Modular discrete event systems control based on logic inference

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 illustrates an application of the recently developed method of dealing with controlled automata-based discrete event systems with the help of logical inference. The method based on the calculus of positively constructed formulas is extended on the case of systems built out of sets of modules. Specifications restricting system behavior are also supposed to be modular. Due to the special features of the calculus of positively constructed formulas, it may be applied at the upper level of a robot group control system. The case study of mobile robots pushing a block to a target area is considered.

1. Introduction

There are various ways to organize control over a complex technical object. It may be centralized, decentralized or distributed control. In this paper, one class of systems is in the focus, namely, the class of Discrete Event Systems (DES), and control implemented by a special object called supervisor is considered. Supervisory Control Theory (SCT), developed as a tool of restricting DES behavior according to a set of constraints imposed by some specification, deals with DES presented in the form of finite state automata as generators of formal languages. A detailed description of state-of-the-art SCT is presented in, e.g. [1, 2, 3].

DES for complex systems usually consists, or may be split, into parts, combined using the parallel composition of automata, and there are a set of specifications describing constraints on system functioning, which are relative only to a part of system modules. Modular structures of both DES and specification on DES behavior may be accounted for supervisor constructing and implementing. The main advantage of using modular supervision compared to the monolithic one is a possible saving of memory for storing the supervisor and, in some cases, less computational resources spent for its design.

The modular approach to DES control was studied since 1988 when in [4] the case of specification represented as the conjunction of elementary ones was considered. A local supervisor is constructed to ensure each elementary specification, and global control is then realized by the intersection of sets of events enabled by local supervisors. In [5] a step further was made, and an approach to the modular design of supervisors in the case of modular generator is developed. Important results on controllability of modular specification and nonblockingness were established. The work [6] extended these results on the case of not only two but any number of specifications. Results of [5] are exploited, for example, in [7] and [8] to solve problems of robot group and robot swarm control. The PCF-based approach to the supervisory control of modular DES is presented in this paper.

To intellectualize the solving problems of SCT we suggested the method of dealing with DES based on the Automated Theorem Proving (ATP) in the calculus of Positively Constructed Formulas (PCFs). The PCF calculus is a complete method for ATP [9, 10], admitting functional symbols [11], [12]. Its application at the upper level of a robot group control system for constructing plans of robot actions is discussed in [13]. The computer programs called provers are usually exploited for ATP. Usage of provers is associated with well-known difficulties: a) the proving program makes too many inference steps, most of which are redundant or irrelevant; b) the program has to store too much information in the database; c) inference rules and inference steps are not of the same size; d) inadequate focus, i.e. the program quickly stumbles on a false search path. Unlike many logical calculi that underlie the theoretical basis of modern provers, such as Vampire [14], nanoCoP [15], E [16], the features of the calculus of PCFs help to eliminate or significantly reduce the above difficulties. A detailed discussion of the characteristics and capabilities of the PCF calculus can be found in [12].

Applying the PCF calculus to SCT for DES, such basic problems as controllability checking [17], supremal controllable sublanguage of a given specification language construction [18] and a monolithic supervisor realization were considered and solved. The developed approach is used to create multi-level hierarchical control systems for mobile robots and robot groups [19]. A small overview of the current state of research on ATP in robotics is presented in [20], while a detailed review on planning in robotics, including the use of ATP, is presented in [21]. The work [22] lies at the intersection of ATP and machine learning, presenting a reinforcement learning toolkit for experiments on guiding ATP in the calculus of connections. The core of the toolkit is a Prolog-based prover. In [23] for planning and control in swarm robotics, the PDDL language is used, which is based on the classical STRIPS-style ATP.

The paper structure is the following. The preliminaries on supervisory control for DES are given in section 2. Section 3 presents the PCF language and the PCF calculus. In section 4, the PCF-based approach to supervisory control is described. In section 5, a case study is given for robot group control using PCFs. The task of moving an object to a target area by a pair of robots is considered. A modular DES and modular specifications are described and modular supervision realizing the specifications are implemented with the help of a PCF. In conclusion, some words on future work are said.

2. Modular supervisory control for DES

2.1. Supervisory control theory

Let a discrete event system is specified in the form of a finite-state automata $\mathcal{G} = (Q, \Sigma, \delta, q_0, Q_m)$ as a generator of a formal language [24]. Here Q is the set of states q; Σ is the set of events; δ: $\Sigma \times Q \to Q$ is the transition function; $q_0 \in Q$ is the initial state; $Q_m \subset Q$ is the set of marked states. $\mathcal{G}$ is also called a *plant* in the automatic control theory.

Let Σ^* denote a Kleene closure, ε is an empty string. δ is easily extended on strings from Σ^*. Language generated by $\mathcal{G}$ is $L(\mathcal{G}) = \{w : w \in \Sigma^* \text{ and } \delta(w, q_0) \text{ is defined}\}$, while language marked by $\mathcal{G}$ is $L_m(\mathcal{G}) = \{w : w \in L(\mathcal{G}) \text{ and } \delta(w, q_0) \in Q_m\}$. For any $L \subset \Sigma^*$ a *closure* of L is the set of all strings that are prefixes of words of L, i.e., $\overline{L} = \{s | s \in \Sigma^* \text{ and } \exists t \in \Sigma^* : s \cdot t \in L\}$. Symbol $\cdot$ denotes string concatenation and is often omitted. K is called *prefix-closed* if $\overline{K} = K$.

SCT supposes some events of $\mathcal{G}$ to be controllable, i.e., they may be prohibited from occurring. Let Σ_c be a controllable event set, $\Sigma_{uc} = \Sigma \setminus \Sigma_c$, $\Sigma_c \cap \Sigma_{uc} = \emptyset$. The means of control $\mathcal{G}$ is represented by a *supervisor* [24]. The supervisor observes events generated by the plant and disables undesired controllable events thus realizing a mapping $\gamma : L(\mathcal{G}) \to 2^{\Sigma_c}$. The supervisor switches control patterns in such a way that the supervised DES achieves a control objective described by some regular language K. The supervisor may be realized, for example, as a pair $\mathcal{J} = (\mathcal{S}, \phi)$ where $\mathcal{S}$ is a deterministic automaton while $\phi : X \to \Gamma$ is a function that maps automaton states x, which is a result of a string $s \in L(\mathcal{G})$ occurring, into control patterns $\gamma \in 2^{\Sigma}$. The notion of controllability characterizes languages which may be achieved by the closed-loop structure of the plant and the supervisor. A formal language K is called controllable (with respect to $L(\mathcal{G})$ and Σ_{uc}) if $\overline{K}\Sigma_{uc} \cap L(\mathcal{G}) \subseteq \overline{K}$.

A language generated by the closed-looped behavior of the plant and the supervisor is denoted as $L(\mathcal{J}/\mathcal{G})$. Let $L_m(\mathcal{J}/\mathcal{G})$ denotes the language marked by the supervisor: $L_m(\mathcal{J}/\mathcal{G}) = L(\mathcal{J}/\mathcal{G}) \cap L_m(\mathcal{G})$. The main goal of supervisory control is to construct such supervisor that $L(\mathcal{J}/\mathcal{G}) = \overline{K}$ and $L_m(\mathcal{J}/\mathcal{G}) = K$. Note that in [17] the method for controllability checking using PCF calculus is presented.

