computation

MDPI

Article

Analysis and Control of Partially Observed Discrete-Event Systems via Positively Constructed Formulas

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 St., 664033 Irkutsk, Russia; artem@icc.ru (A.D.); bootfrost@zoho.com (A.L.)
* Correspondence: sapling@icc.ru

Abstract: This paper establishes a connection between control theory for partially observed discrete-event systems (DESs) and automated theorem proving (ATP) in the calculus of positively constructed formulas (PCFs). The language of PCFs is a complete first-order language providing a powerful tool for qualitative analysis of dynamical systems. Based on ATP in the PCF calculus, a new technique is suggested for checking observability as a property of formal languages, which is necessary for the existence of supervisory control of DESs. In the case of violation of observability, words causing a conflict can also be extracted with the help of a specially designed PCF. With an example of the problem of path planning by a robot in an unknown environment, we show the application of our approach at one of the levels of a robot control system. The prover Bootfrost developed to facilitate PCF refutation is also presented. The tests show positive results and perspectives for the presented approach.

Keywords: positively constructed formula; automated theorem proving; discrete-event system; supervisory control; partial observability

Citation: Davydov, A.; Larionov, A.; Nagul, N. Analysis and Control of Partially Observed Discrete-Event Systems via Positively Constructed Formulas. *Computation* **2024**, *12*, 95. https://doi.org/10.3390/computation12050095

Academic Editor: Simeone Marino

Received: 30 November 2023
Revised: 27 April 2024
Accepted: 6 May 2024
Published: 9 May 2024

1. Introduction

The class of discrete-event systems (DESs) is a wildly used modeling formalism for a large variety of man-made complex objects [1–3]. Logical DESs, as a subclass of DESs, represent system evolution in terms of states changing in response to the occurrence of some events at non-predetermined time instants. A logical DES is commonly represented as a finite-state automaton, the transitions of which from state to state are labeled with the letters of some finite alphabet and correspond to events occurring in the system. Sequences of such transitions form the words of a regular language that describes the behavior of the system from the high-level, or symbolic, point of view. Consequently, system properties may be described as statements over these formal expressions.

One of the most popular ways to deal with logical DESs containing events that may be switched off is the supervisory control theory (SCT) [4]. SCT was developed as a tool for restricting DES behavior according to a set of constraints defined by some specification. For example, logical DESs are extensively exploited nowadays in mobile robots and robot group control, e.g., [5–7]. A detailed description of SCT is presented in, e.g., [1,8,9].

It is well known that formal logic aims at dealing with symbolic structures, such as formal languages, by formulating their properties as theorems that must be proven. Thus, studying logical DESs is naturally embraced by the paradigm of automated theorem proving (ATP). ATP represents an implementation of a natural human reasoning process with the help of formal logic and special computer programs called provers. Developed to help mathematicians in producing and verification of formal mathematical proofs, nowadays it is useful in program analysis, system verification, etc., providing results in a wide range of areas. The latest examples include proving the 400-year-old Kepler conjecture on sphere packing [10] and the correctness of the seL4 operating system kernel [11], not

to mention earlier proof of the four-color theorem in graph theory [12] and the verified C compiler CompCert [13]. A modern domain of ATP application is robotics, where it helps mostly in planning [14] and decision making [15]. For example, in [16] for planning and control in swarm robotics, the PDDL language is used, which is based on the classical STRIPS-style ATP. In [17], theorem proving is applied for verification of the framework for modeling the controllers of autonomous robots, combined with the automatic generation of C++ code.

We suggest a new way to study and design logical DESs that is based on ATP in the calculus of positively constructed formulas (PCFs). PCFs are first-order formulas that do not contain the negation symbol in their syntax [18,19]. The language of PCFs is a complete first-order language, providing a powerful tool for qualitative analysis of dynamical and intelligent systems. Its applications include telescope orientation [20], elevators group control [21], pursuing goals [22], and achieving targets [23]. To facilitate the inference search in the PCF language, a prover Bootfrost is developed [24]. The most important features of the PCF calculus and its prover implementation are the following:

- Large block data structures for representing formulas and inference rules;
- Absence of necessity to remove existence quantifiers with a skolemization procedure, which decreases the complexity of the inference;
- Compatibility with application-specific heuristics and general logical inference control heuristics;
- Clarity of the logical inference, which helps to find formalization errors;
- Support for the equality predicate;
- Modifiability of semantics to support nonclassical logics.

In modern ATP, wide usage of proof assistants, noted in [25], such as Isabelle/HOL in [10], HOL Light and Isabelle in [11], or Coq in [12], instead of fully automated provers, is caused by the fact that significant user input is often required to clarify a difficult situation that has stumbled the inference. In addition to its other advantages, the prover Bootfrost combines the power of automated reasoning with the ability to implement user-designed strategies.

Our previous papers addressed how the basic problems of the supervisory control theory (SCT) [4] for logical DESs can be solved using the PCF-based technique. The issues considered so far include controllability checking to determine if a formal language restricting DES functioning may be guaranteed by a supervisor, a supremal controllable sublanguage of a given specification language construction, or a monolithic supervisor realization (e.g., [26]). This paper deals with partially observed DESs, in which the occurrence of some events is unavailable for observation. In this case, the property called observability of a formal language determines those specifications on DES functioning that may be ensured by supervisory control. Some effective tests for observability have been already suggested, e.g., in [27,28], where the former considers the observation function in the form of a mask, and the latter is based on the algebraic operations on processes that represent DESs. For regular languages, a fixed-point test for observability from [29] can be effectively implemented. The test employs the operator that is also associated with the algorithm for computing the infimal prefix-closed and observable superlanguage of a given language. It is used in [29] to find a solution for the supervisory control and observation problem in its general case, i.e., when a supervisor should provide a language lying in some predefined range of languages.

This paper extends the use of PCF calculus and ATP in SCT problems. Namely, it will be shown how the polynomial time algorithm for testing observability from [1] may be implemented with PCFs. If observability is violated, then the conflicting strings can also be found with the help of another PCF that will be also presented. The main advantage of the presented PCF-based approach and the employment of the ATP technique is the declarative description of the used algorithms. In this case, the programmer only describes (declares) the properties of the required result, and the solution (method) is provided by the logical programming system, as a result of searching for the logical conclusion of some goal

statement. This is a step up from programming in imperative languages (e.g., C/C++, Java, etc.), as the programmer does not have to worry about the low-level details of the program.

There are some tools developed for the analysis and design of controlled DESs in the framework of SCT. Among them are TCT [30], DESUMA/UMDES [31,32], Supremica [33], and others. Having user-friendly graphical interface and high-performance algorithms, Supremica supports extended finite-state machines where transitions are labeled with guards and actions in addition to events [34]. The guards and actions reference variables, which can be declared over finite integer ranges or as enumerated type. Although a PCF-based tool for SCT is a developing project, our approach suggests that both guards and actions may be expressed in the form of logical statements of any kind. Unlike focusing on large-scale industrial examples like in [35–37], the usage of PCFs and the PCF calculus for DES control suggests exploiting logical tools for knowledge representation and processing.

The contribution of this work consists in presenting a theoretical base of the original logical approach to handling partially observed DESs. An ATP-based technique of DES specifications testing for observability is presented with the usage of PCF representation of logical DESs. This paper's structure is the following. After the necessary preliminaries on the PCF calculus provided in the next section, in Section 3, we state the problem of the supervisory control of partially observed DESs and present a PCF for observability checking. Section 4 provides a PCF extracting conflicting strings. In Section 5, the PCF-based implementation of the supervisory control of DESs is described. Section 6 is dedicated to the prover Bootfrost and its main features. Section 7 provides a case study for the path-following problem for a mobile robot. In the conclusion, we discuss the presented approach and outline directions for our future work.

2. The PCF Calculus

The calculus of positively constructed formulas (PCFs) is based on the refutation of the negation of an original statement which is to be proved. The main idea is the following: if the negation of the statement is proved to be false, then the statement itself is true. The language of PCFs is a special variant of the language of first-order logic (FOL), which consists of first-order formulas (FOFs) built out of atomic formulas, or atoms, with the help of operators &, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, quantifier symbols $\forall$ and $\exists$, and constants *True* and *False*. We suppose the reader is familiar with the concepts of atom, literal, and term, understood in the usual sense.

2.1. The PCF Language

Let $X = \{x_1, \ldots, x_k\}$ be a set of variables, $A = \{A_1, \ldots, A_m\}$ a set of atomic formulas, and $F = \{F_1, \ldots, F_n\}$ a set of subformulas. Then, the formulas $((\forall x_1) \ldots (\forall x_k)(A_1 \& \ldots \& A_m \rightarrow (F_1 \vee \ldots \vee F_n)))$ and $((\exists x_1) \ldots (\exists x_k)(A_1 \& \ldots \& A_m \& (F_1 \& \ldots \& F_n)))$ are denoted as $\forall_X A \colon F$ and $\exists_X A \colon F$, respectively, keeping in mind that the $\forall$–quantifier corresponds to the disjunction of all subformulas, and $\exists$–quantifier corresponds to their conjunction. If $F = \varnothing$, then the above formulas turn to the form $\forall_X A \colon \varnothing \equiv \forall_X A \rightarrow False$ and $\exists_X A \colon \varnothing \equiv \exists_X A \& True$, since the empty disjunction is understood as *False*, whereas the empty conjunction is understood as *True*. Let $\forall_X A$ and $\exists_X A$ be abbreviations of such formulas. If $X = \varnothing$ then $\forall A \colon F$ and $\exists A \colon F$ are analogous abbreviations. The set of atoms A is called a *conjunct*. Variables from X, bound by corresponding quantifiers, are called $\forall$–*variables* and $\exists$–*variables*, respectively. In $\forall_X A$, a variable from X that does not appear in the conjunct A is called an *unconfined* variable. Note that $\forall\varnothing \equiv \forall\varnothing \colon \varnothing \equiv \forall True \rightarrow False \equiv False$.

Definition 1 (Positively constructed formulas (PCF))**.** *Let X be a set of variables and A a conjunct. Then, the following holds:*

1. *$\exists_X A$ and $\forall_X A$ are $\exists$–PCF and $\forall$–PCF, respectively.*
2. *If $F = \{F_1, \ldots, F_n\}$ is a set of $\forall$–PCFs, then $\exists_X A \colon F$ is a $\exists$–PCF.*
3. *If $F = \{F_1, \ldots, F_n\}$ is a set of $\exists$–PCFs, then $\forall_X A \colon F$ is a $\forall$–PCF.*
4. *Any $\exists$–PCF or $\forall$–PCF is a PCF.*

A PCF starting with $\forall\varnothing$ is called a PCF in the *canonical form*. Any PCF can be represented in the canonical form. If F is a noncanonical $\exists$–PCF, then $\forall\varnothing\colon F$ is the canonical PCF, since $\forall\varnothing\colon F \equiv True \to F \equiv F$. If F is a noncanonical $\forall$–PCF, then the canonical PCF is $\forall\varnothing\colon \{\exists\varnothing\colon F\} \equiv True \to True\&F \equiv F$. Type quantifiers $\forall\varnothing$ and $\exists\varnothing$ are used to regularize PCFs, i.e., to transform them to the canonical form.

The term "positively" comes from the fact that according to the definition, PCFs contain no negation operator ($\neg$). This is also true for the semantics of language; negation is "concealed" in the implications that are meant following universal quantifiers. Note that any FOFs can be represented as PCFs, since the PCF language is a special notation of classical FOFs, as well as the conjunctive normal form, the disjunctive normal form, etc. The converting algorithm is presented in [38].

For easier reading, a PCF may be represented graphically as a tree structure. For example, consider a PCF representation of a FOF

$$\mathcal{F} = \neg\big(\forall x \exists y P(x,y) \to \exists z P(z,z)\big).$$

An image $\mathcal{F}'$ of $\mathcal{F}$ in the PCF language is $\mathcal{F}' = \forall\colon \varnothing\{\exists\colon \varnothing\{\forall x\colon \varnothing\{\exists y\colon P(x,y)\}, \forall z\colon P(z,z)\ \{\exists\colon False\}\}\}$. The tree-like form of the latter is

$$\forall\colon \varnothing \text{ - } \exists\colon \varnothing \begin{cases} \forall x\colon \varnothing \text{ —— } \exists y\colon P(x,y) \\ \forall z\colon P(z,z) \text{ — } \exists\colon False. \end{cases}$$

The root $\forall\varnothing$ of a PCF tree is called a PCF *root*. Each PCF root's child $\exists_X A$ is called a PCF *base*, the conjunct A is called a *base of facts*, and a PCF rooted from the base is called a *base subformula*. The PCF base children $\forall_Y B$ are called *questions* to the parent base. The subtrees of the questions are called *consequents*. If a question has no consequent, then the question is referred to as *goal question*, and it is identical to *False*.

2.2. The Inference Rule

The only axiom of the PCF calculus is $\forall\varnothing\colon \varnothing$, i.e., *False*. The inference rule ω in the PCF calculus is based on the search for so-called *answering substitutions*, i.e., such substitutions of variables in terms that satisfy certain conditions.

