called a *partial unfolding of parallel composition* by the component *i* (unlike the complete unfolding).

In general case, the algorithm uses dynamic permutablity of operators, but it isn't optimal because using four times of pt function for each pair of the operators is required. This problem could be improved by using the notion of statically permutable operators.

6.1 Static Permutability Property

Theorem 1. If two operators $p = \alpha \to a, q = \beta \to b$ are permutable with respect to the states $E_1 = \alpha \wedge \beta$, $E_2 = \neg\alpha \wedge \beta$, $E_3 = \alpha \wedge \neg\beta$ then they are permutable for all states.

Assume the contrary that $\exists e(e[p*q] \neq e[q*p])$ and $E_1[p*q] = E_1[q*p],\ E_2[p*q] = E_2[q*p],\ E_3[p*q] = E_3[q*p]$. Consider $E_1[p*q] = E_1[q*p]$:

$$(\alpha \wedge \beta[p*q] \Rightarrow pt(\alpha \wedge \beta \wedge \alpha, a)[p] = pt(\beta \wedge pt(\alpha \wedge \beta, a), b)) \wedge$$
$$\wedge(\alpha \wedge \beta[q*p] \Rightarrow pt(\alpha \wedge \beta \wedge \beta, b)[q] = pt(\alpha \wedge pt(\alpha \wedge \beta, b), a)) \wedge$$
$$\wedge(\alpha \wedge \beta[p*q] = \alpha \wedge \beta[q*p]) \Rightarrow$$
$$\Rightarrow pt(\beta \wedge pt(\alpha \wedge \beta, a), b) = pt(\alpha \wedge pt(\alpha \wedge \beta, b), a) \Rightarrow$$
$$\Rightarrow pt(\beta \wedge pt(\alpha \wedge \beta \wedge (e \vee \neg e), a), b) = pt(\alpha \wedge pt(\alpha \wedge \beta \wedge (e \vee \neg e), b), a) \Rightarrow$$
$$\Rightarrow pt(\beta \wedge pt(\alpha \wedge \beta \wedge e), a), b) \vee pt(\beta \wedge pt(\alpha \wedge \beta \wedge \neg e), a), b) =$$
$$= pt(\alpha \wedge pt(\alpha \wedge \beta \wedge e, b), a) \vee pt(\alpha \wedge pt(\alpha \wedge \beta \wedge \neg e, b), a)$$

Consider $E_2[p*q] = E_2[q*p]$:

$$(\neg\alpha \wedge \beta[p*q] \Rightarrow pt(\alpha \wedge \neg\alpha \wedge \beta, a)[q] \Rightarrow 0) \wedge$$
$$\wedge(\neg\alpha \wedge \beta[q*p] \Rightarrow pt(\beta \wedge \neg\alpha \wedge \beta, b)[p] \Rightarrow pt(\alpha \wedge pt(\neg\alpha \wedge \beta, b), a)) \wedge$$
$$\wedge(\neg\alpha \wedge \beta[p*q] = \neg\alpha \wedge \beta[q*p]) \Rightarrow$$
$$\Rightarrow (pt(\alpha \wedge pt(\neg\alpha \wedge \beta, b), a) = 0)$$

Consider $E_3[p*q] = E_3[q*p]$:

$$(\alpha \wedge \neg\beta[q*p] \Rightarrow pt(\beta \wedge \alpha \wedge \neg\beta, b)[q] \Rightarrow 0) \wedge$$
$$\wedge(\alpha \wedge \neg\beta[p*q] \Rightarrow pt(\alpha \wedge \alpha \wedge \neg\beta, b)[q] \Rightarrow pt(\beta \wedge pt(\alpha \wedge \neg\beta, a), b)) \wedge$$
$$\wedge(\alpha \wedge \neg\beta[p*q] = \alpha \wedge \neg\beta[q*p]) \Rightarrow$$
$$\Rightarrow (pt(\beta \wedge pt(\alpha \wedge \neg\beta, a), b) = 0)$$