If the specification under consideration happen to be not controllable, a controllable part of it may be used for designing a supervisor. A set $\mathfrak{C}_{in}(K) = \{L \subseteq K : \overline{L}\Sigma_{uc} \cap L(\mathcal{G}) \subseteq \overline{L}\}$ is a set of all controllable sublanguages of a given language K [1]. It is well known that since the set of controllable sublanguages of a given regular language L is closed under the union, the supremal controllable sublanguage of L exists, and it is also regular. According [1], we denote this language $K^{\uparrow C}$. Note that in the worst case $K^{\uparrow C} = \emptyset$, while $K^{\uparrow C} = K$ if K is controllable.

2.2. Modular structures in SCT

As a rule, complex technical systems, being subjects of SCT implementation, are composed of smaller subsystems, called modules where each module describes some aspect of system behaviour. Each module may be realised by a generator $\mathcal{G}_i$, $i = \overline{1, \ldots, m}$. To combine modules, the operations of composition of automata are employed, known as product and parallel, or synchronous, compositions. Both operations are essential in supervisor design.

Definition 1 [Parallel composition [1]] 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 as $\mathcal{G}_1 || \mathcal{G}_2 := Ac(Q^1 \times Q^2, \Sigma_1 \cup \Sigma_2, \delta, (q_0^1, q_0^2), Q_m^1 \times Q_m^2)$, where

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

Here $Ac(\mathcal{G})$ denotes the operation of taking accessible part of the automaton, that is, deleting all states that are not reachable from q_0 by some string of $L(\mathcal{G})$ and transitions attached to these states. Using the same principle, the 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: $\mathcal{G}_1||\mathcal{G}_2||\mathcal{G}_3 := (\mathcal{G}_1||\mathcal{G}_2)||\mathcal{G}_3$.

In the parallel composition of two automata events shared by the automata must occur simultaneously thus synchronizing the automata by the common events. Events that are not shared by the automata may occur independently. In SCT parallel composition serves, among other things, to design system model from models of separate modules. If a supervisor is realized with the help of deterministic automaton $\mathcal{S}$ then the desired control can be implemented by constructing the parallel composition $\mathcal{S}||\mathcal{G}$ of the automata of the plant and the supervisor.

Note that if $\Sigma_1 = \Sigma_2$ then $\mathcal{G}_i$ are completely synchronized and the *product* $\mathcal{G}_1 \times \mathcal{G}_2$ of automata $\mathcal{G}_1$ and $\mathcal{G}_2$ is obtained. If $\mathcal{G}_i$ do not share events then they execute events concurrently, and this case also known as *shuffle* of $\mathcal{G}_i$. In [7], the plant model, describing possible behavior for a group of robots executing a set of tasks, is built using the shuffle operation, as modules for different robots do not possess common events and robots interconnection is performed only via supervisors.

2.3. Control for modular structures

There are three paradigms of supervisory control structure nowadays: monolithic, modular, and local modular supervisors.

2.3.1. Monolithic control To built a monolithic supervisor, all generator modules and specifications are composed to a single generator and a single specification. If K_i is a specification language, let $\mathcal{H}_i$ be automaton that marks language K_i. Such $\mathcal{H}_i$ is called a recognizer of K_i. A monolithic specification is obtained as

$$\mathcal{H}^{mono} = \mathcal{H}_1||\mathcal{H}_2||\dots||\mathcal{H}_m$$

while monolithic generator is

$$\mathcal{G}^{mono} = \mathcal{G}_1||\mathcal{G}_2||\dots||\mathcal{G}_n.$$

Then, if $\mathcal{H}^{mono}$ is controllable then it may be chosen as a monolithic supervisor $\mathcal{S}^{mono}$ for $\mathcal{G}^{mono}$, and $L(\mathcal{J}^{mono}/\mathcal{G}^{mono}) = L(\mathcal{S}^{mono}||\mathcal{G}^{mono})$ [1].

2.3.2. Local control The parallel composition may lead to the exponential growth of the number of states in the resulting automata for generator and specification. Therefore, a prohibitively large amount of memory can be required to store the supervisor. To mitigate this problem, modular supervisors were proposed [4]. In the modular approach, all generator modules are composed to a single generator, as it was made for the monolithic supervisor:

$$\mathcal{G}^{mod} = \mathcal{G}_1||\mathcal{G}_2||\dots||\mathcal{G}_n.$$

Then, one supervisor is designed for each specification $\mathcal{H}_i$:

$$L(\mathcal{J}_i/\mathcal{G}^{mod}) = L(\mathcal{S}_i||\mathcal{G}^{mod}).$$

The conjunction of the supervisors, realized by the parallel composition

$$\mathcal{S}^{mod} = \mathcal{S}_1||\mathcal{S}_2||\dots||\mathcal{S}_m,$$

guarantees that any event of the global plant is enabled only if all modular supervisors that have the corresponding event in their event set enable it.

The benefit of the modular approach is that the desired behavior may be achieved without constructing the parallel composition of all components. However, then nonblockingness is considered, the result of modular supervision may appear unsatisfactory, because the composition of nonblocking supervisors may block. Only nonconflicting languages may be combined in this case, i.e. such languages L_1 and L_2 that $\overline{L_1 \cap L_2} = \overline{L_1} \cap \overline{L_2}$. Fortunately, any two prefix-closed languages are nonconflicting. Moreover, if a language K is prefix-closed then $K^{\uparrow C}$ is prefix-closed as well [1]. Thus, dealing with prefix-closed specifications, after constructing $L_1^{\uparrow C}$ and $L_2^{\uparrow C}$ there is no need to check if they are conflicting, and the intersection of corresponding supervisors results in a nonblocking modular supervisor. Only prefix-closed languages are considered in this paper. Moreover, to simplify PCFs involved, the issue of blocking is also not considered here and will be discussed in our future works.

2.3.3. Local modular control In the local modular control, a local supervisor is created for each control specification but only those generator modules are taken into account for this that are affected by the particular specification, i.e. has at least one common event with this specification. Since both the modularity of specification and of the plant are exploited [5], [6] then the number of modules used in the synthesis of each supervisor is reduced what may result in smaller supervisors.

Let each specification $\mathcal{H}_i$ has its own generator model $\mathcal{G}_i^{loc}$ which is the parallel composition of all generator modules $\mathcal{G}_j^i$ that have at least one event in common with $\mathcal{H}_i$:

$$\mathcal{G}_i^{loc} = \mathcal{G}_1^i || \mathcal{G}_2^i || \ldots || \mathcal{G}_{n_i}^i.$$

Then, one supervisor $\mathcal{J}_i^{loc}$ is designed for each specification:

$$L(\mathcal{J}_i^{loc} / \mathcal{G}_i^{loc}) = L(\mathcal{S}_i^{loc} || \mathcal{G}_i^{loc}).$$

Required control as the conjunction of the local supervisors is captured by the parallel composition

$$\mathcal{S}^{locmod} = \mathcal{S}_1^{loc} || \mathcal{S}_2^{loc} || \ldots || \mathcal{S}_n^{loc}.$$

In the next sections we will show how modular supervisor can be constructed with the help of the PCF-based approach.

3. The calculus of PCFs

Consider a language of 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 term, atom, literal we define in the usual way. Hereafter, non-atomic formulas and subformulas will be denoted by capital calligraphic letters ($\mathcal{F}, \mathcal{P}, \mathcal{Q}$, etc.), in the general case with indices. Sets of formulas will be denoted by Greek capital letters (Φ, Ψ, etc.), possibly with indices.