Definition 2 (Answer). *A question $\forall_Y D$: Y to a base $\exists_X A$ has an* answer θ *if and only if θ is a substitution $Y \to H^\infty \cup X$ and $D\theta \subseteq A$, where H^∞ is Herbrand universe based on constant and function symbols that occur in corresponding base subformulas.*

Definition 3 (Splitting). *Let $B = \exists_X A\colon \Psi$, and $Q = \forall_Y D\colon \Upsilon$, where $\Upsilon = \{\exists_{Z_1} C_1\colon \Gamma_1, \ldots, \exists_{Z_n} C_n\colon \Gamma_n\}$ then $split(B,Q) = \{\exists_{X\cup Z_1'} A \cup C_1'\colon \Psi \cup \Gamma_1', \ldots, \exists_{X\cup Z_n'} A \cup C_n'\colon \Psi \cup \Gamma_n'\}$, where $'$ is a variable renaming operator. We say that B is split by Q. Obviously, $split(B, \forall_Y D) = split(B, \forall_Y D\colon \varnothing) = \varnothing$.*

Definition 4 (Inference rule ω). *Consider some canonical PCF $F = \forall\varnothing\colon \Phi$. Let there exists a question Q that has an answer θ to appropriate base $B \in \Phi$, then $\omega F = \forall\varnothing\colon \Phi \setminus \{B\} \cup split(B, Q\theta)$.*

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 base subformula *is refuted* because $split(B, \forall_Y D) = \varnothing$. The refuted base subformula B is removed from the set of base subformulas Φ since $\Phi \setminus \{S\} \cup \varnothing = \Phi \setminus \{S\}$. When all bases subformulas in Φ have been refuted, the formula F is also refuted since $\forall\varnothing\colon \varnothing \equiv False$.

Any finite sequence of PCFs $\mathcal{F}, \omega\mathcal{F}, \omega^2\mathcal{F}, \ldots, \omega^n\mathcal{F}$, 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 $\forall$). An answer to the goal question refutes the corresponding base. When all bases of the PCF are refuted, then the PCF $\mathcal{F}$ is reduced to $\forall$, i.e., *False*. This means that $\mathcal{F}$ as the negation of the

statement under consideration is unsatisfiable; therefore, the statement itself is true. The details on the PCF calculus may be found in [18,19,21,39]. In [19], the correctness and completeness of the PCF calculus with functional symbols was proven.

3. Checking Observability of a Regular Language

3.1. Partially Observable DESs

Consider a discrete event system (DES) in the form of a generator $G = (Q, \Sigma, \delta, q_0, Q_m)$ of a formal language [4]. Here, Q is the set of states q; Σ the set of events; δ: $\Sigma \times Q \to Q$ the transition function; $q_0 \in Q$ the initial state; $Q_m \subset Q$ the set of marked states. Let $L(G)$ be a language generated by G, and $L_m(G)$ be a language marked by G. The Ramadge–Wonham supervisory control framework assumes the existence of a means of control G presented by a *supervisor* [4]. Let Σ_c be a controllable event set, $\Sigma_{uc} = \Sigma \setminus \Sigma_c$, $\Sigma_c \cap \Sigma_{uc} = \emptyset$. The supervisor switches control patterns so that the supervised discrete event systems achieve a control objective described by some regular language K.

Let G be partially observable, i.e., a set Σ_o of observable events is distinguished from all events, $\Sigma_{uo} = \Sigma \setminus \Sigma_o$, $\Sigma_c \cap \Sigma_{uo} = \emptyset$. The observation function is usually defined as the natural projection $P : \Sigma^* \to \Sigma_o^*$, which erases unobservable events for $s \in \Sigma^*$ $P(s\sigma) = P(s)\sigma$ if $\sigma \in \Sigma_o$ and $P(s\sigma) = P(s)$ if $\sigma \in \Sigma_{uo}$. The supervisor only observes events from Σ_o and, based on this information, disables events in Σ_c. Denote $L(J/G)$ a language generated by the closed-looped behavior of the plant and the supervisor. In this paper, for simplicity of presentation, the marked language and related problems, such as the construction of nonblocking supervisory control, are not considered.

Supervisory control and observation problem (SCOP). Given a plant G over an alphabet Σ, a language $L_A \subseteq L(G)$, a language $L_E \subseteq L(G)$, and sets Σ_o, $\Sigma_c \subseteq \Sigma$, construct a supervisor J for G such that $L_A \subseteq L(J/G) \subseteq L_E$.

The less complex problem consists in finding such control patterns that the language marked by the supervisor is equal to some desired language. Thus, the special case of SCOP is constructing such a supervisor that $L(J/G) = \overline{K}$ where K is called a *specification* language. We refer to this problem as basic SCOP (*BSCOP*). The notions of controllable and observable languages are essential in solving this problem. Let $\overline{L}$ be a set of all strings that are prefixes of words of L, i.e., $\overline{L} = \{s | s \in \Sigma^* \text{ and } \exists t \in \Sigma^* : s \cdot t \in L\}$.

Definition 5. *A language K is called* controllable *(with respect to $L(G)$ and Σ_{uc}) if $K\Sigma_{uc} \cap L(G) \subseteq \overline{K}$. Here, $\overline{K}\Sigma_{uc}$ is a shortened expression denoting concatenations of all the strings from $\overline{K}$ with any of the symbols from the set Σ_{uc}.*

Definition 6. *The K is* observable *(with respect to $L(J)$ and P) if $\forall s, t \in \Sigma^*$ $(P(s) = P(t) \to (\forall \sigma \in \Sigma)(s\sigma \in \overline{K}\ \&\ t\sigma \in L(J)\ \&\ t \in \overline{K} \to t\sigma \in \overline{K}))$.*

If observability holds true, this means that no event should be enabled and disabled simultaneously to satisfy specification K. The opposite situation is called a *conflict*.

Supervisor existence criterion for BSCOP sounds as follows: given $K \subseteq L(G)$, there exists a supervisor J such that $L(J/G) = \overline{K}$ iff K is controllable and observable with respect to $L(G)$ and P.

The PCF-based procedure for checking controllability was suggested in [26]. It also allows one to construct supremal controllable sublanguage of uncontrollable specification such that it may be chosen as a new specification and be ensured by a proper supervisor. In the rest of the paper, we show how PCFs allow testing regular language for observability and implementing supervisory control.

3.2. Checking Observability via PCFs

There are several algorithms exist to check if the regular language K is observable. Among them is the algorithm from [1] that is polynomial with respect to the size of the automata generating K. The main idea of the algorithm is constructing an automaton for

tracking two words s_1, s_2 of $\overline{K}$ which have the same projection $P(s_1) = P(s_2)$ but such that $s_1\sigma \in K$ while $s_2\sigma \notin K$. Strings s_1 and s_2 are then called conflicting because they demand different control actions from the supervisor. Let the regular language K be recognized by the finite-state automaton H. The algorithm from [1] suggests to consider two copies of the automaton H and one copy of the automaton G and to design an automaton T with the states of the form (h_1, h_2, q), where $h_1 \in Q_H$, $h_2 \in Q_H$, $q \in Q_G$, and the single state *dead*. The existence of the state *dead* denotes the unobservability of K, because in this case, there exist a set of strings $s_i \in \overline{K}$, $i = 1, 2, 3$, and some event σ such that $s_2 = s_3$ and $P(s_1) = P(s_2)$, $s_1\sigma \in \overline{K}$, $s_3\sigma \in L(G)$, while $s_2\sigma \notin \overline{K}$, i.e., the observability condition is violated.

We prove that the above algorithm may be realized with the help of ATP in the PCF calculus. For this, some preliminary procedures are required, in particular, we determine what transitions are defined in each state of automata involved, namely G and H, using logical inference only. The following list of predicates will be exploited in this procedure: a predicate $Q^X(\_)$ that corresponds to all states of the automaton X, a predicate $E^X(\_)$ that defines all events of the automaton X, and terms $\delta_X(q_1^i, \sigma^i, q_2^i)$ that determine transitions from the state q_1^i of automaton X to the state q_2^i labeled with an event σ^i. Let the term $F_X(q^i, \sigma^j)$ mean that there is a transition from the state q^i of the automaton X labeled by the event σ^j. Let $NoF_X(q^i, \sigma^j)$ mean the opposite, i.e., that there is no such transition. For an automaton X, consider the PCF $\mathcal{F}_{PrepX}$ (1) with the base $B_{PrepX} = \{Q^X(q^i), \delta_X(q_1^i, \sigma^i, q_2 i), E^X(\sigma^i)\}$:

$$\mathcal{F}_{PrepX} = \exists B_{PrepX} \left\langle \begin{array}{l} \forall \sigma, q\ E^X(\sigma), Q^X(q) \text{ ——————— } \exists\ NoF_X(q, \sigma) \\ \forall \sigma, q, q_1\ \delta_X(q, \sigma, q_1), NoF_X^*(q, \sigma) \text{ —— } \exists\ F_X(q, \sigma) \end{array} \right. \quad (1)$$

During the inference search for $\mathcal{F}_{PrepX}$, the following strategy is used: First, all possible answers to the first question are being searched for, thus atoms $NoF_X(q^i, \sigma^j)$ are added into the base. Then, answers to the second question add proper atoms $F_X(q^i, \sigma^j)$ into the base while removing respective atoms $NoF_X(q^i, \sigma^j)$. The inference of PCF $\mathcal{F}_{PrepX}$ for any finite automaton X ends due to exhaustion of answering substitutions since the sets used for searching for substitutions are finite.

The operator $*$, used for the deletion of the redundant atoms $NoF_X(q^i, \sigma^j)$ added by the first question, demonstrates the essential feature of the calculus of PCFs, namely, the possibility of nonmonotonic inference. In particular, after applying the inference rule ω, the atoms that participated in the matching search with the atoms marked with * in question should be removed from the base. In general, the operator * affects the property of completeness of the PCF calculus, but for the problem considered in this paper, the inference using * is always complete.

Denote by B'_{PrepX} the base obtained as a result of the inference search of $\mathcal{F}_{PrepX}$. Let $B_{Obs} = B'_{PrepG} \cup B'_{PrepH} \cup \{Q_0^G(q_0^G), Q_0^H(q_0^H), E_c(\sigma^j), E_o(\sigma^j), E_{uo}(\sigma^j)\}$, where $Q_0^X(\_)$ determines the initial state of automaton X, atoms $E_c(\sigma^i)$, $E_o(\sigma^j)$, and $E_{uo}(\sigma^k)$ define controllable and observable events. The predicate $T(\_,\_,\_)$ will be used to construct states of the automaton T_{Obs}, while the predicate $\delta_T(\_)$ will construct transitions of T_{Obs}, and $\delta_T(q_H^1, q_H^2, q_G, \sigma_1, \sigma_2, \sigma_3, t_H^1, t_H^2, t_G)$ is equal to the phrase "there is a transition labeled $(\sigma_1,\sigma_2,\sigma_3)$ from the state (q_H^1, q_H^2, q_G) to the state (t_H^1, t_H^2, t_G) of the automaton T_{Obs}".

To test observability, we employ a PCF $\mathcal{F}_{Obs}$ (2) that constructs the testing automaton T_{Obs}. The questions of $\mathcal{F}_{Obs}$ are listed as formulas $R_1 - R_6$ below.

$$\mathcal{F}_{Obs} = \exists B_{Obs} \left\langle \begin{array}{l} R_1 \\ R_2 \\ \dots \\ R_6 \end{array} \right. \quad (2)$$

$$R_1 : \forall q_G, q_H^1\ Q_0^G(q_G), Q_0^H(q_H^1) - \exists\ Q_0^T(q_H^1, q_H^1, q_G), T(q_H^1, q_H^1, q_G)$$
$$R_2 : \forall \sigma, q_H^1, q_H^2, q_G, t_H^1, t_H^2, t_G\ T(q_H^1, q_H^2, q_G), E_o(\sigma),$$
$$\delta_H(q_H^1, \sigma, t_H^1), \delta_H(q_H^2, \sigma, t_H^2), \delta_G(q_G, \sigma, t_G) -$$
$$\exists\ T(t_H^1, t_H^2, t_G), \delta_T(q_H^1, q_H^2, q_G, \sigma, \sigma, \sigma, t_H^1, t_H^2, t_G)$$
$$R_3 : \forall \sigma, q_H^1, q_H^2, q_G, t_H^1\ T(q_H^1, q_H^2, q_G), E_{uo}(\sigma), \delta_H(q_H^1, \sigma, t_H^1) -$$
$$\exists\ T(t_H^1, q_H^2, q_G), \delta_T(q_H^1, q_H^2, q_G, \sigma, \varepsilon, \varepsilon, t_H^1, q_H^2, q_G)$$
$$R_4 : \forall \sigma, q_H^1, q_H^2, q_G, t_H^2, t_G\ T(q_H^1, q_H^2, q_G), E_{uo}(\sigma), \delta_H(q_H^2, \sigma, t_H^2), \delta_G(q_G, \sigma, t_G) -$$
$$\exists\ T(q_H^1, t_H^2, t_G), \delta_T(q_H^1, q_H^2, q_G, \varepsilon, \sigma, \sigma, q_H^1, t_H^2, t_G)$$
$$R_5 :\ \forall \sigma, q_H^1, q_H^2, q_G\ T(q_H^1, q_H^2, q_G), E_c(\sigma), F_H(q_H^1, \sigma), NoF_H(q_H^2, \sigma), F_G(q_G, \sigma) -$$
$$\exists\ dead(q_H^1, q_H^2, q_G, \sigma), \delta_T(q_H^1, q_H^2, q_G, \sigma, \varepsilon, \sigma, q_H^1, t_H^2, q_G)$$
$$R_6 : \forall q_H^1, q_H^2, q_G, \sigma\ dead(q_H^1, q_H^2, q_G, \sigma)$$

Proposition 1. *Given a partially observable DES G and a regular language K recognized by a finite-state automaton H, let in the PCF* $\mathcal{F}_{Obs}$ *(2)* $B_{Obs} = B'_{PrepG} \cup B'_{PrepH} \cup \{Q_0^G(q_0^G), Q_0^H(q_0^H), E_c(\sigma^j), E_o(\sigma^j), E_{uo}(\sigma^j)\}$*. Then, the inference of* $\mathcal{F}_{Obs}$ *always terminates, and the language K is unobservable if and only if the resulting base* B'_{Obs} *contains an atom* $dead(q_H^1, q_H^2, q_G, \sigma)$*.*

Proof. To prove the proposition, we consider each question of the PCF $\mathcal{F}_{Obs}$ to show that violation of observability leads to the appearance of the atom $dead(q_H^1, q_H^2, q_G, \sigma)$ in the base of $\mathcal{F}_{Obs}$. The question R_1 adds into the base an atom $T(q_H^1, q_H^1, q_G)$ that serves as a starting point for the inference since no question except the last one may be answered without finding in the base a proper substitution for the term $T(\_, \_, \_)$.

