MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 31, NO. 1, 2024
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

On the Application of the Calculus of Positively Constructed Formulas for the Study of Controlled Discrete-Event Systems

A. V. Davydov[1], A. A. Larionov[1], N. V. Nagul[1]

DOI: 10.18255/1818-1015-2024-1-54-77

[1]Matrosov Institute for System Dynamics and Control Theory SB RAS, Irkutsk, Russia

MSC2020: 68V15, 93C65
Research article
Full text in Russian

Received February 2, 2024
Revised February 8, 2024
Accepted February 14, 2024

The article is devoted to the development of an approach to solving the main problems of the theory of supervisory control of logical discrete-event systems (DES), based on their representation in the form of positively constructed formulas (PCF). We consider logical DESs in automata form, understood as generators of some regular languages. The PCF language is a complete first-order language, the formulas of which have a regular structure of alternating type quantifiers and do not contain a negation operator in the syntax. It was previously proven that any formula of the classical first-order predicate calculus can be represented as a PCF. PCFs have a visual tree representation and a natural question-and-answer procedure for searching for an inference using a single inference rule. It is shown how the PCF calculus, developed in the 1990s to solve some problems of control of dynamic systems, makes it possible to solve basic problems of the theory of supervisory control, such as checking the criteria for the existence of supervisory control, automatically modifying restrictions on the behavior of the controlled system, and implementing a supervisor. Due to some features of the PCF calculus, it is possible to use a non-monotonic inference. It is demonstrated how the presented PCF-based method allows for additional event processing during inference. The Bootfrost software system, or the so-called prover, designed to refute the obtained PCFs is also presented, and the features of its implementation are briefly described. As an illustrative example, we consider the problem of controlling an autonomous mobile robot.

Keywords: positively constructed formula; automated theorem proving; prover; discrete event system; supervisory control

INFORMATION ABOUT THE AUTHORS

Davydov, Artem V.	ORCID iD: 0000-0002-8703-3096. E-mail: artem@icc.ru Research fellow
Larionov, Aleksandr A.	ORCID iD: 0000-0001-7116-9338. E-mail: bootfrost@zoho.com Programmer
Nagul, Nadezhda V. (corresponding author)	ORCID iD: 0000-0003-1439-3274. E-mail: sapling@icc.ru Senior researcher, PhD

Funding: Ministry of Science and Higher Education of the Russian Federation (121032400051-9).

For citation: A. V. Davydov, A. A. Larionov, and N. V. Nagul, "On the application of the calculus of positively constructed formulas for the study of controlled discrete-event systems", *Modeling and Analysis of Information Systems*, vol. 31, no. 1, pp. 54–77, 2024. DOI: 10.18255/1818-1015-2024-1-54-77.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 31, № 1, 2024
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

О применении исчисления позитивно-образованных формул для исследования управляемых дискретно-событийных систем

А. В. Давыдов[1], А. А. Ларионов[1], Н. В. Нагул[1]

DOI: 10.18255/1818-1015-2024-1-54-77

[1]Институт динамики систем и теории управления им. В.М. Матросова СО РАН, Иркутск, Россия

УДК 004.832.3
Научная статья
Полный текст на русском языке

Получена 2 февраля 2024 г.
После доработки 8 февраля 2024 г.
Принята к публикации 14 февраля 2024 г.

Статья посвящена разработке подхода к решению основных задач теории супервизорного управления логическими дискретно-событийными системами (ДСС), основанного на представлении их в виде позитивно-образованных формул (ПОФ). Рассматриваются логические ДСС в автоматной форме, понимаемые как генераторы некоторых регулярных языков. Язык ПОФ представляет собой полный язык первого порядка, формулы которого имеют регулярную структуру из чередующихся типовых кванторов и не содержат в синтаксисе оператора отрицания. Ранее было доказано, что любая формула классического исчисления предикатов первого порядка может быть представлена в виде ПОФ. ПОФ имеют наглядное древовидное представление и естественную вопросно-ответную процедуру поиска вывода с помощью единственного правила вывода. Показано, как разработанное в 1990-х годах для решения некоторых задач управления динамическими системами исчисление ПОФ позволяет решать базовые задачи теории супервизорного управления, такие как проверка критериев существования супервизорного управления, автоматическая модификация ограничений на поведение управляемой системы и реализация супервизора. Благодаря некоторым особенностям исчисления ПОФ существует возможность применения немонотонного вывода. Продемонстрировано, как представленный метод на основе ПОФ позволяет выполнять дополнительную обработку событий во время логического вывода. Также представлена программная система Bootfrost, или так называемый прувер, разработанный для опровержения полученных ПОФ, кратко описываются особенности его реализации. В качестве иллюстративного примера рассматривается задача управления автономным мобильным роботом.

Ключевые слова: позитивно-образованная формула; автоматическое доказательство теорем; прувер; дискретно-событийная система; супервизорное управление

ИНФОРМАЦИЯ ОБ АВТОРАХ

Давыдов, Артем Васильевич	ORCID iD: 0000-0002-8703-3096. E-mail: artem@icc.ru Научный сотрудник
Ларионов, Александр Александрович	ORCID iD: 0000-0001-7116-9338. E-mail: bootfrost@zoho.com Программист
Нагул, Надежда Владимировна (автор для корреспонденции)	ORCID iD: 0000-0003-1439-3274. E-mail: sapling@icc.ru Старший научный сотрудник, к.ф.-м.н.

