

PCF-based formalization of the parallel composition of automata

Artem Davydov, Aleksandr Larionov and Nadezhda Nagul

Matrosov Institute for System Dynamics and Control Theory of the Siberian Branch of the Russian Academy of Sciences, 134 Lermontov str., 664033 Irkutsk, Russia

E-mail: artem@icc.ru, bootfrost@zoho.com, sapling@icc.ru

Abstract. The paper demonstrates how the automatic theorem proving technique of the PCF calculus is applied to construct parallel composition of automata. Parallel composition plays an essential role in the supervisory control theory at different stages of systems and supervisors design. Improved formalization of discrete event systems as positively-constructed formulas along with auxiliary predicates, serving for accessibility of the automaton checking, simplify parallel composition construction.

1. Introduction

This paper continues the series of works developing a new way to deal with discrete-event systems (DES). Being subject to Ramadge-Wonham supervisory control theory (SCT) for DES, a physical system and requirements to system's behavior are usually modeled by one or several finite state automata, sometimes called free behavior models and specifications. Nodes of the state transition diagrams of the automata involved correspond to essential states of the system while edges correspond to events leading to changes of the states. Then, in the SCT framework, a supervisor as a means of control is built to guarantee the specifications fulfillment. In previous papers the application of automated theorem proving based on the positively-constructed formulas (PCF) to the formalization of DES [1] was presented. The PCF calculus advantages allow solving various problems of supervisor constructing sequence, for example, problems of checking observability and co-observability of languages [2, 3].

The PCF calculus is developed in [4, 5] to formalize and solve some control theory problems. In [5] the proof of soundness and completeness of the PCF calculus as the first-order logical formalism is presented, and its further development may be found in [6]. The PCF calculus is naturally aimed at solving problems of dynamic systems control due to its features, such as modifiability of semantics (constructive, nonmonotonic, temporal, etc.) and an ability to construct intuitionistic inferences of some non-Horn formulas. Constructive semantics allows one to extract some knowledge (for example, action plans) from proofs, while non-monotonicity and a treat of time help to construct plans in dynamically changing subject areas. Interactive properties of the PCF calculus, its application for dynamic systems control and action planning are described in [5] with the examples of elevator group control, mobile robot action planning and

telescope guidance. Problems of automated theorem proving software in the PCF calculus design and implementation are briefly discussed in [7]. Due to its features, the PCF calculus allows one to combine the automatic search for logic inferences with special domain based heuristics which are customizable for each task being solved.

Since more and more complicated systems are modeled as DES to apply SCT results and obtain control required, such DES are usually constructed as sets of automata where each automaton corresponds to some aspect of physical system's functioning. For example, in [8] it is shown how state of the art problems of swarm robotics can be formalized and solved using the SCT framework. Problems of segregation, aggregation, object clustering and group formation are considered for two robotic platforms, the e-puck and the Kilobot. To formalize, for example, the case study of segregation, four free behaviour models are used which represent input device to configure the robot, the robot's ability to assume one of the leader types or to be a follower, motion capabilities, and the robot's ability to receive messages from nearby leader robots. Specification for the segregation case study consists of three automata: one of them configures the robot through user interaction; one allows followers to move and leaders to transmit a signal, and one moves the follower robot according to the signal received. To build a supervisor, free behavior models and specifications are combined using an operation known as *parallel, or synchronous, composition*.

In [9] to investigate the applicability of SCT to a high-tech system, a theme park vehicle, called a multimover, is chosen. Its main structure contains mechanical parts to which sensors and actuators are mounted. Supervisory control is used to coordinate the individual components and give the desired functionality to the whole system. The user interface of the multimover contains three buttons (the reset, the forward and the backward button) so three automata are used to represent the buttons. Sensors of a multimover are modelled by six automata, with five of them having the same structure. Two automata correspond to the steer motor and to the drive motor. One automaton is employed for ride control and one for the scene program handler. In the event-based framework, it is not possible to derive a centralized supervisor for a problem of this size because parallel composition leads to state explosion phenomenon. However, combining by parallel composition components effected by the same specification, aggregative distributed synthesis [10] becomes possible.

The examples above show that parallel composition as one of composition operations on automata plays a very important role in SCT. After a supervisor has been designed, behavior of the system under its control is described by the parallel composition of the automaton representing uncontrolled behavior and the automaton of the supervisor. Parallel composition also serves for checking controllability of a specification language and helps to design supervisor if specification occurs to be controllable. Target language [8], built as parallel composition of the specification and the system automata, is a base for founding bad states which should be forbidden by a supervisor.

The current paper demonstrates how the PCF calculus is applied to construct parallel composition using logical inference only. The rest of the paper is organized as follows. In the next section the basic notion of the automata-based DES with a small example of the parallel composition of automata is provided. The third section contains brief description of PCFs and the PCF calculus. In section 4 main features of the PCF calculus are discussed. The PCF-formalization of parallel composition is given in section 5. Application of the results obtained are discussed in Conclusions.

2. Automata representation of discrete event systems

The first [11] and the most popular way to represent DES is a finite-state automaton. It is defined as a structure $\mathcal{G} = (Q, \Sigma, \delta, q_0, Q_m)$ where Q is the set of states q ; Σ the set of events; $\delta: \Sigma \times Q \rightarrow Q$ the transition function; $q_0 \in Q$ the initial state; $Q_m \subset Q$ the set of marker

states. Marked states correspond to some completed tasks performed by the system under consideration, for example, finished processing sequence in manufacturing system. Note that marking does not have any terminal meaning that is system proceeds after a marker state is achieved. That is why \mathcal{G} is called a *generator* of a formal language [11]. As usual, let Σ^* denote the set of all strings over Σ , including the empty string ε . δ is easily extended on strings from Σ^* . A language generated by \mathcal{G} is $L(\mathcal{G}) = \{w : w \in \Sigma^* \text{ and } \delta(w, q_0)!\}$, while a language marked by \mathcal{G} is $L_m(\mathcal{G}) = \{w : w \in L(\mathcal{G}) \text{ and } \delta(w, q_0) \in Q_m\}$. Here $\delta(s, q_0)!$ means that $\delta(s, q_0)$ is defined.