Questions $R_2 - R_4$ are aimed at constructing states and transitions of the automaton T_{obs}, wherein R_2 is responsible for processing observable events, while R_3 and R_4 are responsible for processing unobservable events occurrences. T_{obs} is constructed in such a way as to ensure $s_2 = s_3$ and $P(s_1) = P(s_2)$, thus implementing the main idea of the rules for constructing T_{obs} from [1]. Each state of the desired automaton T_{Obs} has the form (h_1, h_2, q), where the first two components of the triple are some states of automaton H, and q is the state of automaton G, i.e., $h_1 \in Q_H$, $h_2 \in Q_H$, $q \in Q_G$. Thus, we consider a set of strings $s_i \in \overline{K}$, $i = 1, 2, 3$ corresponding to transitions between these states. An observable event σ allows answering to the question R_2 that results in adding into the base B_{obs} a transition of T_{obs} labeled by a triple (σ, σ, σ). If event σ is unobservable, then two transitions are constructed: The first one is determined by the question R_4, labeled by a triple $(\varepsilon, \sigma, \sigma)$ to ensure $s_2 = s_3, s_2\sigma \in \overline{K}, s_3\sigma \in L(G)$. The second transition is constructed by the question R_3, which is labeled by a triple $(\sigma, \varepsilon, \varepsilon)$ to ensure $P(s_1) = P(s_2), s_1\sigma, s_2\sigma \in \overline{K}$.

The question R_5 checks if a controllable event e violates the observability condition at the current state (X_1, X_2, Q) of T_{obs} achieved by a set of strings $s_i \in \overline{K}$, $i = 1, 2, 3$. Indeed, let there be a substitution $\{\sigma \to e, q_H^1 \to X_1, q_H^2 \to X_2, q_G \to Q\}$ such that atoms $T(q_H^1, q_H^2, q_G)$, $E_c(\sigma)$, $F_H(q_H^1, \sigma)$, $NoF_H(q_H^2, \sigma)$, $F_G(q_G, \sigma)$ become *True* simultaneously. This means that for $s_i \in \overline{K}$, $i = 1, 2, 3$, and for the event e, we have $s_1 e \in \overline{K}$, $s_3 e \in L(G)$, while $s_2 e \notin \overline{K}$. By construction of T_{obs}, $s_2 = s_3$, and $P(s_1) = P(s_2)$, and such a combination means a violation of observability. Given such a substitution, the answer to the question R_5 adds the atom $dead(q_H^1, q_H^2, q_G, \sigma)$ into the base. It contains the information about the state and event where the observation conditions are violated. The question R_5 cannot be answered if observability is not violated by some controllable event e.

The question R_6 is the goal question an answer to which terminates the inference search. It may be answered only if the base contains the atom $dead(q_H^1, q_H^2, q_G, \sigma)$; thus, if it has been answered, then the language K is unobservable.

The inference of the PCF $\mathcal{F}_{Obs}$ is always finite since no functional symbols are used (only calculated ones), and all sets used for searching for substitutions are finite. The

inference always ends with either a refutation of the base by answering the goal question R_6, i.e., adding the atom *dead* into the base, or an exhaustion of options for searching of answering substitutions. □

Example 1. *Consider a DES presented by the generator G in Figure 1 and specification language K generated by the automaton H in Figure 2. Let* $\Sigma_{uo} = \{u, v, c, e\}$, $\Sigma_c = \Sigma$. *It may be noted that the strings* $s = uv$ *and* $t = \varepsilon$ *are those that cause the conflict in the system G. Indeed, the occurrence of event a leads to the situation when* $P(s) = P(t)$, $sa \in \overline{K}$ *but* $ta \notin \overline{K}$, *which violates the observability condition.*

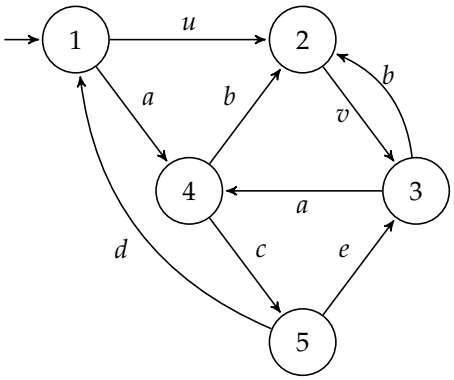

Figure 1. Automaton *G*.

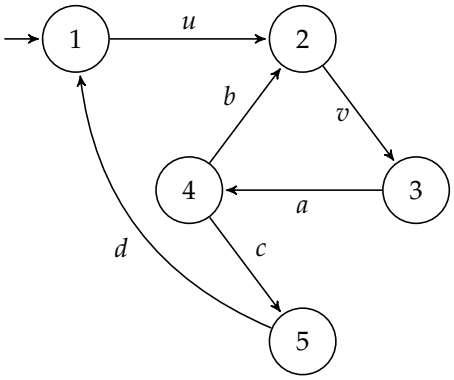

Figure 2. Automaton *H* as a recognizer of $\bar{K}$.

Table 1 illustrates the process of adding new atoms to the base that define the states and transitions of the automaton T_{Obs} *serving for observability check. As one can see, the minimum inference for PCF* $\mathcal{F}_{Obs}$ *with the base corresponding to the automata in Figures 1 and 2 consists of four steps. Since inference is random in nature, its length may vary. The choice of questions and answers occurs according to the chosen strategy (see Section 6), but whether the strategy will lead to the contradiction in a short or long way is unknown a priori. Table 1 shows the inference constructed by the prover Bootfrost, consisting of eight steps. The entire automaton* T_{Obs} *and all possible conflicts can be constructed by removing the goal question* R_6 *from PCF* $\mathcal{F}_{Obs}$. *Figure 3 shows a part of automaton* T_{Obs} *constructed by the inference presented in Table 1.*

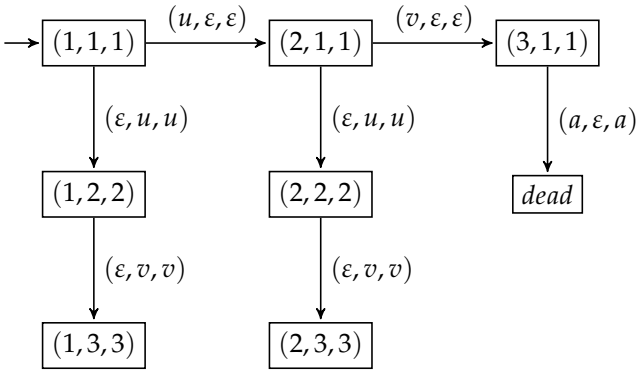

Figure 3. A part of automaton T_{Obs} for observability checking.

Table 1. PCF $\mathcal{F}_{Obs}$ refutation progress for checking observability using automaton T_{Obs}.

$R\#$	σ	q_H^1	q_H^2	q_G	t_H^1	t_H^2	t_G	Additions into the Base
1		1		1				$T(1,1,1)$
3	u	1	1	1	2			$T(2,1,1)$,$\delta^T(1,1,1,u,\varepsilon,\varepsilon,2,1,1)$
4	u	1	1	1		2	2	$T(1,2,2)$,$\delta^T(1,1,1,\varepsilon,u,u,1,2,2)$
4	u	2	1	2		2	2	$T(2,2,2)$,$\delta^T(2,1,1,\varepsilon,u,u,2,2,2)$
4	v	1	2	2		3	3	$T(1,3,3)$,$\delta^T(1,2,2,\varepsilon,v,v,1,3,3)$
4	v	2	2	2		3	3	$T(2,3,3)$,$\delta^T(2,2,2,\varepsilon,v,v,2,3,3)$
3	v	2	1	1	3			$T(3,1,1)$,$\delta^T(2,1,1,v,\varepsilon,\varepsilon,3,1,1)$
5	a	3	1	1				$dead(3,1,1,a)$
6	a	3	1	1				$\varnothing$

4. Conflict Extracting

To examine partially observable systems, observer automata are often employed. There are standard methods of constructing such automata, so we direct readers to refer to the literature, e.g., [1]. One of the methods consists in changing all unobservable events with the silent τ event and converting the resulting nondeterministic automaton to a deterministic one.

Example 2. *In Example 1, the observer in Figure 4 shows that the event a in the state* $\{1,2,3\}$ *causes the conflict. Indeed, to satisfy specification H in Figure 2, the event a must be disabled at state* 1 *but must be enabled at state* 3.

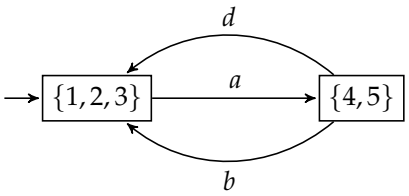

Figure 4. Observer automaton for H with $\Sigma_{uo} = \{u, v, c, e\}$.

If the specification language proved to be unobservable using PCF $\mathcal{F}_{Obs}$, conflicting strings may be extracted from the automaton T_{obs} with the help of another PCF, and we name it $\mathcal{F}_{Conf}$ (Equation (3)). The PCF $\mathcal{F}_{Conf}$ processes the final version B'_{Obs} of the base B_{Obs} obtained during the observability checking so set $B_{Conf} = B'_{Obs}$.

$$\mathcal{F}_{Conf} = \exists B_{Conf} \begin{cases} R_1^{Conf} \\ R_2^{Conf} \end{cases} \tag{3}$$

$$R_1^{Conf} : \forall \sigma, q_H^1, q_H^2, q_G, q_H^1 f, q_H^2 f, q_G f, s_1, s_2, s_3 \; \delta_T(q_H^1 f, q_H^2 f, q_G f, s_1, s_2, s_3, q_H^1, q_H^2, q_G), \\ dead(q_H^1, q_H^2, q_G, \sigma) - \exists \; next(q_H^1 f, q_H^2 f, q_G f, s_1, s_2, s_3, \sigma)$$

$$R_2^{Conf} : \forall \sigma, q_H^1, q_H^2, q_G, q_H^1 f, q_H^2 f, q_G f, s_1, s_2, s_3, \sigma_1, \sigma_2, \sigma_3 \; next(q_H^1, q_H^2, q_G, s_1, s_2, s_3, \sigma), \\ \delta_T(q_H^1 f, q_H^2 f, q_G f, \sigma_1, \sigma_2, \sigma_3, q_H^1, q_H^2, q_G) - \\ \exists \; next(q_H^1 f, q_H^2 f, q_G f, \sigma_1 \cdot s_1, \sigma_2 \cdot s_2, \sigma_3 \cdot s_3, \sigma)$$

Proposition 2. *Given a partially observable DES G and a regular language K recognized by a finite-state automaton H, let in the PCF $\mathcal{F}_{Conf}$ (3) $B_{Conf} = B'_{Obs}$, where B'_{Obs} is the final version of the base of the PCF $\mathcal{F}_{Obs}$ after the end of its inference. Then, the inference of $\mathcal{F}_{Conf}$ always terminates, and the last added atom $next(q_0^H, q_0^H, q_0^G, s_1^{conf}, s_2^{conf}, s_2^{conf}, e)$ contains conflicting strings as its forth and fifth arguments.*

Proof. The question R_1^{Conf} in the PCF $\mathcal{F}_{Conf}$ is the only question that can be answered after the start of the inference. It uses the argument values of $dead(q_H^1, q_H^2, q_G, \sigma)$ and $\delta_T(\ldots, s_1, s_2, s_3, q_H^1, q_H^2, q_G)$ to obtain the last symbols of the required conflicting strings s_1^{conf} and s_2^{conf}. As stated in the observability checking algorithm, $s_2 = s_3$ in all terms $\delta_T(\ldots, s_1, s_2, s_3, q_H^1, q_H^2, q_G)$.

