An Interleaving Reduction for Reachability Checking in Symbolic Modeling

Alexander Letichevsky[1], Oleksandr Letychevskyi[1], Vladimir Peschanenko[2]

[1]Glushkov Institute of Cybernetics of National Academy of Sciences, Kyiv, Ukraine
{let,lit}@iss.org.ua
[2]Kherson State University, Kherson, Ukraine
vladim@kspu.edu

Abstract. This paper is devoted to the whole problem of interleaving reduction in modeling of concurrent processes. The main notions of insertional modeling were described. The verification problem in terms of insertional modeling was examined. General algorithm of interleaving reduction in terms of insertional modeling was presented. A static and incremental algorithm of reduction for reachability checking was presented. The proof of correctness of presented algorithm was introduced. The results of experiments of such algorithm application was described.

Keywords. Interleaving, predicate transformer, symbolic modeling.

Key Terms. MathematicalModel.

1 Introduction

Usually the multiagent distributed systems are high level non-deterministic. The nature of this non-determinism is symbolic nature of models and concurrency (choice of parallel process which should operate at each time of modeling). One of the main problem of reachability checking in verification is exponential explosion of states number. Some of the sources of such explosion is the number of parallel processes in model and their interleaving[1] .

There are two different approaches for modeling: model checking and symbolic modeling[2]. The model checking tool works with concrete states where state is represented by values of its variables. A transition is occurred by assignment of new values for the variables. The problem of exponential explosion could be solved by using well known model checking methods: methods that introduce partial order to reduce interleaving[3], methods for determining the symmetry when verifying the equivalence of states[4], techniques of abstraction[5], approximation[6], data-flow analyses[7], McMillan's algorithm of unfolding[8].

A state of environment in symbolic modeling presents some formula in corresponded theory (first order logic etc) which covers some set of concrete states. A transition is occurred with a help of predicate transformers (weakest precondition, strongest postcondition[9])[10]. Unfortunately not all methods of model checking for

reducing states space could be applied for symbolic case. The problem which was described previously could be solved with a help of the next symbolic methods: narrowing[11], unfolding concurrent well-structured transition systems[12].

This paper continues the work [13] where an algorithm with some restriction of symbolic model was described. Here we present the algorithm for full symbolic case. The algorithm bases on the McMillan's algorithm adopted to symbolic modeling in notion of insertion modeling [14]. This algorithm bases on notion of permutability which is defined with help of predicate transformer (strongest postcondition, *pt* function below). It was described in [10]. So, the paper is devoted to the solution of the problem of interleaving reduction in insertion models with infinite number of states.

The algebra of behaviors is presented in chapter Behavior Algebras, the verification environments, corresponding insertion function, and predicate transformer are considered in chapter Verification Environments. The normal form of behavior is defined in chapter Behaviors Over Basis *B*. The problem of reachability of the states is described in chapter Verification. The notion of partial unfolding is examined in chapter Partial Unfolding. The optimization of partial unfolding by statically permutable operators is reviewed in chapter Static Permutability Property. The incremental algorithm for reducing of interleaving for transition systems is presented in chapter Main Interleaving Reduction Algorithm. The static algorithm of interleaving reduction is described in chapter Static Interleaving Reduction Algorithm. The statistic of applying of such algorithm to few examples is presented in chapter Examples of Application.

2 Behavior Algebras

One of the main notions of insertion modeling, which is used for describing algorithm of interleaving reduction is behavior algebra. Behavior algebra [14] is a kind of process algebra; 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 a 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 (but is not idempotent) and has 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 the identity $u = u||\Delta$ (parallel composition with one component). Unfolding of parallel composition by interleaving will be considered

only after inserting of agents that are formed by parallel composition into the environment.

3 Verification Environments

Verification environments of the form $E = E(U, P, B)$ 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 is 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 *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 *pt* 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. An example of such pair *(syntax, pt)* can be found in [16].

Example. The basic language is a first order language. Conditions are formulae over simple attributes - symbols that change their values when a system changes its state. Formally they are considered as function symbols with arity 0. Unconditional operators are assignments (parallel assignments, sequences of assignments, 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.

Example of conditional operator. Let x be an integer variable, $u = (x < 5) \to (x := x + 1)$ be an operator, $x > 3$ is statement in basic language, $u \,||\, u$ is a behavior. For this case, $U = \not\subset, P = \{u\}, B = \{u = (x < 5) \to (x := x + 1)\}$. The equation $u = (x < 5) \to (x := x + 1)$ considered here as a basic behavior and it used for definition of agent behavior $u \,||\, u$.