As well known, two operations are used to compose automata: product and parallel, or synchronous, compositions. In SCT parallel composition is used in supervisor constructing algorithms, for checking controllability, to construct system behavior under control of the supervisor. Moreover, since complex systems are usually designed using modular architecture, with each module having its special functionality, generator \mathcal{G} may correspond to one of these modules. Parallel composition then serves to compose system model from models of separate modules. Leaving SCT applications for future work, in what follows general notion of parallel composition of deterministic automata, as used to built complex system, is considered.

Definition 1 [Parallel composition] If two generators $\mathcal{G}_i = (Q^i, \Sigma_i, \delta_i, q_0^i, Q_m^i)$, $i = 1, 2$, are given then *parallel composition* is defined [12] as $\mathcal{G}_1 || \mathcal{G}_2 := Ac(Q^1 \times Q^2, \Sigma_1 \cup \Sigma_2, \delta, (q_0^1, q_0^2), Q_m^1 \times Q_m^2)$,

$$\delta(\sigma, q^1, q^2) = \begin{cases} (\delta_1(\sigma, q^1), \delta_2(\sigma, q^2)) & \text{if } \delta_1(\sigma, q^1)!, \delta_2(\sigma, q^2)!, \text{ and } \sigma \in \Sigma_1 \cap \Sigma_2, \\ (\delta_1(\sigma, q^1), q^2) & \text{if } \delta_1(\sigma, q^1)! \text{ and } \sigma \in \Sigma_1 \setminus \Sigma_2, \\ (q^1, \delta_2(\sigma, q^2)) & \text{if } \delta_2(\sigma, q^2)! \text{ and } \sigma \in \Sigma_2 \setminus \Sigma_1, \\ \text{undefined} & \text{otherwise.} \end{cases} \quad (1)$$

Here $Ac(\mathcal{G})$ denotes operation of taking accessible part of the automaton that is deleting all states that are not reachable from q_0 by some string from $L(\mathcal{G})$ and transitions attached to these states.

In the parallel composition of two automata events shared by the automata must occur concurrently. Thus the automata are synchronized by the common events. Events that are not shared by the automata may occur independently. Using the same principle, above definition may be extended on any number of automata. Parallel composition is commutative up to a reordering of the state components in composed states and associative: $(\mathcal{G}_1 || \mathcal{G}_2) || \mathcal{G}_3 = \mathcal{G}_1 || (\mathcal{G}_2 || \mathcal{G}_3)$. The parallel composition of a set of n automata can therefore be defined using associativity: $\mathcal{G}_1 || \mathcal{G}_2 || \mathcal{G}_3 := (\mathcal{G}_1 || \mathcal{G}_2) || \mathcal{G}_3$.

Example 1. Consider two state transition diagrams in figures 1 and 2. Initial states are marked with arrows while double circles denote marked states. The automaton in figure 3 is the result of the parallel composition of the automata in figures 1 and 2.

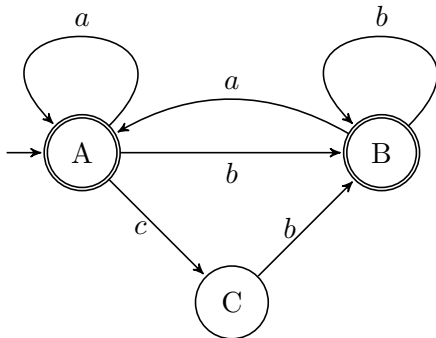


Figure 1. Generator \mathcal{G}_1 .

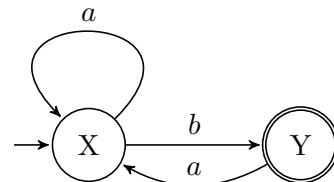


Figure 2. Generator \mathcal{G}_2 .

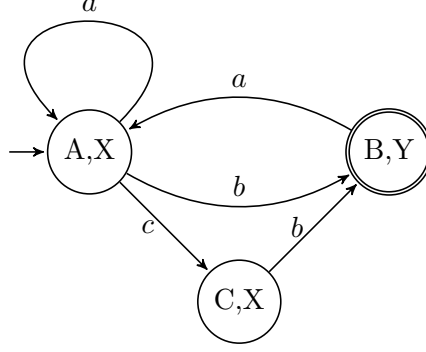


Figure 3. The parallel composition of \mathcal{G}_1 and \mathcal{G}_2 .

Before operation Ac implementation, the resulting automaton has 6 states. Three of them are inaccessible. Attention should be paid to the missing self-loop over b at the state (B, Y) . The fourth rule of (1) is applied in this case since the event b is shared by automata but $\delta_2(b, Y)$ is undefined. In section 5 this example will be formalized by PCFs. Note that \mathcal{G}_2 may be considered as a specification for \mathcal{G}_1 .

3. PCF calculus

Consider a language of a first-order logic that consists of first-order formulas (FOFs) built out of atomic formulas with $\&$, \vee , \neg , \rightarrow , \leftrightarrow operators, \forall and \exists quantifier symbols, and constants *true* and *false*. The concepts of a term, an atom, and a literal are defined in the usual way. Non-atomic formulas and subformulas will be denoted further by capital calligraphic letters (\mathcal{F} , \mathcal{P} , \mathcal{Q} , etc.), possibly with indices. Sets of formulas will be denoted by capital Greek letters (Φ , Ψ , etc.), possibly with indices.