Sequential answers to the second question R_2^{Conf} occur according to transitions of the automaton T_{obs} contained in the base B_{Conf} as atoms $\delta_T(q_H^1 f, q_H^2 f, q_G f, \sigma_1, \sigma_2, \sigma_3, q_H^1, q_H^2, q_G)$. Having started in the state $T(q_H^1, q_H^2, q_G)$ of T_{obs}, the inference ends in the state Q_0^T. During the inference, the atoms $next(q_H^1 f, q_H^2 f, q_G f, \sigma_1 \cdot s_1, \sigma_2 \cdot s_2, \sigma_3 \cdot s_3, \sigma)$ are being added into the base where, by construction of T_{obs}, s_1 and s_2 accumulate suffixes of conflicting words. When the initial state Q_0^T of the automaton T_{obs} will be achieved with some substitution $\{\sigma \to e, q_H^1 \to X_1, q_H^2 \to X_2, q_G \to Q, q_H^1 f \to q_0^H, q_H^2 f \to q_0^H, q_G f \to q_0^G, s_1 \to \hat{s}_1, s_2 \to \hat{s}_2, s_3 \to \hat{s}_3, \sigma_1 \to \hat{\sigma}_1, \sigma_2 \to \hat{\sigma}_2, \sigma_3 \to \hat{\sigma}_3\}$, then $\hat{s}_1 = s_1^{conf}$ and $\hat{s}_2 = s_2^{conf}$, where s_1^{conf} and s_2^{conf} are the strings causing the conflict in case of occurrence of the event e. □

Example 3. *For the conflict in Example 1, the inference of $\mathcal{F}_{Conf}$ consists of two steps. When the inference terminates the arguments of the atom $next(\_)$ in the base store the information required, in particular, the string $s_1 = uv$, the string $s_2 = \varepsilon\varepsilon = \varepsilon$, and the conflict event a. Table 2 shows the inference of $\mathcal{F}_{Conf}$ in Example 1.*

Table 2. The inference of the PCF $\mathcal{F}_{Conf}$.

σ	q_H^1	q_H^2	q_G	q_H^{1f}	q_H^{2f}	q_G^f	s_1	s_2	s_3	σ_1	σ_2	σ_3	**Additions**
a	3	1	1	2	1	1	v	ε	ε				$next(2,1,1,v,\varepsilon,\varepsilon,a)$
a	2	1	1	1	1	1	v	ε	ε	u	ε	ε	$next(1,1,1,u \cdot v, \varepsilon \cdot \varepsilon, \varepsilon \cdot \varepsilon, a)$

5. Controlled System Design

Let $\Gamma : Q \to 2^{\Sigma}$ be the *active event function* (also known as feasible event function); $\Gamma(q)$ is the set of all events e for which $\delta(q, e)$ is defined. $\Gamma(q)$ is also called the *active event set* (or feasible event set) of G at q [1]. In the case of systems with complete observations, the supervisor can be viewed as the recognizer H of $\overline{K}$, i.e., an automaton that marks $\overline{K}$ or as the recognizer H' of its supremal controllable sublanguage $\overline{K'}$ of the specification K. Control actions for the string s generated by the plant are directly obtained from the

active event set $\Gamma(\delta_J(q_0^J, s))]$ of the supervisor automaton after the string s has been read. A supervisor J thus ensures $J(s) = [\Sigma_{uc} \cap \Gamma(\delta(q_0, s))] \cup \{\sigma \in \Sigma_c : s\sigma \in \overline{K}\}$. It is known that a closed-looped behavior of the plant and the supervisor may be realized by the parallel composition of the corresponding automata, i.e., $L(J/G) = L(H||G)$.

Now, our goal is to implement supervisory control in the case of partial observation of events in G to solve BSCOP. For this, in the case of observable and controllable specification, one often employs an observer automata. Let H_{Obs} be an observer for automaton H. Let s be a current string of DES G and let $t = P(s)$ be a current observed string available. Let x_{obs} be the current state of H_{Obs} after the execution of t. Since observer H_{Obs} states in general are sets of states of H, this means that after the last event in t automaton H could be in any one of the states in the set x_{obs}. Then, we have that $J_P(t) = \bigcup_{x \in x_{obs}} [\Gamma_{H_{Obs}}(x)]$, where $\Gamma_{H_{Obs}}$ is the active event function of H_{Obs}. As stated in [1], in order to use the same technique that was used for the case of complete observation of events, it is enough for each state x_{obs} to add self-loops for the unobservable events that are feasible in corresponding $x \in x_{obs}$ in H. The resulting automaton H'_{Obs} guarantees the desired result $L(J_P/G) = L(H'_{Obs}||G)$ for a supervisor J_P.

Let $B_{loops} = B_H \cup B_{H_{Obs}} \cup \{unObs(\bar{e})\}$. The predicate $unObs(\_)$, having a list of elements as an argument, is used to define all unobservable events of H. To obtain an element of a list $\bar{e}$, the computable predicate $\in$ is used. Details on the realization of lists in PCF prover Bootfrost are presented in Section 6.3. During the inference search of the PCF $\mathcal{F}_{loops}$ (4), new transition atoms $\delta^{Obs}(\bar{q}, \sigma, \bar{q})$ corresponding to auxiliary loops for unobservable events σ feasible in states x of H corresponding to x_{obs} of H_{Obs} are added to the base B_{loops}.

$$\mathcal{F}_{loops} = \exists\, B_{loops} \leftarrow \begin{matrix} \forall q, \sigma, q', \sigma' \bar{q}, \bar{q}', \bar{\sigma}\ \delta^H(q, \sigma, q'), \delta^{H_{Obs}}(\bar{q}, \sigma', \bar{q}'), \\ q \in \bar{q}, \sigma \in \bar{\sigma}, unObs(\bar{\sigma}) \end{matrix} - \exists\, \delta^{Obs}(\bar{q}, \sigma, \bar{q}) \tag{4}$$

In PCF formalization, the behavior of the system under the supervisory control may be described by the PCF $\mathcal{F}_{BSCOP}$ (5). The predicate $L^{J_P/G}(\_,\_)$ is employed to store words of the controlled language $L^{J_P/G}$ as its first argument. A state to which this string brought the system is stored as the second argument. Although the language $L(J_P/G)$ is a restricted version of $L(G)$, we use a new predicate $L^{J_P/G}(\_,\_)$ to emphasize that the language constructed is a result of teamwork of the plant and the supervisor. The base B_{BSCOP} consists of sets of atoms corresponding to the transitions of the plant and the supervisor, correspondingly, and contains initial atom $L^{J_P/G}(\varepsilon, 1)$.

$$\mathcal{F}_{BSCOP} = \exists\, B_{BSCOP} \leftarrow \begin{matrix} \forall \sigma, s, q, q', \bar{q}_J, \bar{q}'_J\ L^{J_P/G}(s, q), \delta^G(q, \sigma, q'), \\ \delta^{H'_{Obs}}(\bar{q}_J, \sigma, \bar{q}'_J), q \in \bar{q}_J \end{matrix} - \exists\, L^{J_P/G}(s \cdot \sigma, q') \tag{5}$$

The single question of $\mathcal{F}_{BSCOP}$ may be interpreted as follows. If the system is at the state q and an event σ occurs, then according to the δ^G, the system is switched to the specified state q', and σ is added to the current string of events s stored as the first argument of the predicate $L^{J_P/G}(\_,\_)$. The rule works only on those strings that are allowed by the supervisor, i.e., atoms $\delta^{H'_{Obs}}(\_,\_,\_)$ limit the answers that could be found with atoms $\delta^G(\_,\_,\_)$.

Example 4. *In Example 1, let $\Sigma_{uo} = \{u, c, e\}$, $\Sigma_c = \Sigma$, i.e., the event u is now considered observable. The corresponding observer is presented in Figure 5. It may be seen that no event must be disabled and enabled simultaneously. In this case, the logical inference search of the PCF $\mathcal{F}_{Obs}$ performed by Bootfrost terminates due to exhaustion of all possible substitutions. When the inference search is stopped, the predicates in the base of $\mathcal{F}_{Obs}$ define the states and transitions of the automaton T_{Obs}, but none of them are the dead$(_)$ atom.*

Table 3 shows a few steps of the inference of PCF $\mathcal{F}_{BSCOP}$ in Example 1, with the observer in Figure 5 modified with the PCF $\mathcal{F}_{loops}$ (4).

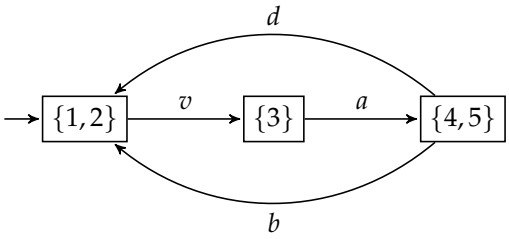

Figure 5. Observer automaton for H with $\Sigma_{uo} = \{u, c, e\}$.

Table 3. The first steps of the PCF $\mathcal{F}_{BSCOP}$ inference constructing $L(J_P/G)$.

Base Atoms Used	s	q	σ	q'	$\bar{q}_J$	$\bar{q}'_J$	Additions
$L^{J_P/G}(\varepsilon,1),\delta^G(1,u,2),\delta^{H'_{Obs}}([1,2],u,[1,2])$	ε	1	u	2	$[1,2]$	$[1,2]$	$L^{J_P/G}(\varepsilon\cdot u,2)$
$L^{J_P/G}(\varepsilon\cdot u,2),\delta^G(2,v,3),\delta^{H'_{Obs}}([1,2],v,[3])$	$\varepsilon\cdot u$	2	v	3	$[1,2]$	$[3]$	$L^{J_P/G}(\varepsilon\cdot u\cdot v,3)$
$L^{J_P/G}(\varepsilon\cdot u\cdot v,3),\delta^G(3,v,4),\delta^{H'_{Obs}}([3],a,[4,5])$	$\varepsilon\cdot u\cdot v$	3	a	4	$[3]$	$[4,5]$	$L^{J_P/G}(\varepsilon\cdot u\cdot v\cdot a,4)$
$L^{J_P/G}(\varepsilon\cdot u\cdot v,3),\delta^G(4,v,5),\delta^{H'_{Obs}}([4,5],c,[4,5])$	$\varepsilon\cdot u\cdot v\cdot a$	4	c	5	$[4,5]$	$[4,5]$	$L^{J_P/G}(\varepsilon\cdot u\cdot v\cdot a\cdot c,5)$
$L^{J_P/G}(\varepsilon\cdot u\cdot v,3),\delta^G(4,v,2),\delta^{H'_{Obs}}([4,5],b,[1,2])$	$\varepsilon\cdot u\cdot v\cdot a$	4	b	2	$[4,5]$	$[1,2]$	$L^{J_P/G}(\varepsilon\cdot u\cdot v\cdot a\cdot b,2)$

The representation of DESs with the help of PCFs allows one to use information coming from the environment, as well as data on the functioning of the system itself in the process of constructing the logical inference. This can be realized by special logical rules represented in PCF in the form of event processing questions. Answering them triggers subinferences, in which the events coming from the environment serve as parameters used in calculations or other decisions. To implement this inference search behavior, formulas use the # label to indicate that the labeled term is a call to an external function.

Here, we utilize # in PCF $\mathcal{F}^{\#}_{BSCOP}$ (6) that presents a version of $\mathcal{F}_{BSCOP}$ obtaining an event from outside. Note the atom $E(\sigma^{\#})$ in the base of $\mathcal{F}^{\#}_{BSCOP}$ where $\sigma^{\#}$ is the result of calling the function *get_random_event*(), which provides a random event that can occur in the current state.

$$\mathcal{F}^{\#}_{BSCOP} = \begin{array}{l}\exists\ B_{BSCOP}\cup \\ \{E(\sigma^{\#})\}\end{array} \_ \begin{array}{r}\forall\sigma,s,q,q',\bar{q}_J,\bar{q}'_J\ L^{J_P/G}(s,q),\delta^G(q,\sigma,q'), \\ \delta^{H'_{Obs}}(\bar{q}_J,\sigma,\bar{q}'_J), q\in\bar{q}_J, E^*(\sigma)\end{array} \_ \begin{array}{r}\exists\ L^{J_P/G}(s\cdot\sigma,q'), \\ E(\sigma^{\#})\end{array} \quad (6)$$

Example 5. *Table 4 shows the first steps of the inference of PCF $\mathcal{F}^{\#}_{BSCOP}$ in Example 4. Note the difference in the last lines of Tables 3 and 4. Unlike $\mathcal{F}_{BSCOP}$, PCF $\mathcal{F}^{\#}_{BSCOP}$ does not generate words of the same length, thus simulating the real system behavior.*

Table 4. The first steps of the PCF $\mathcal{F}^{\#}_{BSCOP}$ inference constructing $L^{\#}(J_P/G)$.