In insertion modeling environment considered as agent with insertion function. So, **Insertion function** is defined by the following identities and rules of operational semantics.

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 $\beta = 1$.

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 (it generates 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 the environment*. Formally, the states of the form $E[0]$ are equivalent to 0, and states of the form $E[\Delta]$ are equivalent to Δ (if $E[\Delta] = E[\Delta] + \Delta$ 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 a successful termination. If there is at least one degenerated component in parallel

composition, then the corresponding state is a dead lock state. 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 a *full unfolding algorithm for a parallel composition*. This is a complete unfolding and the main result of this chapter shows that it is not needed to make the complete unfolding at each step of verification. Let $u = a_1.u_1 ||\ldots||a_n.u_n$,

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

then 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[14]. 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 is 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 left hand side by the right 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, $a_1, a_2, \ldots$

are operators or conditions such that if a_i is a condition, then u_i is a termination constant.

Normal form of environment state is a term of the 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 a trace equivalent to some normal form.*

5 Verification

A property ξ of environment state is said 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 in terms of insertion modeling

Let S_1, S_2 be state of the model M. The problem of reachability checking is the answer to the question if a path exists from the state S_1 to the state S_2 on model M, or not. Usually models are highly non-deterministic. This non-determinism is based on interleaving of parallel processes: $a||b=(a;b)+(b;a)$ (here a,b are some processes, "||" is parallel composition, ";" is sequential composition and "+" is non-deterministic composition). From other side this non-determinism could produce additional paths from S_1 to S_2 and additional states. So, let call interleaving reduction problem an answer to the question how to reduce non-determinism of the model M to find the path from S_1 to S_2 as quickly as possible.

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 properties to be checked contains the property of a state "to be a dead lock" and a property "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 following formula of unfolding: $\sum_{i=1}^{n} E[unfold(u,i)]$. Such algorithm was described in [14]. It builds all states space for reachability checking which isn't possible always. The properties to be checked are checked in the process of unfolding and the states that satisfy checked properties are collected. More economic unfolding algorithm can be constructed using the following *partial unfolding algorithm.*

6 Partial Unfolding

Two operators a and a' are called permutable regarding the state of E if $E[a*a']=E[a'*a]$ and dynamically permutatable regarding the state E (denoted by

$a \xleftrightarrow{E} a'$) if $E[a*a'] = E[a'*a] \neq 0$. Let $E[u] = E[a_1.u_1 || \ldots || a_n.u_n]$ is a state of the environment. Let's select the component $s = a_i.u_i$ and build $nonp(E, a_i) = \{a_j \mid i \neq j \wedge \neg(a_i \xrightarrow{E} a_j)\}$. We obtain:

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

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

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

$$C(E,i) = \sum_{k \neq i \wedge (a_k, a_i) \notin nonp(E,a_i) \wedge a_k \xleftrightarrow{E} a_w} a_k.(\ldots || a_{k-1}.u_{k-1} || ((p; a_w); u'_k) || a_{k+1}.u_{k+1} || \ldots)$$

In the last formula $((p; a_w); u'_k) = u_k$ and *p* are sequences of compositions of actions (behavior). Function *punfold* is called *partial unfolding of parallel composition*. Let's consider the following algorithm of reachability checking: we need to check the properties on a current state of the environment and each state that is reachable from this in one step. Partial unfolding is used for main function of unfolding states. This algorithm is called partial unfolding algorithm of reachability checking.

In general, the *punfold* uses the notion of dynamic permutability of operators, but it is not optimal, because it uses 4 times application of function predicate transformer *pt* for each pair of operators. Using *punfold* can be optimized by using the concept of static permutability of operators. Algorithm which uses *punfold* with some optimization is considered in section 6.3.

6.1. Optimization of partial unfolding of states.

Theorem 1. If two operators $p = \alpha \to a, q = \beta \to b$ are permutable regarding the states $E_1 = \alpha \wedge \beta$, $E_2 = \neg\alpha \wedge \beta$, $E_3 = \alpha \wedge \neg\beta$ then they are permutable regarding any state [13].

The sufficient condition of permutability of two operators $p = \alpha \to a, q = \beta \to b$ is valid under the following conditions:

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

Example 1. Let *a,b:int* and $1[init.(a_1.good || b_0.bad + b_1.good)]$ is initial state and behavior, where *init*, a_1, b_0, b_1 - operators. Agent's behavior could be represented by the following list of equations: $init = ((a = b) \to 1).AndFork$,

$AndFork = a_1 || (b_0 + b_1)$, $a_1 = ((a = 1) \to 1)$, $b_0 = ((b = 0) \to 1)$, $b_1 = ((b = 1) \to 1)$.