Let $X = \{x_1, \dots, x_k\}$ be a set of variables, $A = \{A_1, \dots, A_m\}$ be a set of atomic formulas called *conjunct*, and $\Phi = \{\mathcal{F}_1, \dots, \mathcal{F}_n\}$ be a set of FOFs. The formulas $\forall x_1 \dots \forall x_k (A_1 \& \dots \& A_m) \rightarrow (\mathcal{F}_1 \vee \dots \vee \mathcal{F}_n)$ and $\exists x_1 \dots \exists x_k (A_1 \& \dots \& A_m) \& (\mathcal{F}_1 \& \dots \& \mathcal{F}_n)$ are denoted as $\forall x_1, \dots, x_k A_1, \dots, A_m \{\mathcal{F}_1, \dots, \mathcal{F}_n\}$ and $\exists x_1, \dots, x_k A_1, \dots, A_m \{\mathcal{F}_1, \dots, \mathcal{F}_n\}$, respectively. They can be abbreviated as $\forall_X A \Phi$ and $\exists_X A \Phi$, respectively, keeping in mind that the \forall -quantifier corresponds to $\rightarrow \Phi^\vee$, where Φ^\vee means disjunction of all the formulas from Φ , and \exists -quantifier corresponds to $\& \Phi^\&$, where $\Phi^\&$ means conjunction of all the formulas from Φ . Any of sets X , A , Φ may be empty and in this case they could be omitted in formula formalization. Thus, if $Q \in \{\forall, \exists\}$ then $Q_X A \emptyset \equiv Q_X A$, $Q_X \emptyset \Phi \equiv Q_X \Phi$, and $Q_\emptyset A \Phi \equiv Q A \Phi$. Since an empty disjunction is identical to *false* whereas an empty conjunction is identical to *true*, the following equivalences are correct: $\forall_X A \emptyset \equiv \forall_X A \rightarrow \text{false} \equiv \forall_X A$, $\exists_X A \emptyset \equiv \exists_X A \& \text{true} \equiv \exists_X A$, $\forall \emptyset \Phi \equiv \text{true} \rightarrow \Phi \equiv \forall \Phi$, and $\exists \emptyset \Phi \equiv \text{true} \& \Phi \equiv \exists \Phi$.

The constructions $\forall_X A$ and $\exists_X A$ are called *positive type quantifiers* (TQ) because A is a conjunction of positive atoms only and referred to as *type condition* for X . In practice, these constructions denote phrases such as “for all X satisfying A there is...”, “there exists X satisfying property A such that...”, and so on; for example, “for all integer x, y, z and $n > 2$ there is $x^n + y^n \neq z^n$ ”. Originally, the term “type quantifier” was introduced by N. Bourbaki [13] as a part of notation for formalization of mathematics.

3.1. PCF language explicit definition

Definition 2 [Positively constructed formula (PCF)] Let X be a set of variables and A a conjunct. Then

- (i) $\exists_X A$ and $\forall_X A$ are \exists -PCF and \forall -PCF, respectively.
- (ii) If $F = \{F_1, \dots, F_n\}$ is a set of \forall -PCFs then $\exists_X A: F$ is \exists -PCF.
- (iii) If $F = \{F_1, \dots, F_n\}$ is a set of \exists -PCFs then $\forall_X A: F$ is \forall -PCF.
- (iv) Any \exists -PCF or \forall -PCF is PCF.

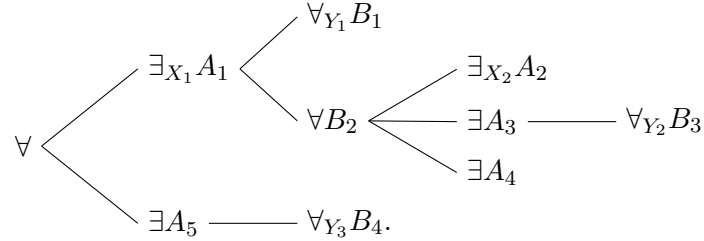
This form of logical formulas is referred to as positively constructed formulas (PCFs) as they are written with positive type quantifiers only and do not contain the explicit logic negation sign. Without loss of generality only closed formulas will be considered. Any FOF may be represented as PCF [5].

PCF starting with $\forall\emptyset$ is called *PCF in the canonical form*. Any PCF can be represented in the canonical form. If F is a non-canonical \exists -PCF then $\forall\emptyset: F$ is the canonical PCF since $\forall\emptyset: F \equiv true \rightarrow F \equiv F$. If F is a non-canonical \forall -PCF then the canonical PCF is $\forall\emptyset: \{\exists\emptyset: F\} \equiv true \rightarrow true \& F \equiv F$. Type quantifiers $\forall\emptyset$ and $\exists\emptyset$ are called *fititious* since they do not influence truth value of the original PCF and do not bind any variables. They are used to regularize PCFs, i.e. transform them to the canonical ones.

For the sake of readability we represent PCFs as trees whose nodes are type quantifiers, and we use corresponding names: *node*, *root*, *leaf*, *branch*. For example, PCF

$$\forall \{ \exists_{X_1} A_1 \{ \forall_{Y_1} B_1, \forall B_2 \{ \exists_{X_2} A_2, \exists A_3 \{ \forall_{Y_2} B_3 \}, \exists A_4 \} \}, \exists A_5 \{ \forall_{Y_3} B_4 \} \}$$

may be represented as a tree



Parts of a canonical PCF are named as follows:

- (i) The root of a canonical PCF's tree-view $\forall\emptyset$ is called a PCF *root*.
- (ii) Each PCF root child $\exists_X A$ is called a PCF *base*, the conjunct A is called *base of facts*, and PCF rooted from the base is called *a base subformula*.
- (iii) PCF base children $\forall_Y B$ are called *questions* to the parent base. If a question is a leaf of a tree then it is called *a goal question*.
- (iv) Subtrees of questions are called *consequents*.

3.2. Inference rule

In the process of reasoning one often proves a statement \mathcal{F} by refuting its negation. We intend to proceed similarly.

Definition 3 [Answer] A question $\forall_Y D: \Upsilon$ to a base $\exists_X A$ has an *answer* θ if and only if θ is a substitution $Y \rightarrow H^\infty \cup X$ and $D\theta \subseteq A$, where H^∞ is Herbrand universe based on constant and function symbols that occur in corresponding base subformula.