Base Atoms Used	s	q	σ	q'	$\bar{q}_J$	$\bar{q}'_J$	Additions
$L^{J_P/G}(\varepsilon,1),\delta^G(1,u,2),\delta^{H'_{Obs}}([1,2],u,[1,2]),E(u)$	ε	1	u	2	$[1,2]$	$[1,2]$	$L^{J_P/G}(\varepsilon\cdot u,2),E(\sigma^{\#})$
$L^{J_P/G}(\varepsilon\cdot u,2),\delta^G(2,v,3),\delta^{H'_{Obs}}([1,2],v,[3]),E(v)$	$\varepsilon\cdot u$	2	v	3	$[1,2]$	$[3]$	$L^{J_P/G}(\varepsilon\cdot u\cdot v,3),E(\sigma^{\#})$
$L^{J_P/G}(\varepsilon\cdot u\cdot v,3),\delta^G(3,u,4),\delta^{H'_{Obs}}([3],a,[4,5]),E(a)$	$\varepsilon\cdot u\cdot v$	3	a	4	$[3]$	$[4,5]$	$L^{J_P/G}(\varepsilon\cdot u\cdot v\cdot a,4),E(\sigma^{\#})$
$L^{J_P/G}(\varepsilon\cdot u\cdot v,3),\delta^G(4,u,5),\delta^{H'_{Obs}}([4,5],c,[4,5]),E(c)$	$\varepsilon\cdot u\cdot v\cdot a$	4	c	5	$[4,5]$	$[4,5]$	$L^{J_P/G}(\varepsilon\cdot u\cdot v\cdot a\cdot c,5),E(\sigma^{\#})$
$L^{J_P/G}(\varepsilon\cdot u\cdot v,3),\delta^G(5,u,1),\delta^{H'_{Obs}}([4,5],b,[1,2]),E(d)$	$\varepsilon\cdot u\cdot v\cdot a\cdot c$	5	d	1	$[4,5]$	$[1,2]$	$L^{J_P/G}(\varepsilon\cdot u\cdot v\cdot a\cdot c\cdot d,1),E(\sigma^{\#})$

The PCF $\mathcal{F}^{T\#}_{BSCOP}$ (7) presents an extended version of PCF $\mathcal{F}^{\#}_{BSCOP}$ that allows to not only process events coming from outside but also take into account the knowledge available in the system. The usage of $\mathcal{F}^{T\#}_{BSCOP}$ is explained in Section 7.

$$\mathcal{F}^{T\#}_{BSCOP} = \begin{array}{c} \exists\, B_{BSCOP} \cup \\ \{T(\bar{\sigma}^{\#})\} \end{array} \left\langle \begin{array}{l} \forall \sigma\ T^{\#}(\bar{\sigma}) \\ \forall \sigma, s, q, q', \bar{q}_J, \bar{q}'_J\ L^{J_P/G}(s,q), E^*(\sigma) \\ \delta^G(q,\sigma,q'), \delta^{H'_{Obs}}(\bar{q}_J,\sigma,\bar{q}'_J), q \in \bar{q}_J \end{array} \right\rangle \begin{array}{l} \\ \\ \end{array} \begin{array}{l} \exists\, E(\sigma) \\ \\ \exists\, L^{J_P/G}(s\cdot\sigma, q') \end{array} \tag{7}$$

6. The Prover for Refutation PCFs

To facilitate inference searches in the PCF calculus, a prover named Bootfrost is developed. The source code of the project, written in the Rust programming language, documentation, and examples, including those presented in this paper, can be found on the GitHub page [24].

A number of modern approaches have been implemented in Bootfrost: perfect sharing of terms that significantly saves memory on some tasks; indexing of the fact base; and automatic, semiautomatic, and manual modes of logical inference searching. Rust's rich type system and the ownership model guarantee memory safety and thread safety. Also, there is no runtime or garbage collector in Rust. These features give the opportunity to realize safe and efficient systems. The prover design is based on a transaction system that allows one to log and roll back any changes that occur during the logical inference. Thus, all algorithms used are taken into account. In addition, this version of the prover is specialized for guarded PCFs. Although this limits the class of problems to be solved, nevertheless, in applied problems and problems describing dynamic systems, only such formulas are usually used. Guarded PCFs are the formulas in typical quantifier conditions in which all variables controlled by the quantifier occur in the conjunct. For example, $\forall x, y A(x), B(y)$ is a subformula with guarded variables because variables x, y occur in the conjunct $A(x), B(y)$, and $\forall x, y A(x), B(x)$ is a subformula with unguarded variables because variable y does not occur in the conjunct $A(x), B(x)$.

The prover was tested using problems from the TPTP library [40] to empirically verify its soundness. About 9000 problems were selected in the First-Order Formula format, and among them, 1001 with guarded variables. None of the problems that had no solution were solved, which is considered as the main criteria of any prover's soundness at the CADE ATP System Competition (https://tptp.org/CASC/, accessed on 25 April 2024).

Strategies employed in Bootfrost are divided into three main groups: question selection strategies, answer selection strategies, and evaluated terms and commands.

6.1. Question Selection Strategies

At each step of the inference, the prover must select a question for which it will be looking for an answer. A special procedure for the question selection is used in Bootfrost. It scores each question by several criteria. Such criteria can be configured by the user, but there is a default strategy that scores a question as follows:

1. Is the question a goal? Goal questions are scored highest because answering them terminates the inference.
2. How many steps ago in the inference was the question answered for the last time? The most distant questions are scored the highest. Such an approach provides a high level of diversity by restricting the usage of the same question several times in a row.
3. How many times has the question been answered before? Questions with the lowest value are scored the highest.
4. What is the level of the question branching? The ones with the lowest value are scored the highest.

This general strategy sorts questions by the score from the highest to the lowest. Then, a special method tries to answer questions from the list using an answer selection strategy

(see the next subsection). If an answer is found, the formula is transformed according to the inference rule ω, and the transition to the next step is performed.

6.2. Answer Selection Strategies

There are two answer selection strategies in the Bootfrost prover, called the First appropriate answer and the Best answer.

The *First appropriate answer* strategy selects the first found answer that satisfies a given criteria. Such an approach leads to effective usage of the memory and CPU resources. The criteria can be configured, and the default criteria is a trivial function returning *true*, which leads to selecting the first answer found.

The *Best answer* strategy selects the best answer from the set of all possible answers. Thus, this strategy finds all possible answers and then selects the best of them according to a special selection function, which can be configured by the user. The default function just selects an answer with the lowest total weight of all terms involved. A term weight is the amount of nodes in a tree representing the term; for example, the term $A(e)$ has weight 2, and the term $A(e, f(e))$ has weight 4.

Another auxiliary procedure used for answer searching is a starting point selection. There are two variants of where to start the search: "from the last" and "from scratch". The "From the last" method means that the next search for an answer to the question will begin at the point where the prover stopped during the previous searching procedure. "From scratch" means that the procedure for finding answers starts from scratch, i.e., without taking into account previous search results. This approach is applicable for nonmonotonic inferences and in similar situations where the previous history of the inference can change over time and become irrelevant. For example, the "from scratch" method is used for constructing the sublanguage $K^{\uparrow C}$ of an uncontrollable specification K [26]. In the PCF $\mathcal{F}_{Sub}$, for providing a sublanguage construction, a special operator * is employed. If a question with an atom marked with the * operator has an answer, then after applying the inference rule the atoms in the base that participated in the matching search with the marked atom should be removed from the base. The operator * can be also modeled by the command "remove-fact", described in the next section.

6.3. Evaluated Terms and Commands

Evaluated terms (*ETerms*) are terms treated by the prover not as syntactical structures but as functions that must be evaluated. For now, Eterms implemented in the prover are arithmetical operations, compare operations, list operations (in, notin, first, last, concatenation, length), and solve. The main framework of term evaluation is as follows: a special procedure retrieves terms by IDs from the environment; then, these terms are performed; and then, for the resulting term, using the perfect sharing structure, the ID is calculated, and this ID is returned as the result.

Evaluated terms that are performed immediately after the inference rule ω application are called *commands*. For example, "remove-fact" is a command. ETerms and commands are classified as strategies because their main purposes are modifying the process of logical inference searching and extending the logical capabilities of PCF. By the commands, it is possible to model the nonmonotonic inference.

Another feature in the task description syntax for the Bootfrost prover is the ability to use lists as arguments of terms. The following computable operations are implemented:

- Evaluated function $\bar{a}$ ++ $\bar{b}$—concatenation of lists $\bar{a}$ and $\bar{b}$;
- Evaluated predicate a \in $\bar{b}$—the membership of element a to list $\bar{b}$;
- Evaluated predicate $\bar{a}$ \subseteq $\bar{b}$—whether $\bar{a}$ is a sublist of $\bar{b}$;
- Evaluated function sort($\bar{a}$)—sorting list items in ascending order;
- Computed function dedup($\bar{a}$)—deletion of repeating elements of the list;

List elements can be other terms and lists, and the above operations handle them only syntactically, without considering the possibility of unification of terms.

6.4. Complexity Evaluation

Let us say some words on the complexity of the algorithms involved. We consider the procedure of construction automaton T_{obs} for checking the observability of the specification language during refutation of PCF $\mathcal{F}_{Obs}$ (Section 3).

As an elementary operation of PCF refutation, we consider the search for a unifying substitution for an atom. This assumption holds because the formula $\mathcal{F}_{Obs}$ under consideration has a fairly simple structure (i.e., belongs to the Horn class) and an algorithm for finding the inference can be provided for it. For example, the algorithm may be chosen in the form of a cycle of traversing questions from top to bottom of the PCF's tree structure and comparing the atoms of the questions with the atoms of the base, while attempting to unify these atoms. For example, for answering question R_4, the terms $T(q_H^1, q_H^2, q_G)$, $E_o(\sigma)$, $\delta_H(q_H^1, \sigma, t_H^1)$, $\delta_H(q_H^2, \sigma, t_H^2)$, $\delta_G(q_G, \sigma, t_G)$ must be unified. Here, variables q_H^1, q_H^2, t_H^1, t_H^2 take values from the set $Q_{\mathcal{H}}$, variables q_G, t_G take values from the set $Q_{\mathcal{G}}$, and variable σ take value from the set Σ_o. So, for complexity, we have the following expression: $|Q_{\mathcal{H}}|^4 \cdot |Q_{\mathcal{G}}|^2 \cdot |\Sigma_o|$.

Let us consider the worst case of possible inference search options. For a given formula, one of the options is the exhaustion of all substitutions, while the answers to the second and third questions do not exist. Another option is that the answer to the second, and consequently, the third target question is carried out after enumerating all triplets of states in the last steps of the inference. Thus, we do not take into account the complexity of answering the second and third questions because they participate in the inference only once, as well as the first question. The computational complexity of answering question R_5 may be evaluated as $|Q_{\mathcal{H}}|^3 \cdot |Q_{\mathcal{G}}| \cdot |\Sigma_{uo}|$ and of answering question R_6 is $|Q_{\mathcal{H}}|^3 \cdot |Q_{\mathcal{G}}|^2 \cdot |\Sigma_{uo}|$. Summarizing complexity for R_4, R_5, R_6, we have $|Q_{\mathcal{H}}|^4 \cdot |Q_{\mathcal{G}}|^2 \cdot |\Sigma_o| + |Q_{\mathcal{H}}|^3 \cdot |Q_{\mathcal{G}}| \cdot |\Sigma_{uo}| + |Q_{\mathcal{H}}|^3 \cdot |Q_{\mathcal{G}}|^2 \cdot |\Sigma_{uo}|$. Reducing, we obtain $|Q_{\mathcal{H}}|^3 \cdot |Q_{\mathcal{G}}| \cdot (|Q_{\mathcal{H}}| \cdot |Q_{\mathcal{G}}| \cdot |\Sigma_o| + |\Sigma_{uo}| + |\Sigma_{uo}| \cdot |Q_{\mathcal{G}}|)$, i.e., if we go to big O notation, the complexity of observability checking is $O(|Q_{\mathcal{H}}|^4 \cdot |Q_{\mathcal{G}}|^2 \cdot |\Sigma|)$.

7. Case Study

As an example, we consider the problem of robot's path planning in an unknown environment. We use the model from [41,42] simplified in the part of the path-following controller and extended by additional states and events, including unobservable ones.

7.1. Problem Statement

Suppose that the robot should follow a given reference path, leaving it to avoid collisions with encountered obstacles and returning to it after completing avoidance maneuvers. Let the robot's dynamics in the horizontal plane be described [43] by equations

$$\begin{cases} \dot{x} = u\cos(\psi_B) - v\sin(\psi_B), \\ \dot{y} = u\sin(\psi_B) + v\cos(\psi_B), \\ \dot{\psi}_B = \omega, \end{cases} \tag{8}$$

where (x, y) are the coordinates of the robot in a global reference frame $\{U\}$, ψ_B is the yaw angle; u and v are the surge and sway speeds, and ω is the yaw rate (Figure 6).

Assuming that u is always nonzero, we define side-slip angle $\beta = arctan(v/u)$ and a reference frame $\{W\}$, which is obtained by rotating $\{B\}$ around the yaw axis through angle β. Subsequently, the kinematic Equation (8) can be rewritten as

$$\begin{cases} \dot{x} = v_t\cos(\psi_W), \\ \dot{y} = v_t\sin(\psi_W), \\ \dot{\psi}_W = \omega + \dot{\beta}, \end{cases} \tag{9}$$

where $\psi_W = \psi_B + \beta$, $v_t = (u^2 + v^2)^{\frac{1}{2}}$ is the absolute value of the total velocity vector $[u \quad v]^T$. The goal is to develop a path-planning solution that meets kinematic constraints

given by minimum turning radius $R_{\min}$ and lies no closer than a safe distance D_s from obstacles.