Sufficient condition of permutability for operators a_1, b_0, b_1 is performed in this case, but there can be a case in the simulation where the state of the environment includes some formula, which combines predicate memory of various parallel processes *(a=b)*. So, one of the operator will not be applicable, ie a pair of operators

will be dynamically permutable regarding this state. Thus, the notion of sufficient conditions of permutability of operators need to be strengthened.

To improve the usage of permutability for this example, we need $sat(E \wedge \alpha \wedge \beta) = 1$, otherwise operators will be dynamically permutable regarding state *E*. Let's try to obtain a sufficient condition for dynamic permutability of two operators regarding some condition *E*.

The notion of dynamic permutability of two operators *p, q* regarding some state *E* uses a condition:

$$E[p*q] = E[q*p] \neq 0$$

So, let $E' = pt(\alpha \wedge pt(\alpha \wedge \beta, b), a) = pt(\beta \wedge pt(\alpha \wedge \beta, a), b) \neq 0$ and try to apply backward predicate transformer to the state *E*. We obtain:

$$pt^{-1}(pt^{-1}(E', \beta, b), \alpha, a) = E''_{(q,p)},\ pt^{-1}(pt(E', \alpha, a), \beta, b) = E''_{(p,q)}.$$

Theorem 2. If $E' = pt(\alpha \wedge pt(\alpha \wedge \beta, b), a) = pt(\beta \wedge pt(\alpha \wedge \beta, a), b) \neq 0$ then $E''_{(q,p)} \wedge E''_{(p,q)} \neq 0$.

Proof.

Let's assume the contrary that $E''_{(q,p)} \wedge E''_{(p,q)} = 0$. Since the backward predicate transformer turns back to its possible state transition set, it means that $(\alpha \wedge \beta \to E''_{(q,p)}) \wedge (\alpha \wedge \beta \to E''_{(p,q)})$. State $E''_{(p,q)}$ ($E''_{(q,p)}$) specifies a set of concrete states from which transitions from state $\alpha \wedge \beta$ with operators *p* and *q* (*q* and *p*) exist, which means that $\alpha \wedge \beta \wedge E''_{(p,q)} \wedge \alpha \wedge \beta \wedge E''_{(q,p)} \Rightarrow \alpha \wedge \beta \wedge E''_{(p,q)} \wedge E''_{(q,p)} \neq 0$. So, we got a contradiction, because if $E''_{(q,p)} \wedge E''_{(p,q)} = 0$ then $E' = 0$. The theorem is proved.

This condition means that if two operators were dynamically permutable regarding *E* then it is necessary that current state of the environment should satisfy theorem 2.

Let *E* be some state of environment.

Theorem 3. If two operators $p = \alpha \to a, q = \beta \to b$ satisfy the sufficient condition of permutability and $E \wedge E''_{(q,p)} \wedge E''_{(p,q)} \neq 0$ then $E[p*q] = E[q*p] \neq 0$.

Proof.

Let's consider the condition of dynamic permutability regarding *E*: $E[p*q] = E[q*p] \neq 0$.

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

Next, let's consider in details the sufficient condition permutability of operators that satisfies the operators *p, q*:

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

$$\Rightarrow pt(\alpha \wedge pt(\neg\alpha \wedge \beta \wedge E, b), a) \vee pt(\alpha \wedge pt(\neg\alpha \wedge \beta \wedge \neg E, b), a) = 0 \Rightarrow$$

$$\Rightarrow pt(\beta \wedge pt(\alpha \wedge \neg\beta \wedge E, a), b) = 0 \wedge pt(\beta \wedge pt(\alpha \wedge \neg\beta \wedge \neg E, a), b) = 0$$

Equality $E[p*q] = E[q*p]$ shall be satisfied because otherwise the operators *p*, *q* do not satisfy the sufficient condition permutability of operators (Theorem 1). Thus, we have:

$$pt(\beta \wedge pt(E \wedge \alpha \wedge \beta), a), b) \vee pt(\beta \wedge pt(E \wedge \alpha \wedge \neg\beta), a), b) =$$

$$= pt(\alpha \wedge pt(E \wedge \beta \wedge \alpha, b), a) \vee pt(\alpha \wedge pt(E \wedge \beta \wedge \neg\alpha, b), a) \Rightarrow$$

$$\Rightarrow pt(\beta \wedge pt(E \wedge \alpha \wedge \beta), a), b) = pt(\alpha \wedge pt(E \wedge \beta \wedge \alpha, b), a)$$