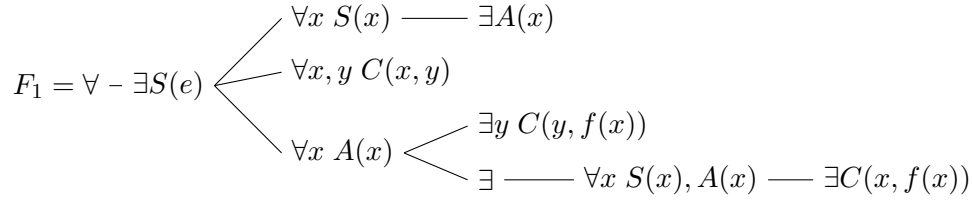
Definition 4 [Splitting] Let $B = \exists_X A: \Psi$, and $Q = \forall_Y D: \Upsilon$, where $\Upsilon = \{\exists_{Z_1} C_1: \Gamma_1, \dots, \exists_{Z_n} C_n: \Gamma_n\}$ then $split(B, Q) = \{\exists_{X \cup Z_1'} AUC_1': \Psi \cup \Gamma_1', \dots, \exists_{X \cup Z_n'} AUC_n': \Psi \cup \Gamma_n'\}$, where $'$ is a variable renaming operator. We say that B is split by Q . Obviously, $split(B, \forall_Y D) = split(B, \forall_Y D: \emptyset) = \emptyset$.

Definition 5 [Inference rule ω] Consider some canonical PCF $F = \forall\emptyset: \Phi$. Let there exist a question Q that has an answer θ to appropriate base $B \in \Phi$. Then $\omega F = \forall\emptyset: \Phi \setminus \{B\} \cup \text{split}(B, Q\theta)$.

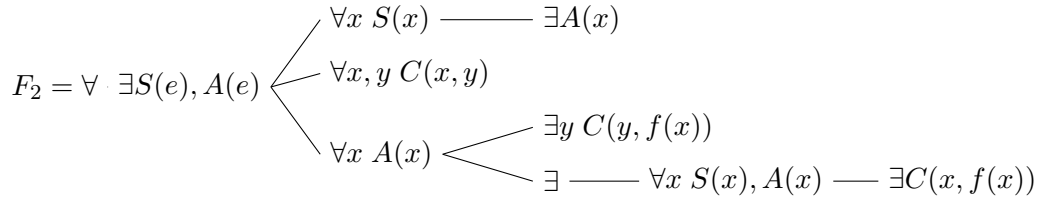
In other words, if a question has an answer to its base then the base subformula is split by this question. In the case of a goal question we say that the basic subformula is refuted because $\text{split}(B, \forall y D) = \emptyset$. The refuted base subformula B is removed from the set of base subformulas Φ since $\Phi \setminus \{S\} \cup \emptyset = \Phi \setminus \{S\}$. Note that when the set Φ becomes empty after applying ω rule and PCF becomes just \forall then it can be concluded that the negation of the original formula is unsatisfiable.

Any finite sequence of PCFs $\mathcal{F}, \omega\mathcal{F}, \omega^2\mathcal{F}, \dots, \omega^n\mathcal{F}$, where $\omega^s\mathcal{F} = \omega(\omega^{s-1}\mathcal{F})$, $\omega^1 = \omega$, $\omega^n\mathcal{F} = \forall$, is called a *deduction* of \mathcal{F} in the PCF calculus (with the axiom \forall). Suppose that a search strategy verifies the questions in consecutive order, without omissions (i.e. the verification is repeated only when the whole list of questions was used), and it does not use several applications of ω to a question with the same θ (question-answering method of automated deduction).

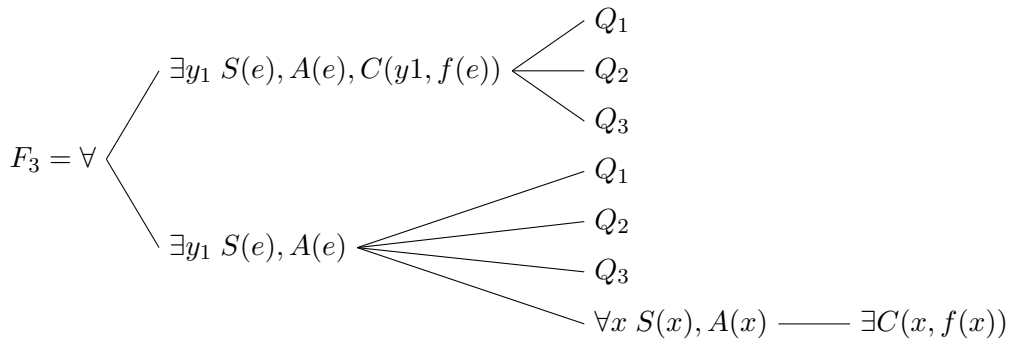
Example 2. [Refutation of PCF] Consider PCF



At the first step of the inference there is the only answer $\{x \rightarrow e\}$ to the first question (name it Q_1 , from the top to the bottom numbering is used). After applying the rule ω with this answer to the only base of F_1 , the formula is converted to the form F_2 :



At the second step there is also only one answer $\{x \rightarrow e\}$ to the question Q_3 . After applying ω with this answer, F_2 is split because Q_3 has the disjunctive branching. The formula gets the form F_3 :



At the third step the first base can be refuted by answering to Q_2 (the goal question) with $\{x \rightarrow y_1; y \rightarrow f(e)\}$. The refuted base (and its whole base subformula) is to be deleted from the list of the base subformulas.

At the fourth step there is the answer $\{x \rightarrow e, y \rightarrow e\}$ to the fourth new question of the second base which adds the atom $C(e, f(e))$ to it.

At the fifth step the only base can be refuted by answering to Q_2 (the goal question) with the answer $\{x \rightarrow e; y \rightarrow f(e)\}$. This finishes refutation because all the bases were refuted.