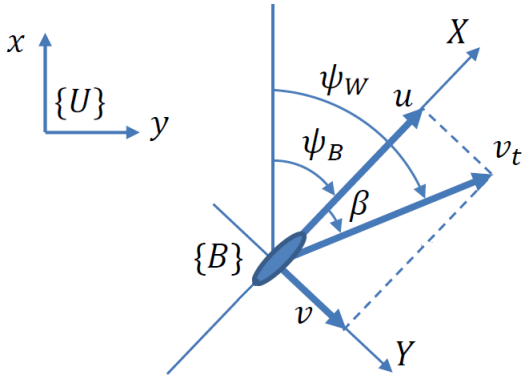

Figure 6. Reference frames (borrowed from [41]).

We assume that the robot is equipped with a multibeam forward-looking sonar (FLS) installed onboard in the horizontal plane in order to detect obstacles in the forward direction. The data obtained from the FLS can be presented as a set of pairs (α_i, ρ_i), where α_i is the beam angle counted from the robot's heading direction $(-\pi/2 < \underline{\alpha} \le \alpha_i \le \overline{\alpha} < \pi/2)$, ρ_i is the distance to an obstacle in the beam direction, $i = \overline{1, N_b}$, N_b is the number of beams. If no obstacles are detected in the direction of beam i, or ρ_i is greater than the detection range ρ_d, then $\rho_i = \infty$.

Let $I_L \triangleq \{i : \alpha_i \le 0\}$ and $I_R \triangleq \{i : \alpha_i > 0\}$ be the sets of left and right beams of the FLS, respectively. For each obstacle point Q_i, $i = \overline{1, N_b}$, we define the maximum turning radius [41,44] as

$$R^i_{\max} = \rho_i \frac{\cos \alpha_i}{\sin 2\beta_i} - D_s, \quad \beta_i = \arctan\left(\frac{D_s}{\rho_i} \sec \alpha_i + \tan \alpha_i\right).$$

It can be used to evaluate the robot's ability to bypass the point Q_i at a safe distance D_s (Figure 7), taking into account the kinematic constraint $R_{\min}$: if $R_{\min} \le R^i_{\max}$, the robot can safely bypass the obstacle point Q_i. Define also $R^L_{\max} = \min_{i \in I_L} R^i_{\max}$, $R^R_{\max} = \min_{i \in I_R} R^i_{\max}$, and $\rho_{\min} = \min_{i \in I_L \cup I_L} \rho_i$.

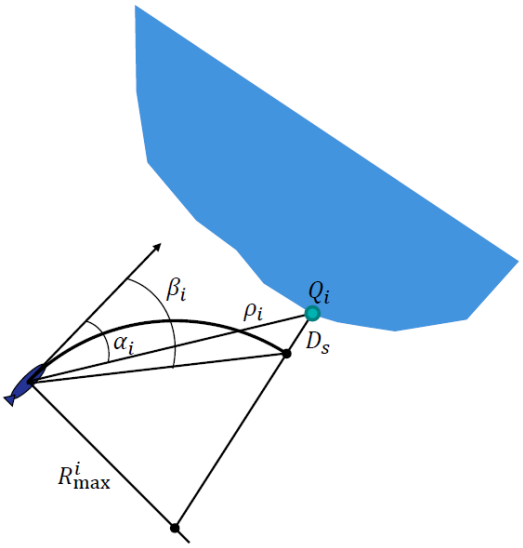

Figure 7. The maximum turning radius R^i_{max} (borrowed from [41]).

The main component of the proposed approach is DES G_{msn} (Figure 8), which is designed to detect situations that require updating the current path. Since, unlike [41], we do not employ a path-following controller to drive the robot along the generated paths, a situation may happen in which the robot finds itself in the wrong position or faced with some obstacle. State *WPD* corresponds to such a situation, and an unobservable event eER denotes transitions to this state. State *RPF* is the initial state. The description of all possible system states is the following:

- *RPF*—reference path following;
- *DOL*—detouring the detected obstacle from its left sid;
- *DOR*—detouring the detected obstacle from its right side;
- *NRP*—navigation to the reference path;
- *SOL*—searching for an obstacle on the left;
- *SOR*—searching for an obstacle on the right;
- *WPD*—wrong position diagnosing;
- *CNP*—computing a new path.

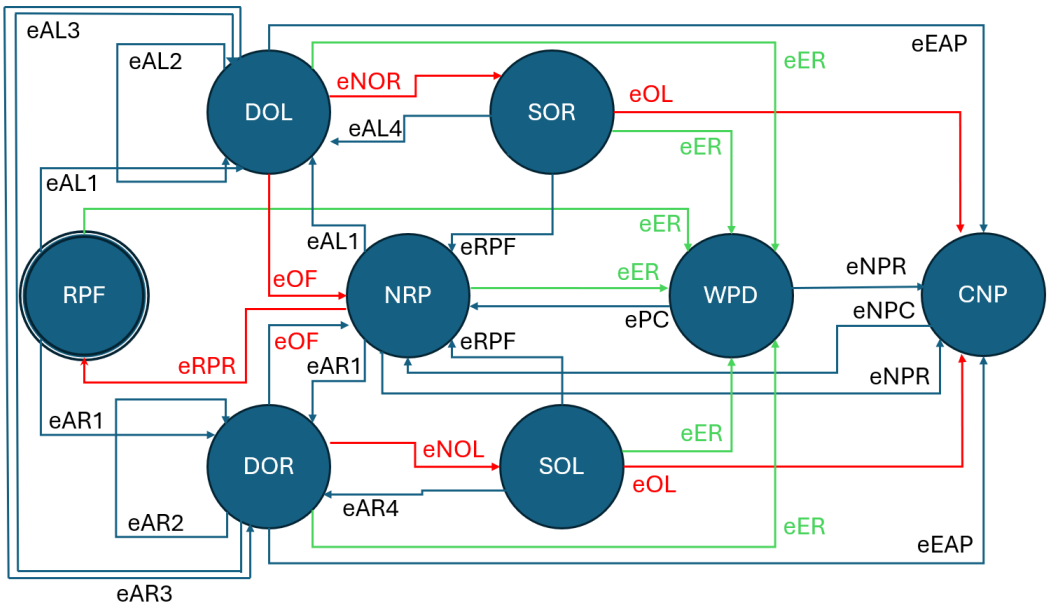

Figure 8. DES G_{msn} for the path-following mission.

The set of system events with their triggering conditions is shown in Table 5. Events ALi, ARi denote switching to the obstacle avoidance modes from the other modes, $i = \overline{1,4}$. They are composite, i.e., determined by several atomic events. It is convenient to define composite events by logical formulas, such as those presented in Table 5. For example, event eAL1 determines switching to the mode of obstacle avoidance from the obstacle's left side and it is the result of occurring of event eOLNF with one of the events eORN, eORNF, or eON.

In Figure 8, transitions caused by uncontrollable events are shown by red arrows (eRPR, eNOR, eNOL, eOF, eOL), while unobservable events are shown by green arrows (eER).

Changing the DES state entails building a new path for the robot. Entering states *DOL* or *DOR* generates a path to avoid the detected obstacle using a modification [41] of the waypoint guidance algorithm [44]. Entering state *NRP* constructs a path that connects the robot's current position with the reference path for what the path planning algorithm from [41] is used, which is based on Dubins curves [45].

Table 5. Events of the DES G_{msn}.

Name	Triggering Condition	Description
eOLN	$R^L_{\max} < R_{\min}$	The obstacle detected on the left is near
eOLNF	$R_{\min} \leq R^L_{\max} < R_{\min} + \Delta R$	The obstacle detected on the left is not far
eNOL	$R^L_{\max} = \infty$	There are no obstacles on the left
eORN	$R^R_{\max} < R_{\min}$	The obstacle detected on the right is near
eORNF	$R_{\min} \leq R^R_{\max} < R_{\min} + \Delta R$	The obstacle detected on the right is not far
eNOR	$R^R_{\max} = \infty$	There are no obstacles on the right
eON	$\rho_{\min} < \rho_n$	The detected obstacle is near
eOF	$\rho_{\min} > \rho_f$	The detected obstacle is far
eEAP		The robot has reached the end of the avoidance path
eRPR		The robot has reached the reference path
eAL1	eOLNF and (eORN $\vee$ eON)	The robot detours the obstacle from the left
eAL2	eEAP $\vee$ eOLNF	The robot detours the obstacle from the left
eAL3	(eOLNF $\vee$ eNOL) and (eORN $\vee$ eON)	The robot detours the obstacle from the left
eAL4	eON $\vee$ eOLNF	The robot detours the obstacle from the left
eAR1	eORNF and (eOLN $\vee$ eOLNF $\vee$ eON)	The robot detours the obstacle from the right
eAR2	eEAP $\vee$ eORNF	The robot detours the obstacle from the right
eAR3	(eORNF $\vee$ eNOR) and (eOLN $\vee$ eON)	The robot detours the obstacle from the right
eAR4	eON $\vee$ eOLRF	The robot detours the obstacle from the right
eRPF		The reference path is found
ePC		The path is corrected
eER		An error has occured
eNPR		A new path is required
eNPC		A new path has been computed
eOL		The obstacle is lost

7.2. System Constraints and Analysis

Let the specification for the problem above be provided by the automaton H_{msn} depicted in Figure 9. The desired robot behavior strategy can be expressed as follows: while DES G_{msn} is designed to ensure that the robot avoids places from which it is impossible to get out using the obstacle avoidance algorithms implemented (this is guaranteed by the rules related to obstacle avoidance modes), the specification language K_{msn} imposes additional constraints, which are interpreted as "do not change once chosen obstacle avoidance direction until the robot returns to the reference path" (left- or right-hand rule, which is ensured by the prohibition of transitions eAL3 and eAR3, and transitions ePRF) and "rebuild the path only if it is vital" (realized by the prohibition of transitions eEAP).

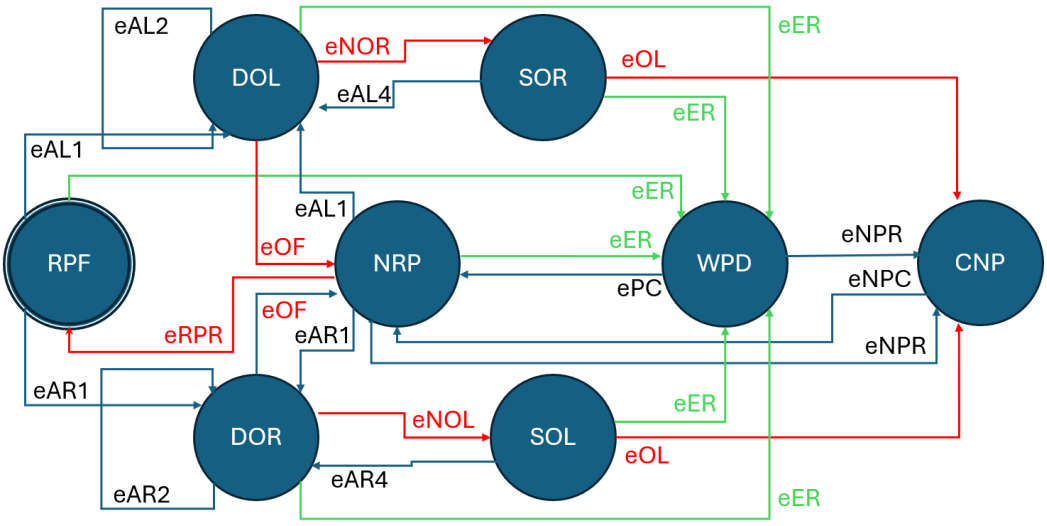

Figure 9. Specification H_{msn} for DES G_{msn}.

Before designing a supervisor as a controller that ensures specification K_{msn}, first, the supervisor existence criterion must be checked. By refuting the special PCF developed for controllability checking in [26], the prover Bootfrost established controllability of K_{msn}. The observability of K_{msn} is checked via inspection of automaton $T_{H_{msn}}$ constructed using the PCF $\mathcal{F}_{Obs}$ (2) (Section 3). $T_{H_{msn}}$ consists of 20 states and 45 transitions and does not contain the *dead* state, which means observability of K_{msn}. Full $T_{H_{msn}}$ may be found in supplementary materials at [24].