Let $X = \{x_1, \ldots, x_k\}$ be a set of variables, $A = \{A_1, \ldots, A_m\}$ be a set of atomic formulas called *conjunct*, and $\Phi = \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$ be a set of FOFs. The following formulas $\forall x_1 \ldots \forall x_k (A_1 \& \ldots \& A_m) \rightarrow (\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n)$ and $\exists x_1 \ldots \exists x_k (A_1 \& \ldots \& A_m) \& (\mathcal{F}_1 \& \ldots \& \mathcal{F}_n)$ are denoted as $\forall x_1, \ldots, x_k A_1, \ldots, A_m \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$ and $\exists x_1, \ldots, x_k A_1, \ldots, A_m \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$. 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 $\Phi^{\vee}$ 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 formulas. Thus, if $Q \in \{\forall, \exists\}$ then $Q_X A\ \varnothing \equiv Q_X A$, $Q_X \varnothing\ \Phi \equiv Q_X\ \Phi$ and $Q_{\varnothing} A\ \Phi \equiv QA\ \Phi$. Since empty disjunction is identical to *false*, whereas empty conjunction is identical to *true*, the following equivalences are correct: $\forall_X A\ \varnothing \equiv \forall_X A \rightarrow false \equiv \forall_X A$ and $\exists_X A\ \varnothing \equiv \exists_X A \& true \equiv \exists_X A$ and $\forall \varnothing\ \Phi \equiv true \rightarrow \Phi \equiv \forall\ \Phi$ and $\exists \varnothing\ \Phi \equiv true \& \Phi \equiv \exists\ \Phi$.

Definition 2. Let X be a set of variables, and A be a conjunct, both can be empty.

(i) $\exists_X A$ is an $\exists$-PCF,

(ii) $\forall_X A$ is a $\forall$-PCF,

(iii) If $\Phi = \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$ is a set of $\forall$-PCFs, then $\exists_X A\ \Phi$ is an $\exists$-PCF,

(iv) If $\Phi = \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$ is a set of $\exists$-PCFs, then $\forall_X A\ \Phi$ is a $\forall$-PCF,

(v) Any $\exists$-PCF or $\forall$-PCF is a PCF,

(vi) There are only PCFs of a form $\exists$-PCF and $\forall$-PCF.

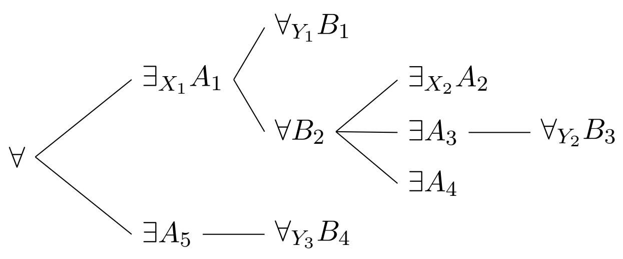

Figure 1. Tree representation of the PCF.

The term "positively" comes from the fact that according to definition 3 PCFs contain no negation operator $\neg$.

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

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

can be represented as a tree like in figure 1.

Given PCFs $\mathcal{P} = \forall\{\mathcal{F}_1, \dots, \mathcal{F}_n\}$ and $\mathcal{F}_i = \exists_{X_i} B_i \{\mathcal{Q}_{i1}, \dots, \mathcal{Q}_{im}\}$, $i = \overline{1,n}$, then $\mathcal{F}_i$ is called *base subformula* of $\mathcal{P}$, B_i is called *base of facts* or just *base*, $\mathcal{Q}_{ij}$ are called *question subformulas*, and roots of question subformulas are called *questions* to the base B_i, $i = \overline{1,n}$. A question of a form $\forall_X A$ (without any children) is called *goal question.*

Inside each of the base subformulas, any variable cannot be free and bound simultaneously. Furthermore, it cannot be bound by different quantifiers simultaneously.

Definition 3. [Answer] Consider some base subformula $\exists_X A \ \Psi$ of a PCF. A question of the subformula $\mathcal{Q} = \forall_Y B \ \Phi$, $Q \in \Psi$ has an answer θ if and only if θ is a substitution $Y \to H^\infty \cup X$ and $B\theta \subseteq A$, where H^∞ is Herbrand universe based on constant and function symbols that occur in the corresponding base subformula.

Definition 4. Let $\mathcal{P}_1 = \exists_X A \ \Psi$ and $\mathcal{P}_2 = \exists_Y B \ \Phi$, then $merge(\mathcal{P}_1, \mathcal{P}_2) = \exists_{X \cup Y} A \cup B \ \Psi \cup \Phi$.

Definition 5. Consider some base subformula $\mathcal{B} = \exists_X A \ \Psi$. A question subformula $\mathcal{Q} \in \Psi$ has the form $\forall_Y D \ \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, where $\mathcal{P}_i = \exists_{Z_i} C_i \ \Gamma_i, i = \overline{1,n}$, then $split(B, Q) = \{merge(B, \mathcal{P}'_1), \dots, merge(B, \mathcal{P}'_n)\}$, where $'$ is a variable renaming operator. We say that $\mathcal{B}$ is split by $\mathcal{Q}$, and $split(\mathcal{B}, \mathcal{Q})$ is the result of the split of $\mathcal{B}$. Obviously, $split(\mathcal{B}, \forall_Y D) = \varnothing$.

Definition 6. [The inference rule ω] Consider some PCF $\mathcal{F} = \forall \ \Phi$. If there exists a base subformula $\mathcal{B} = \exists_X A \ \Psi$, $\mathcal{B} \in \Phi$ and there exists a question subformula $\mathcal{Q} \in \Psi$, and the question of $\mathcal{Q}$ has an answer θ to $\mathcal{B}$, then $\omega(\mathcal{F}) = \forall \ \Phi \setminus \{\mathcal{B}\} \cup split(\mathcal{B}, \mathcal{Q}\theta)$.

Note, that if the set Φ becomes empty after applying the ω rule, and the PCF becomes just $\forall$, then the negation of the original statement is unsatisfiable; therefore, the statement itself is true.

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 *an inference* of $\mathcal{F}$ in the PCF calculus (with the axiom $\forall$). A search strategy used by default does not use repeated application of ω to a question with the same θ (question-answering method of automated inference search).

The ATP programming system, a prover, for PCF calculus, called *Bootfrost* [25], is realized in the Julia programming language as it is aimed mainly at scientific computing and supports JIT-compilation (compilation on the fly). It combines such features as high-level development and ease of writing program code, like in Python, and the speed of program execution comparable to programs written in C and Java. Julia has a property of homoiconicity (like Lisp, Clojure or Prolog) that allows one to use program code as data and execute it at the right time. This property is suitable for implementing additional inference strategies. Moreover, it is possible not only to set a strategy in advance but also to generate it dynamically during program execution basing on the state of the logical inference. Since the area of application of the prover is various scientific problems, the wide set of Julia scientific libraries will improve the quality of writing inference strategies for these problems. Multiple dispatching in Julia is well suited for the implementation of symbolic computations, which include the problem of finding inference, due to the elegant and efficient implementation of pattern matching used to solve the matching problem (a special case of unification). Julia also has built-in tools for developing multithreaded and distributed programs.