4. PCF features

Consider main features of the PCF language. First of all, unlike predicate calculus language syntax, PCFs have rather unconventional and exquisite form:

- (i) Any formula written in the PCF language is characterized by a large-block structure and contains only positive quantifiers.
- (ii) Any PCF has a simple and regular structure, i.e. the formula to some extent is characterized by predictability of its structure determined by the order of \exists - and \forall -nodes which alternate at each branch.
- (iii) The negation of PCF is merely obtained by the replacement of the symbol \exists with \forall , and vice versa (after what canonization follows).
- (iv) The PCF-representation is more compact than the representation in terms of the well known *language of clauses* [14] used, for example, in the resolution method [15]. Furthermore, it is more compact than representations in terms of standard disjunctive and conjunctive normal forms.
- (v) It is not necessary to preprocess (scolemize) the formulas by eliminating of all existential quantifiers. The scolemization procedure related to this elimination leads to elevating the complexity of terms (see [14]).
- (vi) The natural structure of the knowledge in the language of PCFs is *retained* better. Indeed, the description of the knowledge in the original form does not employ "theoretical" quantifiers $\forall x, \exists x$. Instead of them, type quantifiers $\forall x(A \rightarrow \sqcup), \exists x(A \& \sqcup)$ with simple and natural forms of conditions A (e.g., without symbols \forall, \exists inside of A) are employed. The PCF structure retains most of the original structure of the knowledge when the knowledge is written as an expression constructed of atoms (and/or their negations) with the help of positive quantifiers and logic connectives $\&, \vee$. In this case, the PCF structure may differ from the initial structure merely by appending auxiliary quantifiers $\exists \emptyset$ (instead of some connectives $\&, \vee$ mentioned above) and/or replacing the negations of atoms $\neg A$ with the structures $\forall A \emptyset$.

Consider features (iv)–(vi) in greater detail with the help of the following example.

Example 3. The FOF

$$\forall x(A^{\&} \rightarrow (\exists y_1 B_1^{\&} \vee \dots \vee \exists y_k B_k^{\&})), \quad (2)$$

where $B_i^{\&} = C_1^i \& \dots \& C_n^i$, $A^{\&} = A_1 \& \dots \& A_l$, $i = \overline{1, k}$, in terms of the PCF-representation has $l + n \cdot k$ atoms. In terms of the language of clauses it takes the form

$$\&_{(i_1, \dots, i_k) \in (\overline{1, n})^k} (\neg A_1 \vee \dots \vee \neg A_l \vee C_{i_1}^1 \vee \dots \vee C_{i_k}^k), \quad (3)$$

i.e. contains $(l + k)n^k$ atoms (n^k clauses). It is also obvious that, except of the auxiliary quantifiers, the number of atoms in the PCF-representation is not larger than in any classical disjunctive (conjunctive) normal form. Moreover, the representation (3) of the initial formula (2) is not only more complicated but also substantially destroys the structure of (2), although in the language of PCFs the initial structure of (2) is saved:

$$\forall_X A \{ \exists_{Y_1} B_1, \dots, \exists_{Y_k} B_k \}.$$

The language of clauses has been used in the resolution method [16] since its representation (2) in comparison with formulas of the classical predicate calculus is homogeneous. This allowed J. Robinson, basing on Herbrand's results, to develop a popular method of automated deduction with one inference rule known as the resolution rule.

Below we list quite important advantages of PCF calculus which are due not only to the features of the PCF language, but also to the proper deductive power of the calculus.

- (i) Unlike that in many other known logic calculi, e.g. those of Hentzen type, the PCF calculus has only one inference rule ω . Availability of the only rule narrows the combinatorial space. However, this is not the main advantage of the calculus. The set of important merits of the PCF calculus includes not only uniqueness of its inference rule but also its unary character (e.g., in comparison with the resolution method). Another advantage of the PCF calculus, obvious from the view-point of decrease of combinatorics, is that the inference rule ω is a large-block one, likewise the language of the PCF calculus itself. Such a rule leads to additional reduction of complexity of the combinatorial space (unlike that of the resolution rule which is also unique but is binary and a small-block one).
- (ii) The deduction (refutation) technique described has centered on application of ω to the questions only, i.e. to the successors of PCF roots. Application of ω to questions only is based on the properties (i), (ii) of the PCF language and allows one to focus the attention of the technique on the local fragments of PCF, without any loss of completeness of the technique.
- (iii) The deduction technique can be described in pithy terms of the question-answering procedure instead of technical terms of formal deducibility (i.e. in terms of logic connectives, atoms, etc.).
- (iv) Owing to properties (i), (ii), (vi) of the PCF language and (ii), (iii) of the PCF calculus, the deduction technique is quite compatible with heuristics of specific applications as well as with general heuristics of control of deduction. Owing to (i), the deduction process consists of large-block steps, so is well observable and controllable.
- (v) The deduction technique offers natural OR-parallelism, because the refutations of basic subformulas are executed independently of one another.
- (vi) The deductions obtained are quite interpretable by man owing to properties (iii), (iv). This interpretability is quite important from the viewpoint of man-machine applications. So, conceptually, the language and the calculus of PCFs are not only machine-oriented, but also human-oriented: the implementation of these techniques for the purpose of specific applications may use these two capabilities to some greater or lesser extent.
- (vii) Due to the features of the PCF language and calculus, the PCF formalism has another very important merit: its semantics can be modified without any changes of the axiom \forall and the inference rule ω . Such modifications are implemented merely by some restrictions of applying ω and allow one to transform classical semantics of the PCF calculus in nonmonotonous semantics, constructive (intuitionistic) semantics, etc.

All these features are useful for formalizing the discrete event systems since the special structure and convenient way of inference search make it possible to describe the automata underlying the DES and to customize the inference strategies.

5. PCF formalization of parallel composition

In order to formalize a generator representing DES, the following predicates will be used. Let $L(s, S)$ denote “ s is a current sequence of events in a state S ” and $L_m(s, S)$ denote “ s is a current sequence of events in a state S , and s is a string of a marked language”. Thus, first arguments of these atoms accumulate the strings of corresponding languages, generated and marked by the

automaton. Atoms of the form $\delta(S_1, e, S_2)$ will be interpreted as automaton transitions from a state S_1 to a state S_2 with an event e . If the right state of a transition is marked, then delta atoms with an index is used, i.e. $\delta_m(S_1, e, S_2)$ if S_2 is a marked state. The function symbol “.” denotes strings concatenation, and the “ ε ” symbol corresponds to the empty string.