Let's try to prove this theorem by contradiction. Let's consider insertion of two operators *p,q*:

$$e[p*q] = e[(\alpha \to a)*(\beta \to b)] \Rightarrow pt(e \wedge \alpha, a)[\beta \to b] \Rightarrow pt(\beta \wedge pt(e \wedge \alpha, a), b)$$
$$e[q*p] = e[(\beta \to b)*(\alpha \to a)] \Rightarrow pt(e \wedge \beta, b)[\alpha \to a] \Rightarrow pt(\alpha \wedge pt(e \wedge \beta, b), a)$$

So, $\exists e(pt(\beta \wedge pt(e \wedge \alpha, a), b) \neq pt(\alpha \wedge pt(e \wedge \beta, b), a))$.

$$\exists e(pt(\beta \wedge pt(e \wedge \alpha \wedge (\beta \vee \neg\beta), a), b) \neq pt(\alpha \wedge pt(e \wedge \beta \wedge (\alpha \vee \neg\alpha), b), a)) \Rightarrow$$
$$\Rightarrow \exists e(pt(\beta \wedge pt(e \wedge \alpha \wedge \beta \vee e \wedge \alpha \wedge \neg\beta, a), b) \neq$$
$$\neq pt(\alpha \wedge pt(e \wedge \beta \wedge \alpha \vee e \wedge \beta \wedge \neg\alpha, b), a)) \Rightarrow$$
$$\Rightarrow \exists e(pt(\beta \wedge pt(e \wedge \alpha \wedge \beta, a), b) \vee pt(\beta \wedge pt(e \wedge \alpha \wedge \neg\beta, a), b) \neq$$
$$\neq pt(\alpha \wedge pt(e \wedge \alpha \wedge \beta, b), a) \vee pt(\alpha \wedge pt(e \wedge \neg\alpha \wedge \beta, b), a)) \Rightarrow$$

So, using monotonic property of *pt* function obtain

$$e \wedge \neg\alpha \wedge \beta \to \neg\alpha \wedge \beta \Rightarrow pt(e \wedge \neg\alpha \wedge \beta, b) \to pt(\neg\alpha \wedge \beta, b) \Rightarrow$$
$$\Rightarrow pt(\alpha \wedge pt(e \wedge \neg\alpha \wedge \beta, b), a) \to pt(\alpha \wedge pt(\neg\alpha \wedge \beta, b), a) \Rightarrow$$
$$\Rightarrow pt(\alpha \wedge pt(e \wedge \neg\alpha \wedge \beta, b), a) \to 0$$
$$e \wedge \alpha \wedge \neg\beta \to \alpha \wedge \neg\beta \Rightarrow pt(e \wedge \alpha \wedge \neg\beta, a) \to pt(\alpha \wedge \neg\beta, a) \Rightarrow$$
$$\Rightarrow pt(\beta \wedge pt(e \wedge \alpha \wedge \neg\beta, a), b) \to pt(\beta \wedge pt(\alpha \wedge \neg\beta, a), b) \Rightarrow$$
$$\Rightarrow pt(\beta \wedge pt(e \wedge \alpha \wedge \neg\beta, a), b) \to 0$$

It means that

$$\exists e(pt(\beta \wedge pt(e \wedge \alpha \wedge \beta, a), b) \neq pt(\alpha \wedge pt(e \wedge \alpha \wedge \beta, b), a)) \wedge$$
$$\wedge (E_1[p * q] = E_1[q * p]) \Rightarrow$$
$$\Rightarrow \exists e((pt(\alpha \wedge pt(e \wedge \alpha \wedge \beta, b), a) = pt(\beta \wedge pt(\neg e \wedge \alpha \wedge \beta, a), b)) \wedge$$
$$\wedge (pt(\beta \wedge pt(e \wedge \alpha \wedge \beta, a), b) = pt(\alpha \wedge pt(\neg e \wedge \alpha \wedge \beta, b), a)))$$