Let's consider opposite:

$$pt(\beta \wedge pt(E \wedge \alpha \wedge \beta), a), b) = pt(\alpha \wedge pt(E \wedge \beta \wedge \alpha, b), a) = 0$$

Let's continue to consider sufficient conditions of permutability:

$$pt(\alpha \wedge pt(\alpha \wedge \beta, b), a) = pt(\beta \wedge pt(\alpha \wedge \beta, a), b) \neq 0 \Rightarrow$$

$$\Rightarrow pt(\alpha \wedge pt(\alpha \wedge \beta \wedge (E \vee \neg E), b), a) = pt(\beta \wedge pt(\alpha \wedge \beta \wedge (E \vee \neg E), a), b) \neq 0 \Rightarrow$$

$$\Rightarrow pt(\alpha \wedge pt(\alpha \wedge \beta \wedge E \vee \alpha \wedge \beta \wedge \neg E), b), a) =$$

$$= pt(\beta \wedge pt(\alpha \wedge \beta \wedge E \vee \alpha \wedge \beta \wedge \neg E), a), b) \neq 0 \Rightarrow$$

$$\Rightarrow pt(\alpha \wedge pt(\alpha \wedge \beta \wedge E), b), a) \vee pt(\alpha \wedge pt(\alpha \wedge \beta \wedge \neg E), b), a) =$$

$$= pt(\beta \wedge pt(\alpha \wedge \beta \wedge E), a), b) \vee pt(\beta \wedge pt(\alpha \wedge \beta \wedge \neg E), a), b) \neq 0 \Rightarrow$$

$$\Rightarrow pt(\alpha \wedge pt(\alpha \wedge \beta \wedge \neg E), b), a) = pt(\beta \wedge pt(\alpha \wedge \beta \wedge \neg E), a), b) \neq 0$$

This means that the condition $\alpha \wedge \beta \wedge \neg E \wedge E''_{(q,p)} \wedge E''_{(p,q)} \neq 0$ should be satisfied. But we have the following condition $E \wedge E''_{(q,p)} \wedge E''_{(p,q)} \neq 0$. Thus, both conditions must be satisfied, however:

$$\alpha \wedge \beta \wedge \neg E \wedge E''_{(q,p)} \wedge E''_{(p,q)} \wedge E \wedge E''_{(q,p)} \wedge E''_{(p,q)} = 0$$

So we got a contradiction. The theorem is proved.

If there are two operators $p = \alpha \to a, q = \beta \to b$ that satisfy the sufficient condition of permutability. Condition $E \wedge E''_{(q,p)} \wedge E''_{(p,q)} \neq 0$ is called sufficient condition of dynamic permutability of operators *p, q* regarding the environment *E*.

From a practical point of view, let's try to identify requirements for operators with which we can determine statistically whether they satisfy the sufficient condition of dynamic permutability or not.

Let *E* be a state of the environment, and *p* - an operator. The set *A(E)* is called the *set of all attributes from state E* and *A(p) is* called the *set of all attributes in the statement p*[15].

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

$$A(p) \cap A(q) = \varnothing \wedge pt(\alpha, a) \neq 0 \wedge pt(\beta, b) \neq 0$$

Theorem 4. If two operators $p = \alpha \to a, q = \beta \to b$ are statically permutable then they are dynamically permutable.

Proof.

To prove the theorem we need to show that these operators satisfy necessary condition of permutability of operators in this case.

Since $A(p) \cap A(q) = \varnothing \wedge pt(\alpha, a) \neq 0 \wedge pt(\beta, b) \neq 0$ and *theorem 1* then

$pt(\beta \wedge pt(\alpha \wedge \neg\beta, a), b) = pt(\beta \wedge \neg\beta \wedge pt(\alpha, a), b) = 0$

$pt(\alpha \wedge pt(\neg\alpha \wedge \beta, b), a) = pt(\alpha \wedge \neg\alpha \wedge pt(\beta, b)) = 0$

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

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

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

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

The theorem is proved.

This theorem means that if a predicate that combines memory in a state of environment with different operators is absent then checking the necessary condition of dynamic permutability is not required. Since in this case a usage of one of these operators does not affect the applicability of another operator. The appearance and disappearance of such predicates can be defined statically and syntactically.