5.1. Formalization of a generator

Behavior of a generator, representing some DES, may be simulated using the logical inference of a PCF which has the general structure depicted in figure 4.

$$\exists L(\varepsilon, S_0), \delta(S_1, e, S_2), L_m(\varepsilon, S_0), \delta_m(S_1, e, S_2) \left\langle \begin{array}{l} \forall \sigma, s, e, s' L(\sigma, S), \delta(s, e, s') - \exists L(\sigma \cdot e, s') \\ \forall \sigma, s, e, s' L(\sigma, S), \delta_m(s, e, s') - \exists L_m(\sigma \cdot e, s') \end{array} \right.$$

Figure 4. General view of PCF formalizing DES.

This way of formalization of a generator is more compact than the one previously suggested [1] and gives more possibilities when setting up search strategies of the inference. Indeed, there is the only base subformula describing the automaton. Transitions of the automaton are listed in the base of facts and two questions correspond to the implementation of transitions, with one of them corresponding to marked transitions. If an answer to a question is found, $L(\dots, \dots)$ or $L_m(\dots, \dots)$ atoms are added to the base, with a new event as the first argument, thus accumulating strings of the generated and marked languages.

Example 4. Consider the automaton \mathcal{G}_1 from example 1 in section 2. The base of the PCF corresponding to transitions of \mathcal{G}_1 looks as

$$\begin{array}{l} L(\varepsilon, A), \delta(A, a, A), \delta(A, b, B), \delta(A, c, C), \delta(B, a, A), \delta(B, b, B), \delta(C, b, B), \\ L_m(\varepsilon, A), \delta_m(A, a, A), \delta_m(A, b, B), \delta_m(B, a, A), \delta_m(B, b, B), \delta_m(C, b, B). \end{array} \quad (\mathcal{G}_1^{PCF})$$

Recall that the base of the PCF is a set of atoms, but we arrange them, aligned and grouped in meaning, for clarity. Questions that generate the strings of the languages $L(\mathcal{G}_1)$, $L_m(\mathcal{G}_1)$ are the same as in the PCF on figure 4.

An inference constructing is demonstrated on the example of PCF for the automaton \mathcal{G}_2 , which has the following base:

$$\begin{array}{l} L(\varepsilon, X), \delta(X, b, Y), \delta(X, a, X), \delta(Y, a, X), \\ \delta_m(X, b, Y). \end{array} \quad (\mathcal{G}_2^{PCF})$$

Table 1 describes the inference of PCF with the base \mathcal{G}_2^{PCF} . The first column contains the number of a question, an answer to which is used. The second column contains a δ -atom used for the answer search. In the third column a corresponding substitution (answer) is presented. The fourth column contains atoms derived and added to the base, in this case, $L(\dots, \dots)$ and $L_m(\dots, \dots)$. The first arguments of these atoms are the strings of the languages $L(\mathcal{G}_2)$ and $L_m(\mathcal{G}_2)$. The strategy used to build the inference is to answer both questions with the current contents of the base, if possible.

Table 1: Inference of PCF for \mathcal{G}_2 .

Q	Base atoms	Substitution	Atoms added
1	$\delta(X, b, Y)$	$\{\sigma \rightarrow \varepsilon, s \rightarrow X, e \rightarrow b, s' \rightarrow Y\}$	$L(b, Y)$

Continued on the next page

Table 1 – continued from the previous page

Q	Base atoms	Substitution	Atoms added
2	$\delta_m(X, b, Y)$	$\{\sigma \rightarrow \varepsilon, s \rightarrow X, e \rightarrow b, s' \rightarrow Y\}$	$L_m(b, Y)$
1	$\delta(X, a, X)$	$\{\sigma \rightarrow \varepsilon, s \rightarrow X, e \rightarrow a, s' \rightarrow X\}$	$L(a, X)$
1	$\delta(Y, a, X)$	$\{\sigma \rightarrow b, s \rightarrow Y, e \rightarrow a, s' \rightarrow X\}$	$L(ba, X)$
2	$\delta_m(X, b, y)$	$\{\sigma \rightarrow a, s \rightarrow X, e \rightarrow b, s' \rightarrow Y\}$	$L_m(ab, Y)$
1	$\delta(X, b, y)$	$\{\sigma \rightarrow ba, s \rightarrow X, e \rightarrow b, s' \rightarrow Y\}$	$L(bab, Y)$
2	$\delta_m(X, b, y)$	$\{\sigma \rightarrow ba, s \rightarrow X, e \rightarrow b, s' \rightarrow Y\}$	$L_m(bab, Y)$
1	$\delta(X, a, X)$	$\{\sigma \rightarrow ba, s \rightarrow X, e \rightarrow b, s' \rightarrow X\}$	$L(baa, X)$
and so on...			

5.2. Formalization of parallel composition

The goal is to build parallel composition of automata without going beyond the framework of a formal logic, i.e. it is necessary to construct a composition of automata using a logic inference, having their PCF-formalized representations. Consider two PCFs, F_1 and F_2 , similar to the one in figure 4, representing some generators \mathcal{G}_1 and \mathcal{G}_2 . To separate different automata in the combined base, atoms are indexed with the numbers of PCFs. To obtain PCF representing $\mathcal{G}_1 \parallel \mathcal{G}_2$, we combine bases and questions of F_1 and F_2 , and add new questions that represent the rules of (1) in definition 1 (section 2). PCF in figure 5 is a combined formula whose inference builds the languages of automata \mathcal{G}_1 , \mathcal{G}_2 , and also adds new atoms in the base that correspond to transitions of $\mathcal{G}_1 \parallel \mathcal{G}_2$ including marked transitions.

Let us describe the strategy that allows one to stop the inference in the most reasonable way. If the questions are numbered from the top to the bottom, then the first four of them are the questions generating strings of the languages $L(\mathcal{G}_i)$, $L_m(\mathcal{G}_i)$, $i = 1, 2$, and the strategy for answering these questions remains the same. Question 5 is used only once: it adds to the base the initial state of the composition automaton. Question 6 adds a starting atom for further construction of transitions and is also used once. A pair of questions 7 and 8 corresponds to the first rule of (1), which handles transitions over events common to both automata. Questions 9 and 10 correspond to the second rule of (1), and are aimed to handle events that belong to the alphabet of the automaton \mathcal{G}_1 only. Questions 11 and 12 handle events which belong exactly to the alphabet of the second automaton. To determine sets $\Sigma_1 \cap \Sigma_2$, $\Sigma_1 \setminus \Sigma_2$, and $\Sigma_2 \setminus \Sigma_1$ computable predicates are used. Their computation in this case is an elementary action since the alphabets of both automata are known, however, if desired, these actions can be carried out within the logical inference.