Let consider the cases when

$$\exists e(pt(\alpha \wedge pt(e \wedge \alpha \wedge \beta, b), a) = pt(\beta \wedge pt(\neg e \wedge \alpha \wedge \beta, a), b))$$

$e \wedge \alpha \wedge \beta \wedge \neg e \wedge \alpha \wedge \beta = 0$ means that the *pt* function translates this formulae into one state independently from *e* and $\neg e$. It is possible then all attributes expressions from *e* are in *r,s* list of *pt* and after application of both protocols *p,q* no restrictions are left from *e* and $\neg e$, because of contradiction in other cases. It means that both operators *p,q* translate all sub-formulae which depend on attribute expressions from *e* into one sub-formulae, independently from sequence of application. So, by obtaining a contradiction, due to that case $pt(\alpha \wedge pt(e \wedge \beta, b), a) = pt(\beta \wedge pt(e \wedge \alpha, a), b)$, theorem is proved.

Two operators $p = \alpha \to a, q = \beta \to b$ are called *statically permutable* if they satisfy the next conditions:

1. 1. $pt(\alpha \wedge pt(\alpha \wedge \beta, b), a) = pt(\beta \wedge pt(\alpha \wedge \beta, a), b)$
2. 2. $pt(\alpha \wedge pt(\neg\alpha \wedge \beta, b), a) = 0$
3. 3. $pt(\beta \wedge pt(\alpha \wedge \neg\beta, a), b) = 0$

From practical point of view, to make 8 calls of *pt* function to check the static permutability property for two operators is a slow process. So, let's define the weak property for static permutability of two operators.

Let *r(p),s(p),z(p)* be the lists of predicate transformer for operator $p = \alpha \to a$, where *r(p)* - the list of the attribute expressions from left part of assignment of *a*, *r(p)* - the list of the attribute expressions from the formulae part of *a*, *z(p)* - the list of attribute expressions from α which are not in $r(p) \cup s(p)$ [7].

Two operators $p = \alpha \to a, q = \beta \to b$ are called *weak static permutable* if

$$((r(p) \cup s(p)) \cap (r(q) \cup s(q) \cup z(q)) = \varnothing) \wedge$$
$$\wedge ((r(q) \cup s(q)) \cap (r(p) \cup s(p) \cup z(p)) = \varnothing)$$

Theorem 2. If two operators $p = \alpha \to a, q = \beta \to b$ satisfy weak condition of static permutability then they are statically permutable

If *p, q* satisfy weak condition of static permutability then both operators work with different memory. It means that all conditions for static permutability are satisfied and the theorem proved.

7 Algorithm of Reducing Interleaving

Using of the Breadth-first search (BFS) algorithm is better then the Depth-first search (DFS) algorithm for checking of reachability property. But, in general case, our algorithm could be used with any traversal strategy. So, let use BFS algorithm and the component which was chosen is in normal form $E[a_1.u_1 \| a_2.u_2 \| \ldots], s_i = a_i.u_i$. Then

component $nonp(s_j) = \min_i nonp(s_i)$ is chosen for partial unfolding $punfold(a_1.u_1 \| a_2.u_2 \| \ldots, j)$, $j \in \{1,2,\ldots\}$. Of course the algorithms for visited and dead lock detection should be defined for partial unfolding.