4. The calculus of PCFs in DES control

4.1. PCF representation of DES

Figure 2 shows a general form of a PCF representing a DES. It consists of the single base $B = \{L(\varepsilon, S_0),\ L_m(\varepsilon, S_0), \delta(S_1^i, \sigma^i, S_2^i), \delta_m(S_1^i, \sigma^i, S_2^i), \Sigma_c(\sigma^j), \Sigma_{uc}(\sigma^j)\}$, $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, k\}$, n is the number of transitions, k is the number of events, and two questions where the following predicates are exploited. The predicate $L(s, S)$ denotes "s is a current sequence of events in the state S" and the predicate $L_m(s, S)$ denotes "s is a current sequence of events in the state S, and s is a marked string". The first arguments of these atoms accumulate the strings of languages generated and marked by the automaton. S_0 corresponds to the initial state of the DES. A predicate of the form $\delta(S_1, \sigma, S_2)$ is interpreted as a transition from a state S_1 to a state S_2 due to event σ occurring. If the target state of a transition is marked, then atoms with an index m are used, i.e., $\delta_m(S_1, \sigma, S_2)$. Controlled and uncontrolled events are represented in the base by separate atoms using the predicates $\Sigma_c(\_)$ and $\Sigma_{uc}(\_)$, respectively. The function symbol "$\cdot$" denotes strings concatenation, and the "ε" symbol corresponds to the empty string. Applying the inference rules to this PCF, the words of the languages generated and marked by the automaton are constructed as the first arguments of the atoms $L(s, S)$, $L_m(s, S)$ in the base.

$$
\exists\, B \left\langle
\begin{array}{l}
\forall \sigma, s, \sigma', s'\ L(\sigma, s), \delta(s, \sigma', s') \;—\!\!—\; \exists L(\sigma \cdot \sigma', s') \\[1em]
\forall \sigma, s, \sigma', s'\ L(\sigma, s), \delta_m(s, \sigma', s') \;—\; \exists L_m(\sigma \cdot \sigma', s')
\end{array}
\right.
$$

Figure 2. General form of PCF representation of DES.

The PCF calculus allows one to construct a supremal controllable sublanguage of uncontrollable specification during the controllability checking. The corresponding PCF and the procedure is presented in [18]. Note that one essential feature of the calculus of PCFs is used for sublanguage design, namely, the possibility of constructing a *non-monotonic* inference by slightly adjusting the definition of the inference rule ω. In the non-monotonic inference, atoms may be removed from the base. In general, this operation affects the property of completeness of the PCF calculus but for the problem considered the inference always stops.

4.2. Parallel composition of automata construction

The PCF representation of constructing the parallel composition of two automata may also be written using a single PCF. The full PCF, constructing not only automaton $\mathcal{G}_1 || \mathcal{G}_2$ itself but also languages generated by this automaton, are presented in [26]. Here we provide a short version of the PCF from [26] to illustrate automaton structure design using ATP (figure 3). The PCF $\mathcal{F}_{\mathcal{G}_1 || \mathcal{G}_2}$ consists of one base subformula which base conjunct $B_{\mathcal{G}_1 || \mathcal{G}_2} = \{\delta^1(S_1^i, \sigma^i, S_2^i), \delta^2(S_1^k, \sigma^k, S_2^k), \delta^3(\phi, \varepsilon, S_0^1 S_0^2)\}$ contains atoms for transitions δ^1, $i = \overline{1, n_1}$, of

$$\begin{tikzcd}[column sep=small]
 & \begin{array}{r}\forall\sigma, s_1, p_1, s_2, p_2\ \delta^1(s_1,\sigma,s_2), \delta^2(p_1,\sigma,p_2),\\ \delta^i(\_,\_,s_1p_1)\end{array} \arrow[r, dash] & \exists\delta^3(s_1p_1,\sigma,s_2p_2) \\
\begin{array}{r}\exists\delta^3(\phi,\varepsilon,S_0^1S_0^2),\\ \delta^1(S_1^i,\sigma^i,S_2^i),\\ \delta^2(S_1^k,\sigma^k,S_2^k),\end{array} \arrow[ur, dash] \arrow[r, dash] \arrow[dr, dash] & \begin{array}{r}\forall\sigma, s_1, p_1, s_2, p_2\ \delta^1(s_1,\sigma,s_2), \sigma\in\Sigma_1, \sigma\notin\Sigma_2,\\ \delta^i(\_,\_,s_1p_1)\end{array} \arrow[r, dash] & \exists\delta^3(s_1p_1,\sigma,s_2p_1) \\
 & \begin{array}{r}\forall\sigma, s_1, p_1, s_2, p_2\ \delta^2(p_1,\sigma,p_2), \sigma\notin\Sigma_1, \sigma\in\Sigma_2,\\ \delta^i(\_,\_,s_1p_1)\end{array} \arrow[r, dash] & \exists\delta^3(s_1p_1,\sigma,s_1p_2)
\end{tikzcd}$$

Figure 3. PCF $\mathcal{F}_{\mathcal{G}_1||\mathcal{G}_2}$ constructing parallel composition of two automata.

the automaton $\mathcal{G}_1$ and transitions δ^2, $k = \overline{1, n_2}$, of the automaton $\mathcal{G}_2$, and also the atom $\delta^3(\phi, \varepsilon, S_0^1S_0^2)$ corresponding to the parallel composition automaton. It determines initial state of the composition automaton as combination of the initial states S_0^1, S_0^2 of the automata $\mathcal{G}_1$ and $\mathcal{G}_2$.

The advantage of the PCF-based method of the parallel composition constructing 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 where index i is a parameter that takes any of the values $\{1, 2, 3\}$.

4.3. Supervisor realization by PCF

After the building of the supremal controllable sublanguage of a given specification in the case when the original specification happened to be uncontrollable, this language may be taken as a new specification. Then we can find a solution to the basic problem of supervisory control for a new specification, i.e., a supervisor $\mathcal{J}$ such that $L(\mathcal{J}/\mathcal{G}) = K^{\uparrow C}$. If $\mathcal{H}'$ is a recognizer for $K^{\uparrow C}$ then using automata realization of supervisors, a closed-looped behaviour of the plant and the supervisor is realized by the parallel composition of the automata of the plant and the supervisor, i.e., $L(\mathcal{J}/\mathcal{G}) = L(\mathcal{H}'||\mathcal{G})$.

In PCF formalization, the joint work of the system and the supervisor is carried out using the PCF $\mathcal{F}_{SC}$ in figure 4. Here bases B and B_S are the sets of atoms corresponding to the transitions

$$\exists\ B, B_S \text{ --- } \forall\sigma, s, \sigma', s'\ L(\sigma, s), \delta(s, \sigma', s'), \delta^S(s, \sigma', s') \text{ --- } \exists L(\sigma\cdot\sigma', s')$$

Figure 4. The general form of a PCF realizing supervisory control.

of the plant and the supervisor, correspondingly. The only question of $\mathcal{F}_{SC}$ may be interpreted as follows. If the system is at the state s and an event σ occurs, then according to the δ_2, the system is switched to the specified state s', and σ is added to the current chain of events stored as the first argument of the predicate $L(\_,\_)$. That is, for any transition corresponding to the language $L(\mathcal{G})$ (an atom $\delta()$), we simultaneously trace the corresponding event in the automaton of the supervisor (an atom $\delta^S()$). The rule works only on those strings that are allowed by the supervisor, i.e., atoms $\delta^S()$ limit the answers that could be generated with atoms $\delta()$ only.