Thus, in *Example 1* operators are statically permutable, but after applying *init* operator formula will contain predicate that combines memory of operators a_1, b_0 and a_1, b_1. So, we have to use sufficient condition for dynamic permutability of pairs of operators, a_1, b_0 and a_1, b_1 regarding the state of the environment after application of *init* operator . So, $E = (a = b)$. Let's statically compute sufficient condition of permutability of operators:

$(a_1, b_o) : E''_{(a_1, b_0)} \wedge E''_{(b_0, a_1)} = (a = 1) \wedge (b = 0)$

$(a_1, b_1) : E''_{(a_1, b_1)} \wedge E''_{(b_1, a_1)} = (a = 1) \wedge (b = 1)$

Next let's try to apply sufficient condition of dynamic permutability of operators regarding the condition *E* for both pairs of operators:

$(a_1, b_o) : E \wedge E''_{(a_1, b_0)} \wedge E''_{(b_0, a_1)} = (a = b) \wedge (a = 1) \wedge (b = 0) = 0$

$(a_1, b_1) : E \wedge E''_{(a_1, b_1)} \wedge E''_{(b_1, a_1)} = (a = b) \wedge (a = 1) \wedge (b = 1) \neq 0$

Thus, operators (a_1, b_o) will be dynamically permutable regarding the condition *E*, and operators (a_1, b_1) will be dynamically permutable. This means that interleaving will be removed in correct way for this problem.

6.2. The Problem of Reachability of Some State

The approach proposed in the previous sections can be applied to the problem of finding deadlocks in a given model, but if the user specifies a state of environment you want to check coverage, whereas previously proposed approach should be strengthened.

Example 2. Let *a,b:int* and 1[*init.*$(a_1 || b_1)$] be initial behavior and a state of the environment, where *init*, a_1, b_1 - operators. Agent's behavior could be represented by the following list of equations:

$$init = ((a = 0) \wedge (b = 0) \to 1), AndFork, AndFork = a_1 || b_1,$$
$$a_1 = (1 \to (a := 1)), b_1 = (1 \to (b := 1))$$

Let's check recheability of the state $F = (a = 0) \wedge (b = 1)$.

After applying the operator *init* obtains the state of the environment $E = (a = 0) \wedge (b = 0)$. Operators a_1, b_1 are statically permutable and can be applied to the state *E*, which means that they are dynamically permutable regarding *E*. So, $E[a_1 || b_1] \Rightarrow E[a_1.b_1]$, which means that the operator b_1 never will be applied before the operator a_1 and user defined state *F* will be unreachable after interleaving reduction. Let's try to enhance sufficient condition of operators permutability regarding some state *E* with some conditions related to formula *F*. $F \wedge (E[a_1 * b_1]) = F \wedge (E[b_1 * a_1]) = 0$ for this example then we consider conditions for operators separately (not for pairs of operators).

Let $p = \alpha \to a$ be an operator.

Theorem 5. If $\alpha \wedge pt^{-1}(pt(\alpha, a), \alpha, a) \neq 0$ then $pt(\alpha, a) \neq 0$.

Proof.

Let's consider the opposite $\alpha \wedge pt^{-1}(pt(\alpha, a), \alpha, a) \neq 0$ and $pt(\alpha, a) = 0$. In this case by performed substitution it can be easily obtained the following:

$$\alpha \wedge pt^{-1}(pt(\alpha, a), \alpha, a) \neq 0 \Rightarrow \alpha \wedge pt^{-1}(0, \alpha, a) \neq 0 \Rightarrow 0 \neq 0$$

So we got a contradiction. The theorem is proved.

The operator $p = \alpha \to a$ is called permutable regarding some user defined state *F*, if the following conditions are satisfied:

1) $\alpha \wedge pt^{-1}(pt(\alpha, a), \alpha, a) \neq 0$;

2) $F \wedge \alpha \wedge pt^{-1}(pt(\alpha, a), \alpha, a) = F \wedge pt(\alpha, b)$.

This permutability means that an operator does not change the state of the environment in order to reach the user defined state changed. From reachability point of view we are interested in two cases (if $\alpha \wedge pt^{-1}(pt(\alpha, a), \alpha, a) \neq 0$):

1) $F \wedge \alpha \wedge pt^{-1}(pt(\alpha, a), \alpha, a) \neq 0 \wedge F \wedge pt(\alpha, b) = 0$;

2) $F \wedge \alpha \wedge pt^{-1}(pt(\alpha, a), \alpha, a) = 0 \wedge F \wedge pt(\alpha, b) \neq 0$.

In first case, the reachability of user defined state should be checked immediately before application of an operator, and in the second case - after.

If operators satisfy the sufficient condition of dynamic permutability, but at least one of them is not permutable regarding a user defined state then this operator should be applied first.

This approach can be applied to any algorithm of unfolding.

So, for checking of reachability of the user defined state *F* the notion of permutability regarding the user defined state could be used. You can't consider a pair of operators if both of them do not satisfy this condition.