The state in the set of search tree is considered as visited if it is in the set of already visited states and all its possible successors are in this set:

$E[a_1.(u_1 \| a_2.u_2 \| \ldots \| a_i.u_i \| \ldots \| a_n.u_n)] - visited$

…

$E[a_i.(a_1.u_1 \| \ldots \| a_{i-1}.u_{i-1} \| u_i \| a_{i+1}.u_{i+1} \| \ldots \| a_n.u_n)] - visited$

…

$E[a_n.(a_1.u_1 \| \ldots \| a_{n-1}.u_{n-1} \| u_n)] - visited$

The state in the set of search tree is considered as dead lock if some of inserted actions gives 0:

$E[a_1.(u_1 \| a_2.u_2 \| \ldots \| a_i.u_i \| \ldots \| a_n.u_n)] \Rightarrow 0$

…

$E[a_i.(a_1.u_1 \| \ldots \| a_{i-1}.u_{i-1} \| u_i \| a_{i+1}.u_{i+1} \| \ldots \| a_n.u_n)] \Rightarrow 0$

…

$E[a_n.(a_1.u_1 \| \ldots \| a_{n-1}.u_{n-1} \| u_n)] \Rightarrow 0$

In general case the partial unfolding loses states, because $punfold(u,i) \neq unfold(u,i)$.

Theorem 3. The function $punfold(u,i)$ doesn't break reachability property if algorithm checks reachability property for lost states after partial unfolding.

First of all the partial unfolding doesn't break the reachability of terminal states: 0, $\Delta, \perp$, because of definition of dead lock state, $u \| \Delta = u$, $u \| \perp = \perp$. Algorithm for visited doesn't break reachability property because of definition which helps system to stop the consideration of infinite number of states.

Finally, let's check the reachability property for goal and safety detection algorithm. The partial unfolder $punfold(u,i)$ loses the states for components of parallel composition $a_j.u_j$ which are considered as permutable for current action $(a_i, a_j) \notin nonp(s_i)$ and $\forall (a'_j \in u_j)((a_i, a'_j) \notin nonp(s_i))$. In that case *punfold* loses states $a_j.E'[a_1.u_1 \| \ldots \| u_j \| \ldots \| a_n.u_n]$. It means that the algorithm should check the reachability property here. From other point of view, it's known that $E[a_i * a_j] = E[a_j * a_i]$, because $(a_i, a_j) \notin nonp(s_i)$. It means that the reachability property after insertion of a_j operator will be saved and after sequential insertion of a_i, a_j. So, the theorem is proved.

If the algorithm of partial unfolding is used for checking reachability property of goal and safety states then algorithm should check those states after each work of function *punfold*.

8 Example of Application

Let initial state be $E[R20 \| U312]$. The behavior R20 and U312 is defined by the following graphs fig. 1, fig. 2 respectively.

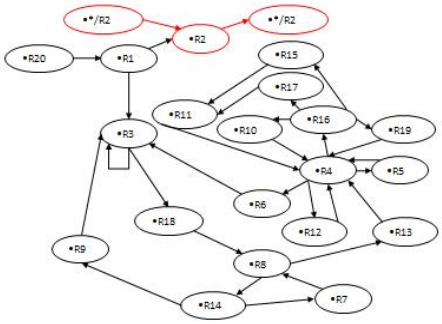

Fig. 1. Behavior for R20.

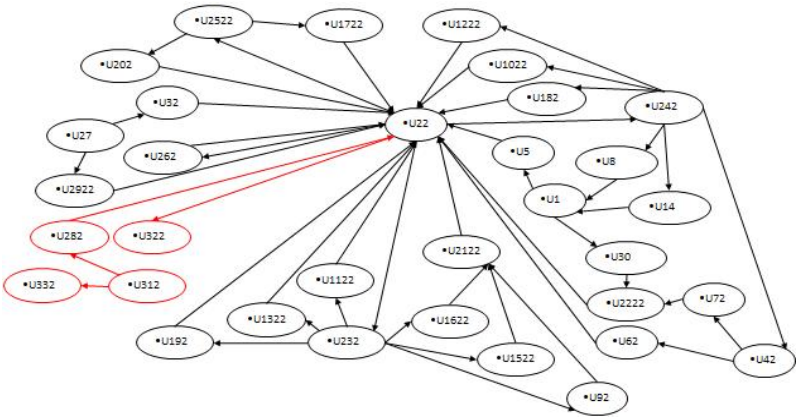

Fig. 2. Behavior for U312.