Note that the inference is organized in such a way as to assure a sequential accumulation of events. This means that, first of all, all possible continuations from the initial state will be added to the empty string. After then, all events from the neighbouring states will be added to all strings of the length one in the base, and so on. The search strategy can also be configured to analyze strings. For example, when all strings of a given length are generated, each next transition can be controlled in addition to the supervisor control, by applying additional rules to the strings generated. This feature of the PCF calculus may be utilized for optimal supervisory control. The mentioned rules may be defined by an operator. Moreover, the inference can be made interactive, for example, by pausing after each event, or after an event leading the system to the marked state.

4.4. Supervisor implementation for modular DES

Figure 5 shows the general PCF $\mathcal{F}_{S_{mod}}$ that represents the work of a generator composed of two modules under supervisory control aimed to guarantee a single specification. Thus, here $\mathcal{G}^{loc} = \mathcal{G}_1 || \mathcal{G}_2$ and one supervisor $\mathcal{J}^{loc}$ was previously designed such as to assure $L(\mathcal{J}^{loc}/\mathcal{G}^{loc}) = L(\mathcal{S}^{loc}||\mathcal{G}^{loc})$. The base $B_{\mathcal{S}^{loc}} = B_{\mathcal{G}_1} \cup B_{\mathcal{G}_2} \cup B_{\mathcal{S}} \cup \{L_E(\varepsilon, E_0), L_c(\varepsilon, S_0^{\mathcal{G}_1} \cdot S_0^{\mathcal{G}_2})\}$ consists of the atoms corresponding to transitions of the automata $\mathcal{G}_1$ and $\mathcal{G}_2$ (the conjuncts $B_{\mathcal{G}_1} = \{\delta^{\mathcal{G}_1}(S_1, e, S_2)\}$, $B_{\mathcal{G}_2} = \{\delta^{\mathcal{G}_2}(S_1, e, S_2)\}$) and of the automata $\mathcal{S}$ (the conjunct $B_{\mathcal{S}} = \{\delta^{\mathcal{S}}(S_1, e, S_2)\}$) defining specification language. The conjuncts $L_{\mathcal{S}}(\varepsilon, S_0)$ and $L_c(\varepsilon, S_0^{\mathcal{G}_1} \cdot S_0^{\mathcal{G}_2})$ determine initial states for controlled and specification languages generating.

The PCF $\mathcal{F}_{\mathcal{S}^{loc}}$ is based on the formula for constructing the parallel composition of automata (section 4.2). In contrast to the latter, the questions of $\mathcal{F}_{\mathcal{S}^{loc}}$ generate strings of the parallel composition of languages, bounded by the supervisor, i.e. by $\delta^{\mathcal{S}}(S_1, e, S_2)$ atoms in the questions. These questions can be interpreted as follows: if a new symbol of the language of the parallel composition of automata may be generated, and at the same time the same symbol can be generated by the automaton of the specification, then construct new strings of these languages accompanied with the states specified in the transitions, and add them to the base.

$$\begin{tikzcd}[arrows=-]
 & \begin{array}{r} \forall \sigma, \sigma', s_1, s_1', s_2, s_2', s_3, s_3'\ L_c(\sigma, s_1 \cdot s_2), \\ \delta^{\mathcal{G}_1}(s_1, \sigma', s_1'), \delta^{\mathcal{G}_2}(s_2, \sigma', s_2'), \\ L_{\mathcal{S}}(\sigma, s_3), \delta^{\mathcal{S}}(s_3, \sigma', s_3') \end{array} \arrow[r] & \exists L_c(\sigma \cdot \sigma', s_1' \cdot s_2'), L_{\mathcal{S}}(\sigma \cdot \sigma', s_3') \\
\exists B_{\mathcal{S}^{loc}} \arrow[ur] \arrow[r] \arrow[dr] & \begin{array}{r} \forall \sigma, \sigma', s_1, s_1', s_2, s_3, s_3'\ L_c(\sigma, s_1 \cdot s_2), \\ \delta^{\mathcal{G}_1}(s_1, \sigma', s_1'), \\ L_{\mathcal{S}}(\sigma, s_3), \delta^{\mathcal{S}}(s_3, \sigma', s_3'), \\ \sigma' \in \Sigma^{\mathcal{G}_1}, \sigma' \notin \Sigma^{\mathcal{G}_2} \end{array} \arrow[r] & \exists L_c(\sigma \cdot \sigma', s_1' \cdot s_2), L_{\mathcal{S}}(\sigma \cdot \sigma', s_3') \\
 & \begin{array}{r} \forall \sigma, \sigma', s_1, s_2, s_2', s_3, s_3'\ L_c(\sigma, s_1 \cdot s_2), \\ \delta^{\mathcal{G}_2}(s_2, \sigma', s_2'), \\ L_{\mathcal{S}}(\sigma, s_3), \delta^{\mathcal{S}}(s_3, \sigma', s_3'), \\ \sigma' \notin \Sigma^{\mathcal{G}_1}, \sigma' \in \Sigma^{\mathcal{G}_2} \end{array} \arrow[r] & \exists L_c(\sigma \cdot \sigma', s \cdot s_2'), L_{\mathcal{S}}(\sigma \cdot \sigma', s_3')
\end{tikzcd}$$

Figure 5. PCF for modular DES under supervisory control.

In the next section we provide an example of the inference result for the case of robot control. Required control as the conjunction of the local supervisors is captured by the parallel composition

$$\mathcal{S}^{locmod} = \mathcal{S}_1^{loc}||\mathcal{S}_2^{loc}|| \ldots ||\mathcal{S}_n^{loc}$$

that may be realized with the PCF from section 4.2.

5. Case study: robot control for moving a block

In this section we consider modular supervisor constructing for robot group control. Let on a field, called *a scene*, there be three robots, two blocks, and the target area to which it is necessary to move the blocks (figure 6). Only a pair of robots can push a block, so at first, a robot should find a companion to form a pair and only then push the block to the target area.

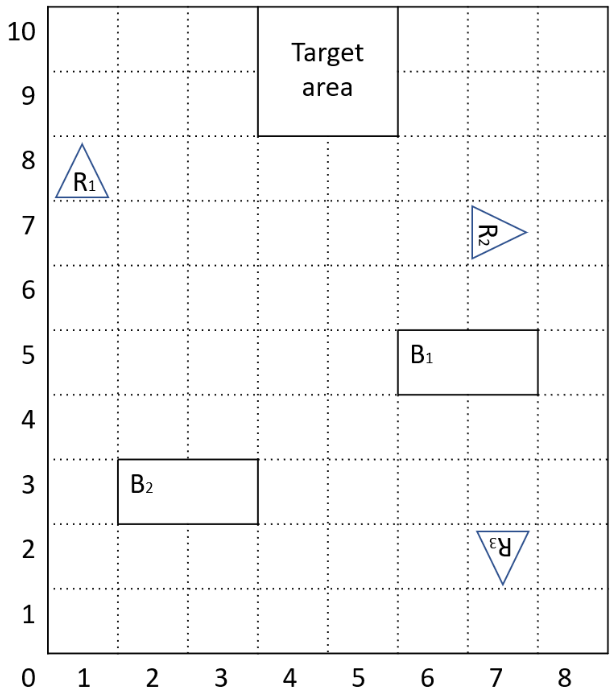

Figure 6. The initial scene.

The DES model for the problem under consideration consists of a set of generators, and each of them captures one of the robot actions. The incoming arrow denotes an initial state of an automaton while dotted lines denote controllable events. Marked states are denoted by double circles.