Once the specification K_{msn} is proved to be controllable and observable, a supervisor may be designed as described in Section 5. To realize its control action, we employ the PCF $\mathcal{F}^{T\#}_{BSCOP}$ (7), which allows processing events coming from outside and taking into account the knowledge available in the system. To handle composite events, predicate $T^{\#}$ can be taken in the form

$$T(\bar{e}) = \exists\ E(\bar{e}) \begin{cases} \forall \bar{s}, \bar{t}\ E(\bar{s}), |\bar{s}| > 1, Comp^{\#}(\bar{s}, \bar{t}), \bar{s} \subseteq \bar{t} - \exists\ Comp(\bar{s}) \\ \forall \bar{s}, \bar{t}\ E(\bar{s}), |\bar{s}| = 1, Proc^{\#}(\bar{s}) \text{ ——— } \exists\ E(\bar{s}) \\ \forall x\ Comp(x) \\ \forall x\ E(x) \end{cases}$$

The first question of PCF $T(\bar{e})$ is a general form of a question processing a composite event in the system. The number of such questions in the instantiated PCF equals the number of composite events in the system. In the case of G_{msn}, there are eight such questions for all eALi and eARi, and in each of them, the predicate $Comp(\bar{t})$ is computed according to Table 5 using corresponding propositional formulas. Below is a part of the instantiated form of $T^{\#}$ in our example:

$$T(\bar{e}) = \exists\ E(\bar{e}) \begin{cases} \forall \bar{s}, \bar{t}\ E(\bar{s}), |\bar{s}| > 1, \\ eAR^{\#}_1(\bar{s}, \{eOLNF, eOLN, eORNF, eON\}), \text{ —— } \exists\ eAR_1(\bar{s}) \\ \bar{s} \subseteq \{eOLNF, eOLN, eORNF, eON\} \\ \ldots \\ \forall \bar{s}, \bar{t}\ E(\bar{s}), |\bar{s}| = 1, Proc^{\#}(\bar{s}) \text{ ——— } \exists\ E(\bar{s}) \\ \forall x\ Comp(x) \\ \forall x\ E(x) \end{cases}$$

7.3. Simulation Results

The path-following and obstacle avoidance mission was realized at the robotic test bed by a LEGO Mindstorm EV3 robot with a differential drive. The robot moves due to the movement of two separately controlled wheels; therefore, Equation (9) turns to the following kinematic scheme:

$$\begin{cases} \dot{x} = r\frac{\omega_L + \omega_R}{2}\cos(\psi_W), \\ \dot{y} = r\frac{\omega_L + \omega_R}{2}\sin(\psi_W), \\ \dot{\psi}_W = (\omega_L - \omega_R)\frac{r}{l}, \end{cases} \tag{10}$$

where ω_L, ω_R are the corresponding angular speeds of rotation of the wheels, r is the radius of the wheel, and l is the distance between the wheels. The employed configuration of the EV3 robot has $2r = 56$ mm and $l = 100$ mm. To detect obstacles, the robot was equipped with a LIDAR with a viewing sector $[-60°;60°]$. For certainty, we accept $R_{\min} = 50$ cm, $D_s = 30$ cm, $\triangle R = 10$ cm, $\rho_n = 30$ cm, $\rho_f = 70$ cm, $N_b = 60$, $\rho_d = 150$ cm, $\underline{\alpha} = -60°$, and $\overline{\alpha} = 60°$.

The results of the preliminary computer simulation are shown in Figures 10 and 11. The robot moved along the predefined preference path denoted by a green arrow until it faced an obstacle. During the obstacle detouring maneuver, the obstacle on the right was lost, so DES G_{msn} entered the *SOL* state. Then, a failure was detected by the event eER,

which led G_{msn} into the NRP state and immediately back to DOR due to an obstacle on the right. Figure 11 shows changes in state G_{msn} corresponding to changes in the robot's functioning modes. A new path was generated in the DOR, NRP, and other states, which is denoted by the dots on the line in Figure 11.

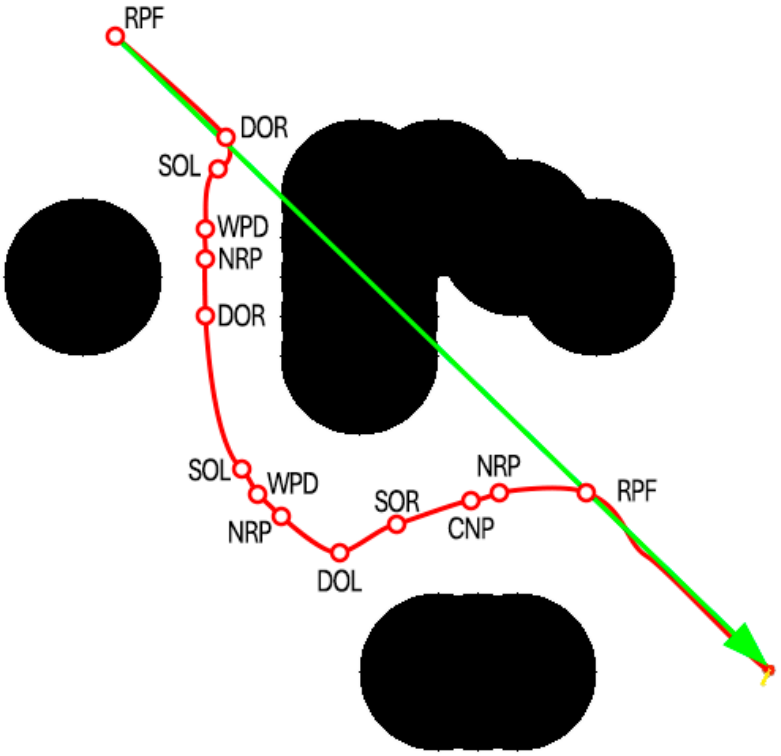

Figure 10. The result of computer modeling of the path-following and obstacle avoidance mission with the supervised partially observed DES G_{msn}.

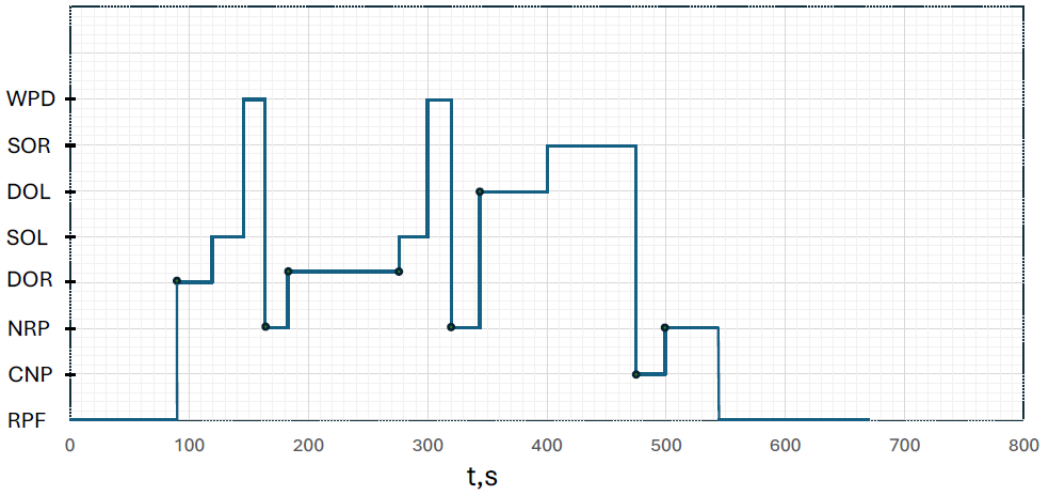

Figure 11. The changes in the state of G_{msn}. The marks on the line indicate moments when a new path is generated.

To explain the prover implementation, consider inference steps resulting in the sequence of the first state changes in Figure 11. In the initial state RPF, the $get\_event()$ function returned two atomic events eORNF and eON, which were passed as a list to the subinference of the $T^{\#}$ formula when its first question was checked. The subinference triggered the question containing eAR1. Instead of $\bar{s}$, the list $\{eORNF, eON\}$ was substituted and the formula for subinference was triggered:

$$eAR_1^{\#}(\{eORNF, eON\}, \ldots) = \exists eORNF, eON \begin{cases} \forall\, eORNF, eOLN \\ \forall\, eORNF, eOLNF, \\ \forall\, eORNF, eON. \end{cases}$$

This PCF corresponds to the negation of the propositional formula

$$\begin{aligned}(eORNF\ \&\ eON)\&(&\neg(eORNF\ \&\ eOLN)\vee\\ &\neg(eORNF\ \&\ eOLNF)\vee\\ &\neg(eORNF\ \&\ eON)),\end{aligned}$$

and is refuted in one step by answering the third question. Therefore, the constant *True* was returned to the formula $T^{\#}$, making an answer to its first question possible. Thus, the atom $eAR_1(\{eORNF, eON\})$ was added to the base. Then, by answering the question that corresponds to the question $\forall x\ Comp(x)$ of the formula $T^{\#}$, inference control is returned to the original formula $\mathcal{F}^{T\#}_{BSCOP}$. Instead of σ, eAR_1 was substituted in the consequent of its first question. Then, the atom $E(eAR_1)$ was added into the base B_{BSCOP}, which allowed answering the second question and adding the atom $L^{J_P/G_{msn}}(\varepsilon \cdot eAR_1, DOR)$ into the base. This addition caused a message to the main thread, which informed the program about the change in the DES's state.

The corresponding seminatural experiments were carried out on a robotic stand (Figure 12), which is described in detail in [46]. Due to changes in the obstacle's configuration and a number of uncontrollable factors affecting the robot dynamics, the trajectory of the robot is different compared with the one obtained in the computer simulation. However, the proper processing of the unobservable fault event eER led the implemented DES G_{msn} to WPD state, which allowed the robot to avoid a collision with the obstacle.

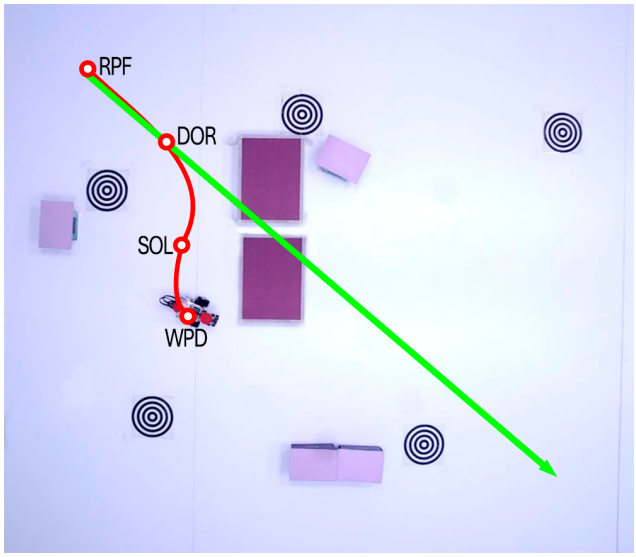

Figure 12. Simulation at the robotic test bed. The fault event eER led to WPD state, and a collision with the obstacle was avoided.

8. Discussion

This paper continues the work of developing a new way of formalizing and solving control problems for the important class of dynamic systems known as DESs. The PCF calculus provides powerful tools for dealing with sophisticated control problems, and above, it was shown how the PCF inference helps check the observability of regular languages.

Close to our research is an approach for testing the diagnosability of DESs based on their logical representation [47]. In [47], conjunctive normal forms (CNFs) are exploited to

study the diagnosability properties of DESs. Automata transitions are described as a set of clauses and when the well-known resolution method is applied to test whether failure events can be detected in a finite number of observable events. Keep in mind that CNF is less expressive compared with PCF, which means to represent automata underlying DESs, we leave the problem of the diagnosability of DESs and the intriguing comparison with the CNF-based approach for future research.

Moreover, the representation of DESs with the help of PCFs allows one to use information coming from the environment, as well as data on the functioning of the system itself, in the process of constructing the logical inference. This feature may help greatly in problems of supervisory controller synthesis and implementation for safety PLCs (programmable logic controllers) [48], where extended finite-state automata (EFA) are exploited. Transitions in an EFA may contain guards (i.e., logical conditions) over the variables and updates (i.e., assignments) to the variables. A simple guard, used in EFA, may replace a large automaton expressing the same requirement. In PCF, guards can be realized by event processing questions serving as special logical rules. Answers to them trigger subinferences, in which the events coming from the environment are used in calculations or other kinds of data processing.

Another promising direction of the PCF calculus's theoretical and practical development is embracing temporal logic and temporal-logic-based controls of DESs [49]. Since early papers, linear time temporal logic (LTL) has been proposed for specifying and verifying the safety and liveness properties of systems (e.g., [50,51]). LTL formulas allow formalizing such statements as "nothing bad will ever occur" and "something good such as accomplishment of tasks will occur regularly", so they cover a useful range of control specifications about finishing tasks regularly without compromising safety [49]. In subsequent years, computational tree logic (CTL) and epistemic temporal logic have been applied to deal with more sophisticated properties of DESs, e.g., property of stability which requires that the system should eventually reach a set of states where some statement holds and stay there forever [52–54].

It should be noted that the Bootfrost prover has not yet been formally verified by any specific testing software. Its applications currently lie in the areas of SCT problems mentioned in this and our previous articles, as well as in several case studies. In the future, it is planned to better demonstrate the validity of the work and the broad applicability of the presented approach. We will continue to develop the PCF-based approach so that it becomes a full-fledged tool capable of solving a wide range of tasks. In future works, its application to constructing supervisors for decentralized and distributed DESs will be suggested. Moreover, if the specification language is either not controllable or not observable, one may be interested in finding the less restricting controllable and observable sublanguage of the specification. While there are effective algorithms to construct the supremal controllable sublanguage of the given language, the supremal observable sublanguage does not exist. Only a maximal observable sublanguage may be found, which is not unique in general. Finding these languages and implementing them in control systems using the PCF approach is the line of our future research. Results obtained will be embedded at the different levels of the hierarchical control system for mobile robots.