The **/R2* means all operators except *R2*, the red color in fig. 1 means the special situation when the *R2* could be inserted after any of operators except *R2* and that after insertion of *R2* any protocol could be inserted except *R2*. The red color on fig. 2 used to mark sub-path for successful termination state (terminal states).

The following results are obtained after analyzing the two lists of operators from behavior *R20* and *U312*:

1. Behavior *R20* has 5 from 20 operators which are statically permutable for all operators from *U312*.
2. Behavior *U312* has 31 from 33 operators which are statically permutable for all operators from *R20*.
3. The time for creating the list of statically permutable operators is aprox. 1 min.
4. The whole state space was covered after aprox. 25 min.
5. Total number of covered states is 1102.

Out of memory error was obtained without partial unfolding algorithm after aprox. 1 hour of work and aprox. 50000 of states were covered. So, it is good results for this example, because if one of operators is statically permutable for all of the operators from other parallel processes then $punfold(u,i) = A(i)$, because $B(i) = C(i) = 0$ here.

9 Conclusions

The verification algorithm based on partial unfolding has been implemented in the system of insertion modeling IMS and has shown considerable decreasing of the verification time on this example. Generally speaking, the *C++* version (the language of implementation of IMS) of this algorithm will be faster not less than 10 times. It is true because of well known notion of IMS: *C++* algorithm is faster not less in 10 times than corresponded prototype (which was written on APLAN language – the language of IMS) of this algorithm in IMS. We hope this algorithm will be fast for the similar examples as well.

In the near future this algorithm will be applied for the verification of set of industrial examples, for verification of parallel programs [8], and for VFS (verification of formal specification) - our tool for symbolic modeling with infinite number of states.

References

1. Escobar, S., Meseguer, J.: Symbolic Model Checking of Infinite-State Systems Using Narrowing. Proceedings of the 18th International Conference on Term Rewriting and Applications, LNCS 4533, pp. 153--168, Springer (2007).
2. McMillan, K.L.: Trace Theoretic Verification of Asynchronous Circuits Using Unfoldings. Proceedings of the 7th Workshop on Computer Aided Verification, Liege, LNCS 939, pp. 180-195, Springer (1995)
3. Esparza, J. and Heljanko, K.: Unfoldings - A Partial-Order Approach to Model Checking. EATCS Monographs in Theoretical Computer Science, ISBN: 978-3-540-77425-9, Springer-Verlag,172 p. (2008)
4. Letichevsky, A., Gilbert, D.: A Model for Interaction of Agents and Environments. In D. Bert, C. Choppy, P. Moses, editors. Recent Trends in Algebraic Development Techniques. Lecture Notes in Computer Science 1827, Springer (1999)
5. Letichevsky, A.: Algebra of behavior transformations and its applications, in V.B.Kudryavtsev, I.G.Rosenberg (eds.) Structural theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry - Vol. 207, pp. 241-272, Springer (2005)
6. Letichevsky, A., Kapitonova, J., Kotlyarov, V., Letichevsky Jr., A., Nikitchenko, N., Volkov, V., Weigert, T.: Insertion modeling in distributed system design, Problems of programming (ISSN 1727-4907), Vol. 4, pp. 13-39 (2008)
7. Letichevsky, A.A., Godlevsky, A.B., Letichevsky, A.A., Jr., Potienko, S.V., Peschanenko, V.S.: Properties of Predicate Transformer of VRS System. Cybernetics and System Analyses, 4:3-16 (2010) (in Russian)
8. Letichevsky, A., Letychevskyi, O., Peschanenko, V.: Insertion Modeling System, PSI 2011, Lecture Notes in Computer Science, Vol.7162, pp. 262-274. Springer (2011)