The automaton $\mathcal{G}_1$ in figure 7 describes forming of a group of two robots. Event rl is for "companion robot is lost", rf is for "companion robot is found". At the state 0, the robot has not yet found a companion, so the event rl does not change the state. Here $\Sigma_c^{\mathcal{G}_1} = \{rf\}$. We set rf to be controllable as the status "companion is found" is determined by external facts. Such notion as composite events may be considered for that [19].

The automaton $\mathcal{G}_2$ in figure 8 describes the search of the direction for movement to the block. Events dl is for "direction is lost", df is for "direction is found". Again, additional information is used to decide that the right direction is found.

The automaton $\mathcal{G}_3$ in figure 9 describes the possible directions of rotation of the robot to find the proper direction of moving. The robot can rotate clockwise and counterclockwise (events cw and ccw). The initial state is north orientation. Both events are controllable. The actual information, whether the direction is found or not, comes from external sources (a deducible event).

The automaton $\mathcal{G}_4$ in figure 10 is an automaton that describes the operational modes of robots. The robot can stand still, move, rotate and push. The modes restrict the robot's functions in different situations. For example, when the robot moves or pushes a block, it cannot rotate. Or, until the robot has found a companion, it should not push. This case, for example, is described

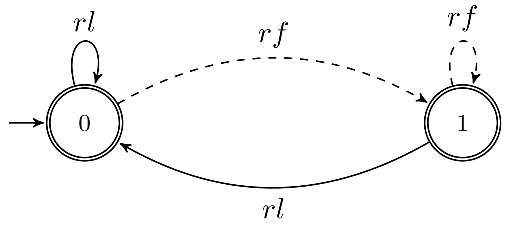

Figure 7. $\mathcal{G}_1$. Group formation for 2 robots.

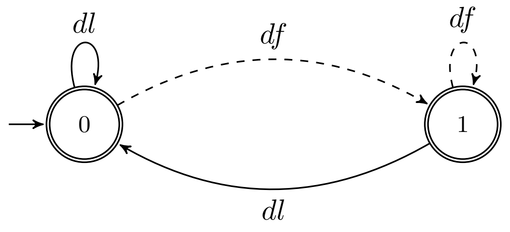

Figure 8. $\mathcal{G}_2$. Search for direction.

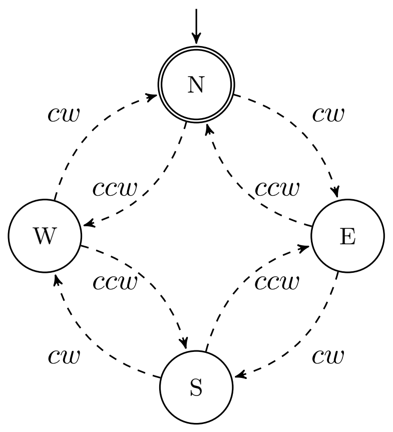

Figure 9. $\mathcal{G}_3$. Rotation.

by a combination of states of automata $\mathcal{G}_1$ and $\mathcal{G}_4$.

The automaton $\mathcal{G}_5$ (figure 11) serves for checking the achievement of the goal assigned, i.e. if the block has reached the target area. The events of $\mathcal{G}_5$ are *gc* – the goal check, *ga* – the goal is achieved, *gna* – the goal is not achieved.

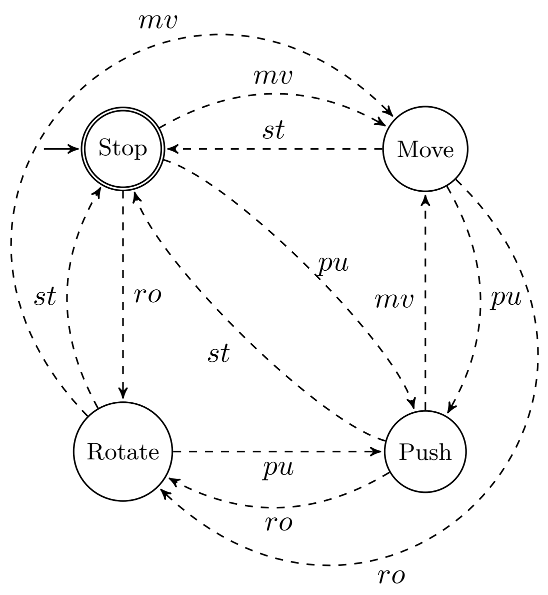

Figure 10. $\mathcal{G}_4$. Modes switching.

The parallel composition of automata $\mathcal{G}_1 - \mathcal{G}_5$ represents the current state of the robot. Figure 12 depicts a specification automaton $\mathcal{H}_1$. This specification requires that at first, the robot looks

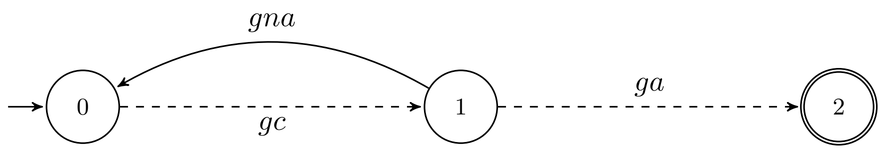

Figure 11. $\mathcal{G}_5$. Goal checking.

for a partner. Then it finds the right direction, then moves, pushes or rotates, depending on what action is required for the current position of the block. After that, the goal achievement is checked. If the goal is not achieved the actions continue. If the goal has been achieved the task assigned has been solved.

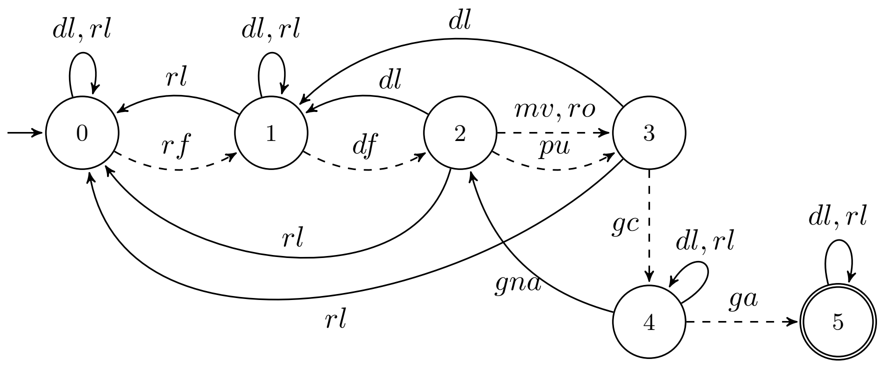

Figure 12. Specification $\mathcal{H}_1$. The main strategy.

The specification automaton $\mathcal{H}_2$ (figure 13) limits the search for directions. If the direction has not yet been found, then the robot can only rotate clockwise or move.

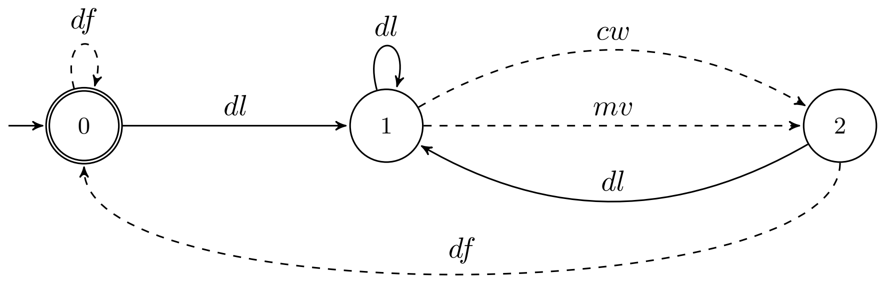

Figure 13. Specification $\mathcal{H}_2$. Finding proper direction.