Obviously, the answers to questions 7-12 form a finite set and are exhausted as the composition automaton is constructed. According to the strategy of inference search, after answering questions 5 and 6 the inference uses questions 7-12 only until all of their answers have been exhausted, then they may be omitted from using. It should be noted that the formula on figure 5 can be used to obtain languages for the constructed composition, if we add a pair of questions similar to the first four, which generate languages of the automata \mathcal{G}_1 and \mathcal{G}_2 .

Example 5. An example of constructing parallel composition of automata with the help of a constructive inference in the PCF calculus is given for the automata \mathcal{G}_1 and \mathcal{G}_2 from example 1 (section 2). Bases \mathcal{G}_1^{PCF} and \mathcal{G}_2^{PCF} are indexed as mentioned above and the inference is constructed using the questions depicted in figure 5. The inference with the corresponding composition automaton drawing is depicted in table 2.

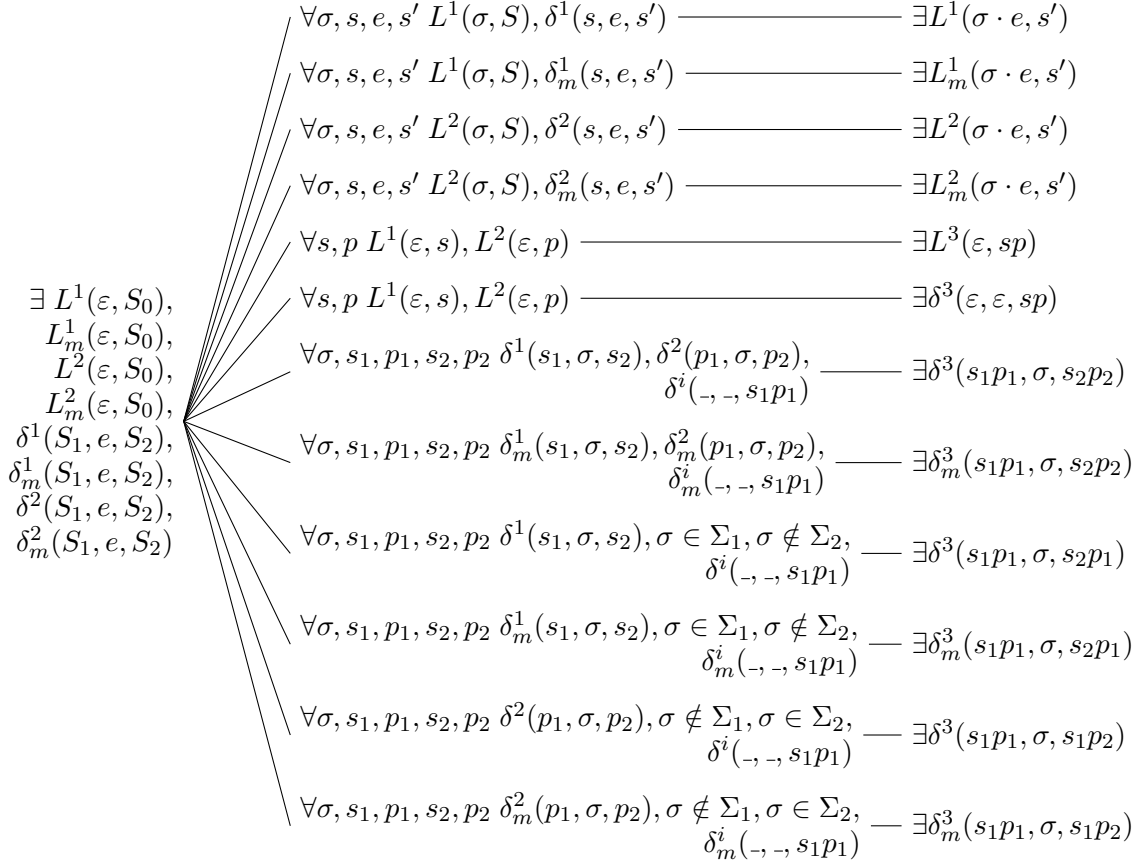


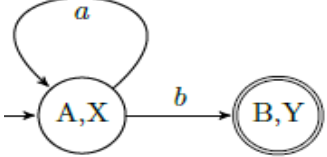
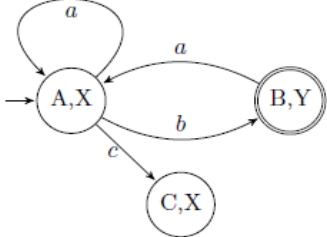
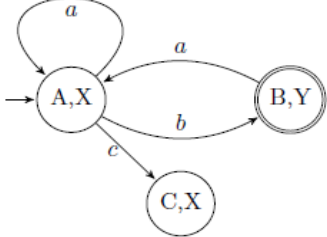
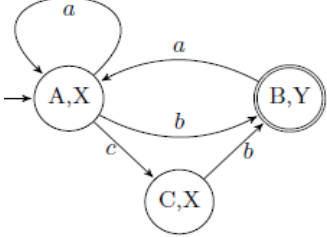
Figure 5. General view of PCF constructing parallel composition of two automata.

Table 2: Inference of PCF constructing $\mathcal{G}_1 || \mathcal{G}_2$.