Author Contributions: Conceptualization, A.D. and N.N.; formal analysis, A.D. and N.N.; investigation, A.D. and N.N.; methodology, A.D., A.L. and N.N.; resources, A.L.; software, A.D. and A.L.; validation, A.D. and N.N.; visualization, A.D. and A.L.; writing—original draft, A.D., A.L., and N.N.; writing—review and editing, A.D. and N.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and Higher Education of the Russian Federation, project no. 121032400051-9.

Data Availability Statement: The prover Bootfrost for the refutation of PCFs is written in the Rust programming language. The source code of the prover, documentation, and examples, including those presented in the manuscript, can be found on the GitHub page [24].

Acknowledgments: The authors thank Sergei Ul'yanov for providing the model and algorithms for obstacle avoidance, as well as Anton Tolstikhin for his enormous assistance in the practical implementation of the presented approach at the robotic test bed.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PCF	Positively constructed formula
ATP	Automated theorem proving
DES	Discrete event system
SCT	Supervisory control theory
CNF	Conjunctive normal form
EFA	Extended finite automaton
LTL	Linear time temporal logic
CTL	Computational trees logic
FOF	First-order formula
FOL	First-order logic
SCOP	Supervisory control and observation problem
BSCOP	Basic supervisory control and observation problem
ID	Identifier

References

1. Cassandras, C.G.; Lafortune, S. *Introduction to Discrete Event Systems*; Springer: Cham, Switzerland, 2021. [CrossRef]
2. Lafortune, S. Discrete Event Systems: Modeling, Observation, and Control. *Annu. Rev. Control Robot. Auton. Syst.* **2019**, *2*, 141–159. [CrossRef]
3. Seatzu, C.; Silva, M.; van Schuppen, J.H. (Eds.) *Control of Discrete-Event Systems*; Springer: London, UK, 2013. [CrossRef]
4. Ramadge, P.J.; Wonham, W.M. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* **1987**, *25*, 206–230. [CrossRef]
5. Dai, X.; Jiang, L.; Zhao, Y. Cooperative exploration based on supervisory control of multi-robot systems. *Appl. Intell.* **2016**, *45*, 18–29. [CrossRef]
6. Lopes, Y.K.; Trenkwalder, S.M.; Leal, A.B.; Dodd, T.J.; Groß, R. Supervisory control theory applied to swarm robotics. *Swarm Intell.* **2016**, *10*, 65–97. [CrossRef]
7. Hill, R.C.; Lafortune, S. Scaling the formal synthesis of supervisory control software for multiple robot systems. In Proceedings of the 2017 American Control Conference (ACC), Seattle, WA, USA, 24–26 May 2017; pp. 3840–3847. [CrossRef]
8. Wonham, W.M.; Cai, K. *Supervisory Control of Discrete-Event Systems*; Springer International Publishing: Cham, Switzerland, 2019.
9. Wonham, W.; Cai, K.; Rudie, K. Supervisory control of discrete-event systems: A brief history. *Annu. Rev. Control* **2018**, *45*, 250–256. [CrossRef]
10. Hales, T.; Adams, M.; Bauer, G.; Dang, T.D.; Harrison, J.; Hoang, L.T.; Kaliszyk, C.; Magron, V.; Mclaughlin, S.; Nguyen, T.T.; et al. A formal proof of the kepler conjecture. *Forum Math. Pi* **2017**, *5*, e2. [CrossRef]
11. Klein, G.; Andronick, J.; Elphinstone, K.; Heiser, G.; Cock, D.; Derrin, P.; Elkaduwe, D.; Engelhardt, K.; Kolanski, R.; Norrish, M.; et al. SeL4: Formal Verification of an Operating-System Kernel. *Commun. ACM* **2010**, *53*, 107–115. [CrossRef]
12. Gonthier, G. Formal Proof—The Four-Color Theorem. *Not. Am. Math. Soc.* **2008**, *11*, 1382–1393.
13. Leroy, X. Formal Verification of a Realistic Compiler. *Commun. ACM* **2009**, *52*, 107–115. [CrossRef]
14. Karpas, E.; Magazzeni, D. Automated planning for robotics. *Annu. Rev. Control Robot. Auton. Syst.* **2020**, *3*, 417–439. [CrossRef]
15. Zombori, Z.; Urban, J.; Brown, C.E. Prolog technology reinforcement learning prover. In Proceedings of the International Joint Conference on Automated Reasoning, Paris, France, 1–4 July 2020; Springer: Cham, Switzerland, 2020; pp. 489–507.
16. Schader, M.; Luke, S. Planner-Guided Robot Swarms. In Proceedings of the International Conference on Practical Applications of Agents and Multi-Agent Systems, L'Aquila, Italy, 7–9 October 2020; Springer: Cham, Switzerland, 2020; pp. 224–237.
17. Li, W.; Miyazawa, A.; Ribeiro, P.; Cavalcanti, A.; Woodcock, J.; Timmis, J. From Formalised State Machines to Implementations of Robotic Controllers. In *Distributed Autonomous Robotic Systems: The 13th International Symposium*; Groß, R., Kolling, A., Berman, S., Frazzoli, E., Martinoli, A., Matsuno, F., Gauci, M., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 517–529. [CrossRef]
18. Vassilyev, S.N. Machine synthesis of mathematical theorems. *J. Log. Program.* **1990**, *9*, 235–266. [CrossRef]
19. Davydov, A.; Larionov, A.; Cherkashin, E. On the calculus of positively constructed formulas for automated theorem proving. *Autom. Control Comput. Sci.* **2011**, *45*, 402–407. [CrossRef]

20. Cherkashin, E.A.; Postoenko, A.; Vassilyev, S.N.; Zherlov, A. New Logics for Intelligent Control. In Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference, Orlando, FL, USA, 1–5 May 1999; Kumar, A.N., Russell, I., Eds.; AAAI Press: Washington, DC, USA, 1999; pp. 257–261.
21. Zherlov, A.K.; Vassilyev, S.N.; Fedosov, E.A.; Fedunov, B.E. *Intelligent Control of Dynamic Systems*; Fizmatlit: Moscow, Russia, 2000. (In Russian)
22. Vassilyev, S.; Galyaev, A. Logical-optimization approach to pursuit problems for a group of targets. *Dokl. Math.* **2017**, *95*, 299–304. [CrossRef]
23. Vassilyev, S.; Ponomarev, G. Automation methods for logical derivation and their application in the control of dynamic and intelligent systems. *Proc. Steklov Inst. Math.* **2012**, *276*, 161–179. [CrossRef]
24. Larionov, A. Bootfrost. Available online: https://github.com/snigavik/bootfrost (accessed on 26 April 2024).
25. Klein, G.; Andronick, J.; Keller, G.; Matichuk, D.; Murray, T.; O'Connor, L. Provably trustworthy systems. *Philos. Trans. R. Soc. Math. Phys. Eng. Sci.* **2017**, *375*, 20150404. [CrossRef] [PubMed]
26. Davydov, A.; Larionov, A.; Nagul, N.V. The construction of controllable sublanguage of specification for DES via PCFs based inference. In Proceedings of the 2nd International Workshop on Information, Computation, and Control Systems for Distributed Environments, ICCS-DE 2020, Irkutsk, Russia, 6–7 July 2020; Bychkov, I., Tchernykh, A., Eds.; CEUR-WS.org: Aachen, Germany, 2020; CEUR Workshop Proceedings, Volume 2638, pp. 68–78.
27. Cho, H.; Marcus, S.I. Supremal and maximal sublanguages arising in supervisor synthesis problems with partial observations. *Math. Syst. Theory* **1989**, *22*, 177–211. [CrossRef]
28. Inan, K. An algebraic approach to supervisory control. *Math. Control Signals Syst.* **1992**, *5*, 151–164. [CrossRef]
29. Rudie, K.; Murray Wonham, W. The infimal prefix-closed and observable superlanguange of a given language. *Syst. Control Lett.* **1990**, *15*, 361–371. [CrossRef]
30. Feng, L.; Wonham, W.M. TCT: A computation tool for supervisory control synthesis. In Proceedings of the 2006 8th International Workshop on Discrete Event Systems, Ann Arbor, MI, USA, 10–12 July 2006; pp. 388–389.
31. Lafortune, S. Desuma. Available online: https://gitlab.eecs.umich.edu/wikis/desuma (accessed on 15 April 2024).
32. Ricker, L.; Lafortune, S.; Genc, S. DESUMA: A Tool Integrating GIDDES and UMDES. In Proceedings of the 2006 8th International Workshop on Discrete Event Systems, Ann Arbor, MI, USA, 10–12 July 2006; pp. 392–393. [CrossRef]
33. Åkesson, K.; Fabian, M.; Flordal, H.; Vahidi, A.; Malik, R. Supremica. Available online: https://supremica.org/ (accessed on 5 April 2024).
34. Malik, R.; Åkesson, K.; Flordal, H.; Fabian, M. Supremica—An Efficient Tool for Large-Scale Discrete Event Systems. *IFAC-PapersOnLine* **2017**, *50*, 5794–5799. [CrossRef]
35. Åkesson, K.; Flordal, H.; Fabian, M. Exploiting modularity for synthesis and verification of supervisors. *IFAC Proc. Vol.* **2002**, *35*, 175–180. [CrossRef]
36. Brandin, B.; Malik, R.; Malik, P. Incremental verification and synthesis of discrete-event systems guided by counter examples. *IEEE Trans. Control Syst. Technol.* **2004**, *12*, 387–401. [CrossRef]
37. Mohajerani, S.; Malik, R.; Fabian, M. A Framework for Compositional Synthesis of Modular Nonblocking Supervisors. *IEEE Trans. Autom. Control* **2014**, *59*, 150–162. [CrossRef]
38. Larionov, A.; Davydov, A.; Cherkashin, E. The method for translating first-order logic formulas into positively constructed formulas. *Softw. Syst.* **2019**, *4*, 556–564. [CrossRef]
39. Larionov, A.; Davydov, A.; Cherkashin, E. The calculus of positively constructed formulas, its features, strategies and implementation. In Proceedings of the 2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 20–24 May 2013; pp. 1023–1028.
40. Sutcliffe, G. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *J. Autom. Reason.* **2017**, *59*, 483–502. [CrossRef]
41. Ulyanov, S.; Bychkov, I.; Maksimkin, N. Event-Based Path-Planning and Path-Following in Unknown Environments for Underactuated Autonomous Underwater Vehicles. *Appl. Sci.* **2020**, *10*, 7894. [CrossRef]
42. Bychkov, I.; Ulyanov, S.; Nagul, N.; Davydov, A.; Kenzin, M.; Maksimkin, N. Hierarchical event-based control of multi-robot systems in unstructured environments. *J. Phys. Conf. Ser.* **2021**, *1864*, 012001. [CrossRef]
43. Lapierre, L.; Soetanto, D. Nonlinear path-following control of an AUV. *Ocean Eng.* **2007**, *34*, 1734–1744. [CrossRef]
44. Yan, Z.; Li, J.; Zhang, G.; Wu, Y. A Real-Time Reaction Obstacle Avoidance Algorithm for Autonomous Underwater Vehicles in Unknown Environments. *Sensors* **2018**, *18*, 438. [CrossRef] [PubMed]
45. Dubins, L.E. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *Am. J. Math.* **1957**, *79*, 497–516. [CrossRef]
46. Kostylev, D.; Tolstikhin, A.; Ulyanov, S. Development of the complex modelling system for intelligent control algorithms testing. In Proceedings of the 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 20–24 May 2019; pp. 943–948. [CrossRef]
47. Geng, X.; Ouyang, D.; Han, C. Verifying Diagnosability of Discrete Event System with Logical Formula. *Chin. J. Electron.* **2020**, *29*, 304–311. [CrossRef]
48. Reijnen, F.F.; Erens, T.R.; van de Mortel-Fronczak, J.M.; Rooda, J.E. Supervisory controller synthesis and implementation for safety PLCs. *Discret. Event Dyn. Syst.* **2022**, *32*, 115–141. [CrossRef]

49. Seow, K.T. Supervisory Control of Fair Discrete-Event Systems: A Canonical Temporal Logic Foundation. *IEEE Trans. Autom. Control* **2021**, *66*, 5269–5282. [CrossRef]
50. Thistle, J.G.; Wonham, W.M. Control problems in a temporal logic framework. *Int. J. Control* **1986**, *44*, 943–976. [CrossRef]
51. Rawlings, B.C.; Lafortune, S.; Ydstie, B.E. Supervisory Control of Labeled Transition Systems Subject to Multiple Reachability Requirements via Symbolic Model Checking. *IEEE Trans. Control Syst. Technol.* **2020**, *28*, 644–652. [CrossRef]
52. Jiang, S.; Kumar, R. Supervisory Control of Discrete Event Systems with CTL* Temporal Logic Specifications. *SIAM J. Control Optim.* **2006**, *44*, 2079–2103. [CrossRef]
53. Aucher, G. Supervisory Control Theory in Epistemic Temporal Logic. In Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems, Paris, France, 5–9 May 2014; Volume 1.
54. Ritsuka, K.; Rudie, K. Do what you know: Coupling knowledge with action in discrete-event systems. *Discret. Event Dyn. Syst.* **2023**, *33*, 257–277. [CrossRef]