Coming back to example 2. Operator a_1 will not be permutable regarding the user defined state *F*:

$1 \wedge pt^{-1}(pt(1, a := 1), 1, a := 1) \Rightarrow 1 = E_1$

$pt(1, a := 1) \Rightarrow (a = 1) = E_2$

$F \wedge E_1 = F \wedge E_2 \Rightarrow (a = 0) \wedge (b = 1) \wedge 1 = (a = 0) \wedge (b = 1) \wedge (a = 1) \Rightarrow 0$

Operator b_1 is permutable regarding *F*:

$1 \wedge pt^{-1}(pt(1, b := 1), 1, b := 1) \Rightarrow 1 = E_1$

$pt(1, b := 1) \Rightarrow (b = 1) = E_2$

$F \wedge E_1 = F \wedge E_2 \Rightarrow (a = 0) \wedge (b = 1) \wedge 1 = (a = 0) \wedge (b = 1) \wedge (b = 1) \Rightarrow 1$

From other side operators a_1, b_1 are statically permutable since $E = (a = 0) \wedge (b = 0)$ has no predicates that combine memory of these operators.

This means that in this case you should first apply an operator b_1, then you need to check reachability of F (since $F \wedge E_1 \Rightarrow (a = 0) \wedge (b = 1) \wedge 1 \neq 0$) before applying a_1. And after that you can try to apply a_1. So,

$pt((a = 0) \wedge (b = 0), b := 1) = (a = 0) \wedge (b = 1)$

Then let's check the reachability of user defined state:

$(a = 0) \wedge (b = 1) \wedge (a = 0) \wedge (b = 1) \Rightarrow 1$

So, reachability of user defined state is proved.

6.3. The Main Interleaving Reduction Algorithm

Let $E[u]$ be a model (an initial state of the environment and behavior), where u is behavior, and F is some user defined state which reachability should be checked. So, we need to check reachability of F in the model $E[u]$ and all its deadlocks.

The main interleaving reduction algorithm for reachability checking is represented in fig. 1.

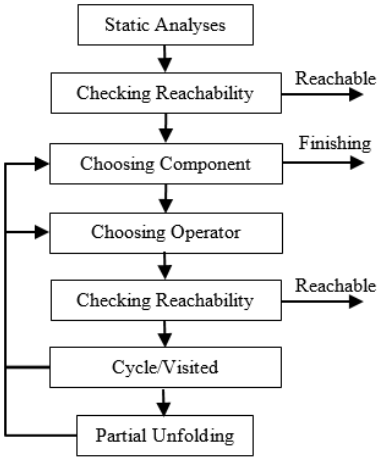

Fig. 1. The main interleaving reduction algorithm for reachability checking

Static Analyses. In the initial behavior u we look for set Op_n (set of operators of a n-th parallel process) on each parallel process. Next, for each pair of operators from different parallel processes we build a table: $H : N \times Op_N \rightarrow N \times Op_N \times G \times Bool$. This table by a pair (parallel process identifier and operator) returns four (number of parallel process that is not equal to the previous one, and the operator which is

permutable to current one, the last two parameters are sufficient condition for permutability of operators and flag for static permutability of operators).

For each pair of operators that does not satisfy the sufficient condition of permutability of operators the table is filled: $D: N \times Op_N \to \{N, Op_N\}$, where $\{N, Op_N\}$ is a set of pairs: the number of parallel process and an operator, which does not satisfy the sufficient condition of permutability.

Each operator is constructed $Flt: Op_N \to Bool \times Bool \times Bool$, which defines the triple for each operator in set: value of reachability of user defined state before and after application of an operator, the third value is 1 if all operators from other processes are statically permutable with this one, and 0 if not.

Checking Reachability. Checking reachability of user defined state *F*. If the filter is reachable then saving corresponded trace and stop modeling.

Choosing Component. We build normal form (*section 4*). From list of components we should choose one to continue working. Here we propose to select first component from the list, but in general case here some heuristics could be applied (it's out of scope of this paper). If there is no component left then finishing.

Choosing Operator. In a chosen component we select operators in the following order. First we check the applicability of operators for which the third option from the table *Fpl* is 1. If there are no operators or they can't be applied, then we choose other operators. If the flag state of the reachability of user defined state is 1 then we first try to apply such operators that are permutable regarding a user defined condition for both operators in the table *Flt* being 1. If the flag state of user defined state equals 0 then we choose to consider operators whose value pairs in the table *Flt* are (0,1). If one of such operators is applicable then after his application we need to check the reachability of user defined state. If user defined state is reachable then finishing. If no such operators left then deadlock is obtained and we get new component. Otherwise, finishing.