Consider modular supervisor realization for the modular plant representing robot actions. Note that the specification language $L(\mathcal{H}) = L(\mathcal{H}_1||\mathcal{H}_2)$ is controllable. Since checking controllability is not considered in this paper, we omit the proof of this statement. Due to controllability of $L(\mathcal{H})$, it may be taken as a supervisor for the plant $\mathcal{G} = \mathcal{G}_1|| \dots ||\mathcal{G}_5$.

According to the local modular control paradigm, we take plant modules $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_4$ $\mathcal{G}_5$ and specification $\mathcal{H}_1$ to design one supervised subsystem and plant modules $\mathcal{G}_2$ and $\mathcal{G}_4$, and specification $\mathcal{H}_2$ to design another supervised subsystem. Consider the second subsystem. Transitions of automata $\mathcal{G}_2$, $\mathcal{G}_4$ and $\mathcal{H}_2$ are put in the base of the PCF in figure 5. The first question of the resulting formula has no answers because $\mathcal{G}_2$ and $\mathcal{G}_4$ have no events in common. Answering the second question with the substitution $\{\sigma \to \varepsilon, \sigma' \to dl, s_1 \to 0, s'_1 \to 0, s_2 \to St, s_3 \to 0, s'_3 \to 1\}$, we add new atoms $L_c(\varepsilon \cdot dl, 0 \cdot St), L_{\mathcal{S}}(\varepsilon \cdot dl, 1)$ in the base, which means that the supervisor permitted the transition to the state 1 labeled with the event dl. Next, these atoms are used to find a new answer to the third question: $\{\sigma \to \varepsilon \cdot dl, \sigma' \to mv, s_1 \to 0, s_2 \to St, s'_2 \to Mv, s_3 \to 1, s'_3 \to 2\}$, which gives new atoms $L(\varepsilon \cdot dl \cdot mv, 0 \cdot Mv)$, $L_{\mathcal{S}}(\varepsilon \cdot dl \cdot mv, 2)$. This means that the supervisor permits the event mv leading to the state 2. And so on, the inference will never end. Table 1 shows the first few steps of the inference found by the prover *Bootfrost*. It contains the strings of the language $L(\mathcal{J}_2^{loc}/\mathcal{G})$ extracted from the first arguments of the atoms $L_{\mathcal{S}}(\_,\_)$, and states corresponding to these strings.

Table 1. The inference of the PCF $\mathcal{F}_{SC}$ constructing the language $L(\mathcal{J}_2^{loc}/\mathcal{G})$.

Step	Question#	String	State
1	2	dl	1
2	3	$dl \cdot mv$	2
3	2	df	0
4	2	$dl \cdot dl$	1
5	3	$dl \cdot dl \cdot mv$	2
6	2	$dl \cdot mv \cdot dl$	1
7	2	$dl \cdot mv \cdot df$	0
8	2	$df \cdot df$	0
9	2	$df \cdot dl$	1
10	3	$df \cdot dl \cdot mv$	2

The design of the first controlled subsystem may be represented by the PCF in figure 14. This formula is also based on the PCF in figure 5. The base contains atoms corresponding to transitions of generators $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_4$ $\mathcal{G}_5$ that have no common events, so the analogue of the first question is not considered as it handles such events only. The second and the third questions, concerning private events, are transformed as shown in the figure.

Overall control assuring both specifications $\mathcal{H}_1$ and $\mathcal{H}_2$ is achieved by the conjunction of the local supervisors: $\mathcal{S}^{locmod} = \mathcal{S}_1^{loc}||\mathcal{S}_2^{loc}$ that is realized with the PCF from section 4.2.

Conclusion

In this paper, the application of the PCF calculus to modular DES control was presented and illustrated with the case study of mobile robots pushing a block to a target area. Realization of the results obtained on the robotic stand is our nearest future work. Further work also supposes solving other SCT problems such as partially observed DES study, design of decentralized supervisors in the automata form, and DES diagnostic. Note that an approach for testing the diagnosability of DES based on a logical representation is proposed in [27]. In contrast to the logical formalism proposed in this paper, [27] uses a less expressive means to represent automata underlying DES: the Conjunctive Normal Form (CNF). DES transitions are described as a set of clauses, which is taken as a new model for DES. Based on the well-known resolution method, an algorithm is presented to test whether failure events can be detected or not for a finite number of observable events. A computational comparison of this study with the PCF-based approach proves to be an interesting challenge. Results obtained will be embedded at the different levels of the hierarchical control system for mobile robots.