Q	Base atoms	Substitution	Atoms added	Result
5	$L^1(\varepsilon, A), L^2(\varepsilon, X)$	$\{s \rightarrow A, p \rightarrow X\}$	$L(\varepsilon, AX)$	\rightarrow
6	$L^1(\varepsilon, A), L^2(\varepsilon, X)$	$\{s \rightarrow A, p \rightarrow X\}$	$\delta^3(\varepsilon, \varepsilon, AX)$	\rightarrow
7	$\delta^3(\varepsilon, \varepsilon, AX),$ $\delta^1(A, a, A),$ $\delta^2(X, a, X)$	$\{s_1 \rightarrow A, p_1 \rightarrow$ $X, \sigma \rightarrow b, s_2 \rightarrow$ $A, p_2 \rightarrow X\}$	$\delta^3(AX, a, AX)$	\rightarrow
7	$\delta^1(A, b, B),$ $\delta^2(X, b, Y)$	$\{s_1 \rightarrow A, p_1 \rightarrow$ $X, \sigma \rightarrow b, s_2 \rightarrow$ $B, p_2 \rightarrow Y\}$	$\delta^3(AX, b, BY)$	\rightarrow

Continued on the next page

Table 2 – continued from the previous page

Q	Base atoms	Substitution	Atoms added	Result
8	$\delta_m^1(A, b, B),$ $\delta_m^2(X, b, Y)$	$\{s_1 \rightarrow A, p_1 \rightarrow$ $X, \sigma \rightarrow b, s_2 \rightarrow$ $B, p_2 \rightarrow Y\}$	$\delta_m^3(AX, b, BY)$	
9	$\delta^3(\varepsilon, \varepsilon, AX),$ $\delta^1(A, c, C)$	$\{s_1 \rightarrow A, p_1 \rightarrow X,$ $\sigma \rightarrow c, s_2 \rightarrow C\}$	$\delta^3(AX, c, CX)$	
7	$\delta^3(AX, b, BY),$ $\delta^1(B, a, A),$ $\delta^2(Y, a, X)$	$\{s_1 \rightarrow B, p_1 \rightarrow$ $Y, \sigma \rightarrow a, s_2 \rightarrow$ $A, p_2 \rightarrow X\}$	$\delta^3(BY, a, AX)$	
7	$\delta^3(AX, c, CX),$ $\delta^1(C, b, B),$ $\delta^2(X, b, Y)$	$\{s_1 \rightarrow C, p_1 \rightarrow$ $X, \sigma \rightarrow b, s_2 \rightarrow$ $B, p_2 \rightarrow Y\}$	$\delta^3(CX, b, BY)$	

Thus, a method has been developed for constructing the parallel composition of automata using only constructive logical inference of PCF. The advantage of this method is that the composition automaton after the completion of the inference does not have inaccessible states, so there is no need to use Ac operation. This is due to control of the connectivity of the graph, representing the automaton, at each step of constructing the inference. It is implemented in PCF with the help of $\delta^i(-, -, x)$ atoms in questions 7–12 where index i is a parameter that takes any of the values $\{1, 2, 3\}$ and the symbols “ $_-$ ” correspond to variables, i.e. placements for the substitution of any terms.

6. Conclusions

The parallel composition of automata, widely employed in SCT, was considered in this paper as a part of the automatic theorem proving based approach to deal with DES in the SCT framework. New formalization of DES as PCF along with auxiliary predicates, serving for accessibility of the automaton checking, simplified parallel composition construction. In future work predicates for controllable and observable events will be added to above PCFs for checking controllability, observability, etc., and designing supervisors. Although some results in this direction were obtained earlier, complex systems, built using the parallel composition, may be considered now due to the new results presented above.

Acknowledgments

The research is supported by the Russian Science Foundation (project no. 16-11-00053). This work was also supported in part by the Presidium RAS, program no. 2, project "Methods and tools for solving hard-search problems with supercomputers" (reg. no. AAAA-A18-118031590005-5).

References

- [1] Davydov A, Larionov A and Nagul N 2017 *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (IEEE)
- [2] Davydov A, Larionov A and Nagul N 2018 *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* pp 0938–0943
- [3] Davydov A, Larionov A and Nagul N 2018 *AIP Conference Proceedings* **2046** 020021 (Preprint <https://aip.scitation.org/doi/pdf/10.1063/1.5081541>) URL <https://aip.scitation.org/doi/abs/10.1063/1.5081541>
- [4] Vassilyev S N 1990 *The Journal of Logic Programming* **9** 235–266
- [5] Zherlov A K, Vassilyev S N, Fedosov E A and Fedunov B E 2000 *Intelligent control of dynamic systems* (Moscow: Fizmatlit) in Russian
- [6] Davydov A, Larionov A and Cherkashin E 2011 *Automatic Control and Computer Sciences* **45** 402–407
- [7] Larionov A, Davydov A and Cherkashin E 2013 *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija* **2013** 1023–1028
- [8] Lopes Y K, Trenkwalder S M, Leal A B, Dodd T J and Groß R 2016 *Swarm Intelligence* **10** 65–97 ISSN 1935-3820 URL <http://dx.doi.org/10.1007/s11721-016-0119-0>
- [9] Forschelen S T J, van de Mortel-Fronczak J M, Su R and Rooda J E 2012 *Discrete Event Dynamic Systems* **22** 511–540 ISSN 1573-7594 URL <http://dx.doi.org/10.1007/s10626-012-0130-6>
- [10] Su R, van Schuppen J H and Rooda J E 2010 *IEEE Transactions on Automatic Control* **55** 1627–1640 ISSN 0018-9286
- [11] Ramadge P J and Wonham W M 1987 *SIAM Journal on Control and Optimization* **25** 206–230
- [12] Cassandras C G and Lafortune S 2008 *Introduction to Discrete Event Systems* (Springer US)
- [13] Bourbaki N 2004 *Theory of sets* (Springer Berlin Heidelberg)
- [14] Davis M and Putnam H 1960 *J. ACM* **7** 201–215 ISSN 0004-5411 URL <http://doi.acm.org/10.1145/321033.321034>
- [15] Robinson J A 1965 *J. ACM* **12** 23–41 ISSN 0004-5411 URL <http://doi.acm.org/10.1145/321250.321253>
- [16] Russell S and Norvig P 2010 *Artificial Intelligence: A Modern Approach* 3rd ed Series in Artificial Intelligence (Upper Saddle River, NJ: Prentice Hall) URL <http://aima.cs.berkeley.edu/>