Cycle/Visited. Checking cycle/visited filters. If the filter is reachable then we choose a next operator to work. If no operator left then we choose other component.

Partial Unfolding. We try to build *B* and *C* if it is required, using notion of sufficient condition of static and dynamic permutability. If the last operators satisfy the sufficient condition of static (dynamic) permutability then *B=0*. Starting delayed to build *C*. In general, this problem is formulated as follows. It is given: current state of environment *E*, operator *a* one of the parallel processes u_1 (other processes are delayed to use some operators in this process, including the process by which it was taken the operator *a*), and delayed parallel processes. In the set of parallel processes it is needed to find the operator *b*, which does not satisfy the sufficient condition of permutability (*table D*) or does not satisfy the condition of sufficient dynamic permutability regarding the *E*. In order to check whether these operators are in a given process, you generally build all states space. If these operators are not found then all resulting state of the search should be removed from storage cycle/visited filters. This is necessary because subtrace which leads to the required operator can modify the current state of the environment and a sufficient condition for dynamic permutability can not be performed, although for state *E* a sufficient condition for dynamic permutability is performed. But in some cases the search for these operators do not need to spend a dynamically performance of all subtrace. If the state of the

environment does not contain predicates that combine memory of these operators and the set of attributes operators intersect, then we can use the concept of specialization[16] in order to break into several operators and sufficient condition will check only those suboperator memory that belongs to the operator. If such operator was found then we add result of it insertion into the list of components.

For trace equivalence each trace for deadlock should be checked additionally because of used normal form. Each cycle/visited trace should be checked for reachability of user defined state in the following way: turning back with a help of backward predicate transformer until operator doesn't have value 1 as first and second parameters in the table *Flt*.

Theorem 6. $punfold(E,u,i)$ Function which was represented in fig. 1 saves property of reachability checking.

Proof.

Let's suppose opposite that the function $punfold(E,u,i)$ does not save the property of reachability checking. This means that for some state of environment E such operator a exists, which is applied to the $E(E \xrightarrow{a} E')$ and doesn't exist as first action in components $A(i), B(E,i), C(E,i)$. So, the operator will be dynamically permutable regarding the environment E and all other operators resulting behavior components $A(i), B(E,i), C(E,i)$ (according to *Choosing Operator, Partial Unfolding*). In addition, operator a can be applied after the application of the first operators in the resulting behavior of components $A(i), B(E,i), C(E,i)$.

This means that the required state of the environment is reachable, but after applying the operator a on the next step these operators are dynamically permutable regarding the environment E. From other point of view we do not take into account E'. If value of pair for the operator in the table *Flt* is (0,1) then according to step 5 we have to take it into consideration and in this case it will be the first operator in the behavior of components $A(i), B(E,i), C(E,i)$. If the value of such pair is (1,1) then the state of the environment is reachable in the next step, as defined *Flt*. If the value is (1,0) then before application of the operator a we need to check the reachability of the environment E and definitions in *Flt.* If the value is (0,0) then required state is not reached in E'. That means that the required state of the environment is not unreached at all states of the environment as a result of unfolding application $punfold(E,u,i)$. So we got contradiction. The theorem is proved.

The main problem of proposed algorithm is complicity to find component $C(E,i)$. One of the ways to speed up such algorithm is delayed computation. The idea of this method contains the following:

1) To collect all such special states from *Partial Unfolding*, where we should find component $C(E,i)$ and finding the required states with a help of different methods: all states coverage, invariants etc.

2) To continue algorithm with built states of component $C(E,i)$.

Such algorithm is called *incremental algorithm of reachability checking*.

6.4. The Static Interleaving Reduction Algorithm

If operators and initial environment state of model do not contain predicates which connected to the memory of different parallel processes then general algorithm in previous section could be simplified. Such algorithm is called *static interleaving reduction algorithm*.

For the component $C(E,i)$ of $punfold(E,u,i)$ we should check reachability of application of operator which is not dynamically permutable for a_i (see section 5). For elimination of such reachability checking we could build additional interleaving according to checking of reachability of corresponded operator in behavior. For example, let $a||b.c||d$ and $\neg(a \leftrightarrow c) \wedge \neg(b \leftrightarrow d)$, E be some environment state. So,

$$punfold(E, a||b.c||d, 1) = E[a.(b.c)||d] + E[b.c.(a||d)] + E[d.(a||b.c)]$$

Here we take into account $b.c.(a||d)$, because $\neg(a \leftrightarrow c)$; $d.(a||b.c)$, because we have taken $b.c.(a||d)$ and $\neg(b \leftrightarrow d)$.