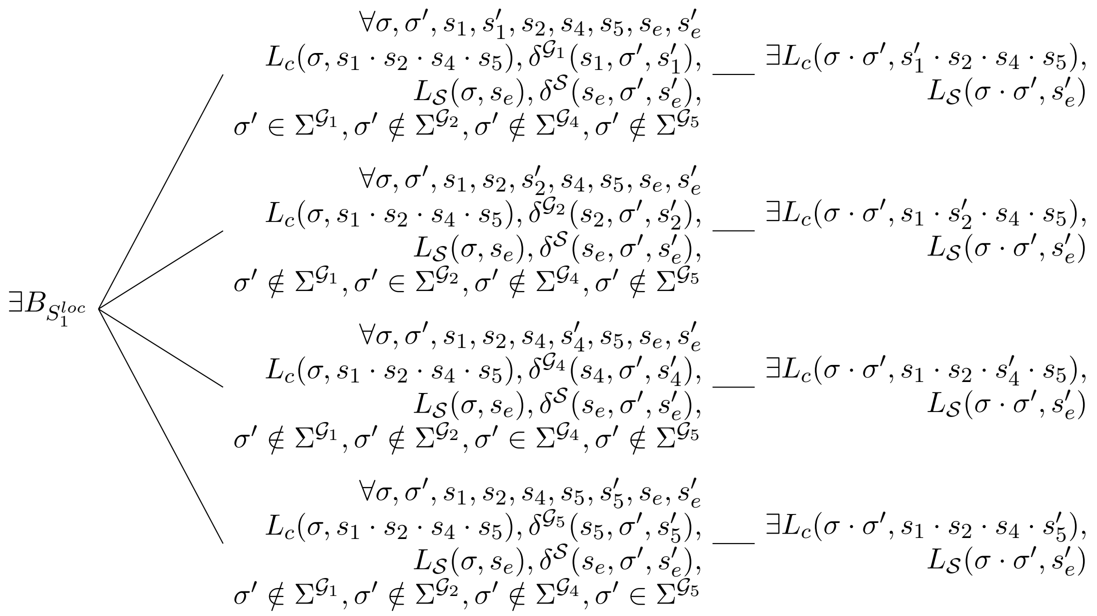
$\exists B_{S_1^{loc}}$
$\forall \sigma, \sigma', s_1, s_1', s_2, s_4, s_5, s_e, s_e'$
$L_c(\sigma, s_1 \cdot s_2 \cdot s_4 \cdot s_5), \delta^{\mathcal{G}_1}(s_1, \sigma', s_1'),$
$L_{\mathcal{S}}(\sigma, s_e), \delta^{\mathcal{S}}(s_e, \sigma', s_e'),$
$\sigma' \in \Sigma^{\mathcal{G}_1}, \sigma' \notin \Sigma^{\mathcal{G}_2}, \sigma' \notin \Sigma^{\mathcal{G}_4}, \sigma' \notin \Sigma^{\mathcal{G}_5}$
$\exists L_c(\sigma \cdot \sigma', s_1' \cdot s_2 \cdot s_4 \cdot s_5),$
$L_{\mathcal{S}}(\sigma \cdot \sigma', s_e')$
$\forall \sigma, \sigma', s_1, s_2, s_2', s_4, s_5, s_e, s_e'$
$L_c(\sigma, s_1 \cdot s_2 \cdot s_4 \cdot s_5), \delta^{\mathcal{G}_2}(s_2, \sigma', s_2'),$
$L_{\mathcal{S}}(\sigma, s_e), \delta^{\mathcal{S}}(s_e, \sigma', s_e'),$
$\sigma' \notin \Sigma^{\mathcal{G}_1}, \sigma' \in \Sigma^{\mathcal{G}_2}, \sigma' \notin \Sigma^{\mathcal{G}_4}, \sigma' \notin \Sigma^{\mathcal{G}_5}$
$\exists L_c(\sigma \cdot \sigma', s_1 \cdot s_2' \cdot s_4 \cdot s_5),$
$L_{\mathcal{S}}(\sigma \cdot \sigma', s_e')$
$\forall \sigma, \sigma', s_1, s_2, s_4, s_4', s_5, s_e, s_e'$
$L_c(\sigma, s_1 \cdot s_2 \cdot s_4 \cdot s_5), \delta^{\mathcal{G}_4}(s_4, \sigma', s_4'),$
$L_{\mathcal{S}}(\sigma, s_e), \delta^{\mathcal{S}}(s_e, \sigma', s_e'),$
$\sigma' \notin \Sigma^{\mathcal{G}_1}, \sigma' \notin \Sigma^{\mathcal{G}_2}, \sigma' \in \Sigma^{\mathcal{G}_4}, \sigma' \notin \Sigma^{\mathcal{G}_5}$
$\exists L_c(\sigma \cdot \sigma', s_1 \cdot s_2 \cdot s_4' \cdot s_5),$
$L_{\mathcal{S}}(\sigma \cdot \sigma', s_e')$
$\forall \sigma, \sigma', s_1, s_2, s_4, s_5, s_5', s_e, s_e'$
$L_c(\sigma, s_1 \cdot s_2 \cdot s_4 \cdot s_5), \delta^{\mathcal{G}_5}(s_5, \sigma', s_5'),$
$L_{\mathcal{S}}(\sigma, s_e), \delta^{\mathcal{S}}(s_e, \sigma', s_e'),$
$\sigma' \notin \Sigma^{\mathcal{G}_1}, \sigma' \notin \Sigma^{\mathcal{G}_2}, \sigma' \notin \Sigma^{\mathcal{G}_4}, \sigma' \in \Sigma^{\mathcal{G}_5}$
$\exists L_c(\sigma \cdot \sigma', s_1 \cdot s_2 \cdot s_4 \cdot s_5'),$
$L_{\mathcal{S}}(\sigma \cdot \sigma', s_e')$

Figure 14. PCF assuring specification $\mathcal{H}_1$.

Acknowledgments

The research was partly supported by the RFBR, project no. 20-07-00397 (section 5), and by the Ministry of Science and Higher Education of the Russian Federation, project no.121032400051-9.

References

[1] Cassandras C G and Lafortune S 2008 *Introduction to Discrete Event Systems* (Springer US)

[2] Seatzu C, Silva M and van Schuppen J H (eds) 2013 *Control of discrete-event systems* (Springer London)

[3] Wonham W M and Cai K 2019 *Supervisory Control of Discrete-Event Systems* (Springer International Publishing)

[4] Wonham W and Ramadge P 1988 *Mathematics of Control, Signals, and Systems* **1** 13–30 ISSN 0932-4194

[5] de Queiroz M H and Cury J E R 2000 *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)* vol 6 pp 4051–4055 vol.6

[6] Hering de Queiroz M and Cury J 2000 **1**

[7] Hill R C and Lafortune S 2017 *2017 American Control Conference (ACC)* pp 3840–3847

[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

[9] Vassilyev S N 1990 *The Journal of Logic Programming* **9** 235–266

[10] Zherlov A K, Vassilyev S N, Fedosov E A and Fedunov B E 2000 *Intelligent control of dynamic systems* (Moscow: Fizmatlit) in Russian

[11] Davydov A, Larionov A and Cherkashin E 2011 *Automatic Control and Computer Sciences* **45** 402–407

[12] Larionov A, Davydov A and Cherkashin E 2013 *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija* **2013** 1023–1028

[13] Davydov A and Larionov A 2020 *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)* vol 1 pp 727–732

[14] Kovács L and Voronkov A 2013 *Computer Aided Verification* ed Sharygina N and Veith H (Berlin, Heidelberg: Springer Berlin Heidelberg) pp 1–35 ISBN 978-3-642-39799-8

[15] Otten J 2016 *Proceedings of the 8th International Joint Conference on Automated Reasoning - Volume 9706* (Berlin, Heidelberg: Springer-Verlag) p 302–312 ISBN 9783319402284 URL `https://doi.org/10.1007/978-3-319-40229-1{\_}21`

[16] Schulz S, Cruanes S and Vukmirović P 2019 *Proc. of the 27th CADE, Natal, Brasil* (*LNAI* no 11716) ed Fontaine P (Springer) pp 495–507

[17] Davydov A, Larionov A and Nagul N 2020 *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)* pp 1151–1156

[18] Davydov A, Larionov A and Nagul N V 2020 *Proceedings of the 2nd International Workshop on Information, Computation, and Control Systems for Distributed Environments, ICCS-DE 2020, Irkutsk, Russia, July 6-7, 2020* (*CEUR Workshop Proceedings* vol 2638) ed Bychkov I and Tchernykh A (CEUR-WS.org) pp 68–78

[19] Davydov A, Larionov A and Nagul N 2021 *J. Phys.: Conf. Series* **1864** 012048

[20] Davydov A and Larionov A 2020 *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)* vol 1 pp 727–732

[21] Karpas E and Magazzeni D 2020 *Annual Review of Control, Robotics, and Autonomous Systems* **3** 417–439

[22] Zombori Z, Urban J and Brown C E 2020 *International Joint Conference on Automated Reasoning* (Springer) pp 489–507

[23] Schader M and Luke S 2020 *International Conference on Practical Applications of Agents and Multi-Agent Systems* (Springer) pp 224–237

[24] Ramadge P J and Wonham W M 1987 *SIAM Journal on Control and Optimization* **25** 206–230

[25] Pcfbucket 2021 URL `http://bootfrost.org`

[26] Davydov A, Larionov A and Nagul N V 2019 *Proceedings of the 1st International Workshop on Information, Computation, and Control Systems for Distributed Environments, ICCS-DE 2019, Irkutsk, Russia, July 8-9, 2019* (*CEUR Workshop Proceedings* vol 2430) ed Bychkov I and Tchernykh A (CEUR-WS.org) pp 29–41 URL `http://ceur-ws.org/Vol-2430/paper3.pdf`

[27] Geng X, Ouyang D and Han C 2020 *Chinese Journal of Electronics* **29** 304–311