6.5. Examples of Application

In Table 1 information about few big examples run with our static interleaving reduction algorithm are presented. All of them give out of memory error (PC with 8 Gb of RAM) if we try to obtain all states space. So, we try to run them on implementation of proposed algorithm in Insertion Modeling System.

Table 1. Experiments result for static interleaving reduction algorithm

No.	Total number of operators pairs	Number of non-permutable operator's pairs	Time
1	660	30	25 min (on prototype)
2	780	14	47 min 4 sec
3	12882	225	1 min 42 sec

Here "on prototype" means that this experiment was done on the algorithm which was implemented in language of Insertion Modeling System[17]. Other experiments were run on the algorithm which was implemented on C++. "Total number of operators pairs" is number of pairs of operators which were detected in parallel behavior of the model. "Number of non-permutable operator's pairs" is a number of detected non-permutable pairs of operators. Example 2 works slower because it has four parallel processes and each sequential process more non-deterministic, example 1 has only 2 parallel processes.

7 Conclusion

Described algorithm of interleaving reduction was implemented in Insertional Modeling System. Its restriction for usage of static permutability condition was good account in set of big examples. In any case, the main interleaving reduction algorithm depends on reachability checking problem (component $C(E,i)$, section 6).

Notoriously this problem is algorithmically unsolvable. It means that you could always prepare example where interleaving reduction will be impossible (for example, all operators will be non-permutable etc).

References

1. The Interleaving Paradigm, http://www-i2.informatik.rwth-aachen.de/i2/fileadmin/user_upload/documents/MC08/mc_lec3.pdf
2. Symbolic Modeling, http://en.wikipedia.org/wiki/Model_checking
3. Alessio Lomuscio, Wojciech Penczek, and Hongyang Qu. 2010. Partial Order Reductions for Model Checking Temporal-epistemic Logics over Interleaved Multi-agent Systems. Fundam. Inf. 101, 71-90, 1-2 (January 2010).
4. C. Norris Ip and David L. Dill. 1996. Better Verification through Symmetry. Form. Methods Syst. Des. 9, 41-75, 1-2 (August 1996).
5. Edmund M. Clarke, Orna Grumberg, and David E. Long. Model Checking and Abstraction. *ACM Trans. Program. Lang. Syst.* 16,1512-1542, 5 (September 1994)
6. Vijay D'Silva, Mitra Purandare, and Daniel Kroening. Approximation Refinement for Interpolation-Based Model Checking. In Proceedings of the 9th international conference on Verification, model checking, and abstract interpretation (VMCAI'08), Francesco Logozzo, Doron A. Peled, and Lenore D. Zuck (Eds.), Berlin, Heidelberg, pp. 68-82, Springer-Verlag (2008)
7. Data-Flow Analysis, http://en.wikipedia.org/wiki/Data-flow_analysis.
8. K.L. McMillan: 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)
9. E. W. Dijkstra. Hierarchical Ordering of Sequential Processes, Acta Informatica 1(2), 115-138. (1971)
10. A. Letichevsky, A. Godlevsky, A. Letichevsky Jr., S. Potienko, V. Peschanenko. Properties of Predicate Transformer of VRS System. Cybernetics and System Analyses 4, 13-16. (2010)
11. Escobar, J. Meseguer: Symbolic Model Checking of Infinite-State Systems Using Narrowing. Proceedings of the 18th International Conference on Term Rewriting and Applications, LNCS 4533, 153-168, Springer (2007).
12. Frédéric Herbreteau, Grrégoire Sutre, and The Quang Tran. 2007. Unfolding Concurrent Well-Structured Transition Systems. In Proceedings of the 13th international conference on Tools and algorithms for the construction and analysis of systems (TACAS'07), Orna Grumberg and Michael Huth (Eds.), Berlin, Heidelberg, 706-720, Springer-Verlag (2007).
13. A. Letichevsky, O. Letychevskyi, V. Peschanenko. About One Efficient Algorithm for Reachability Checking in Modeling and Its Implementation. ICTERI 2012, Communications in Computer and Information Science 149, 149-165. (Springer, 2012)
14. A. Letichevsky, O. Letychevskyi, V. Peschanenko. Insertion Modeling System. PSI 2011, Lecture Notes in Computer Science 7162, 262-274. (Springer, 2011)
15. C. Norris Ip and David L. Dill. 1996. Better Verification through Symmetry. Form. Methods Syst. Des. 9, 41-75, 1-2 (August 1996)
16. V. Peschanenko, A. Guba, C. Shushpanov. Specializations in Symbolic Verification. Communications in Computer and Information Science 412, 332–354, Springer (2013)
17. APS and IMS systems, http://apsystems.org.ua