Финансирование: Министерство науки и высшего образования РФ (121032400051-9).

Для цитирования: A. V. Davydov, A. A. Larionov, and N. V. Nagul, “On the application of the calculus of positively constructed formulas for the study of controlled discrete-event systems”, *Modeling and Analysis of Information Systems*, vol. 31, no. 1, pp. 54–77, 2024. DOI: 10.18255/1818-1015-2024-1-54-77.

Введение

Класс дискретно-событийных систем (ДСС) — это широкий класс моделей, как правило, используемых для представления сложных технических и технологических систем, таких как производственные и сборочные процессы, коммуникационные протоколы, транспортные системы, системы управления базами данных, в которых смена дискретных состояний системы происходит вследствие возникновения некоторых дискретных событий [1—3]. Для описания таких систем используются различные формализмы, которые могут включать или не включать явное указание времени, например, конечные автоматы, сети Петри, диоидные алгебры, темпоральные логики и др. Если важна только последовательность событий, а не время их возникновения и длительность, говорят о логических ДСС. Два формализма наиболее часто применяются для исследования логических ДСС: конечные автоматы, являющиеся самыми наглядными моделями, подверженными, однако, проклятию размерности, и сети Петри, обладающие компактностью представления и достаточной выразительностью.

В случае, когда некоторые события системы могут быть запрещены, возникает возможность ограничить поведение системы в пределах, заданных определенной спецификацией. Для исследования и построения управляемых логических ДСС была разработана теория супервизорного управления (ТСУ) [4], современное состояние которой представлено, например, в [3, 5, 6]. Соответствующее внешнее средство управления называется супервизором, а система, следуя теории автоматического управления, — объектом управления. Например, логические ДСС и ТСУ в настоящее время широко используются в робототехнике. Публикации в этой области касаются управления одиночными роботами [7, 8], группами роботов [9—11], формациями роботов [12] и их роями [13, 14]. Несмотря на интенсивные исследования, проводимые с 1980-х годов, многие задачи ТСУ ожидают новых подходов к их решению, что связано со сложностью реализации известных алгоритмов исследования и построения ДСС, особенно частично наблюдаемых, децентрализованных или распределенных.

В представлении ДСС с помощью конечного автомата переходы из состояния в состояние помечены буквами некоторого конечного алфавита и соответствуют событиям, происходящим в системе. Последовательности таких переходов образуют слова регулярного языка, описывающего поведение системы с высокоуровневой, или символьной, точки зрения. Свойства системы, такие как живучесть, безопасность, неблокирование и др., могут быть описаны как утверждения над этими формальными выражениями, а для проверки истинности таких утверждений оказывается возможным использовать автоматическое доказательство теорем (АДТ). Разработанное для проверки формальных математических доказательств и реализованное с помощью специальных компьютерных программ, называемых пруверами, АДТ в настоящее время применяется в широком ряде областей, включая анализ программ, верификацию систем и др. Последние примеры включают доказательство 400-летней гипотезы Кеплера об упаковке сфер [15] и корректность ядра операционной системы seL4 [16], не говоря уже о более раннем доказательстве теоремы о четырёх красках в теории графов [17] и верификацию компилятора языка C CompCert [18]. В [19] АДТ применяется для проверки закодированных специальным образом условий верификации (verification conditions) с помощью системы ACL2 ([20]). Важной областью современного применения АДТ является робототехника, где с его помощью осуществляется планирование [21] и принятие решений [22]. Например, в [23] для планирования и управления роем роботов используется язык PDDL, основанный на классическом АДТ в стиле STRIPS. В [24] доказательство теорем предлагается использовать для проверки структуры моделирования контроллеров автономных роботов в сочетании с автоматической генерацией кода на C++.

Как известно, использование АДТ сопряжено с рядом известных трудностей, таких как критическое увеличение размера базы фактов, с которыми работает прувер, или следование по ложному

пути при поиске вывода. Для преодоления этих трудностей в 1990-е годы академиком С. Н. Васильевым и А. К. Жерловым было разработано исчисление позитивно-образованных формул (ПОФ) и основанный на нем метод АДТ [25, 26]. ПОФ — это формулы первого порядка, имеющие регулярную структуру и не содержащие символа отрицания. Метод АДТ на основе ПОФ представляет собой мощный инструмент качественного анализа систем, прежде всего, динамических, а приложения включают ориентацию телескопа, управление группой лифтов [26], обеспечение достижимости множества целей [27], преследование подвижных целей [28]. Для автоматизации логического вывода в языке ПОФ разработан прувер Bootfrost[1]. Наиболее важными особенностями исчисления ПОФ и его программной реализации являются следующие:

- крупноблочные структуры данных для представления формул и правил вывода;
- отсутствие необходимости удаления кванторов существования с помощью процедуры сколемизации, что снижает сложность поиска вывода;
- совместимость с эвристиками, специфичными для приложения, и общими эвристиками управления поиском логического вывода;
- наглядность логического вывода, что помогает найти ошибки формализации;
- возможность модификации семантики для поддержки неклассических логик.

Помимо других преимуществ, прувер Bootfrost сочетает в себе возможности полностью автоматического вывода с возможностью реализации разработанных пользователем стратегий вывода.

Принимая во внимание известные примеры применения АДТ в исчислении ПОФ к управлению динамическими системами, нами предложен новый способ исследования и проектирования логических ДСС, основанный на ПОФ. Предлагаемый метод может быть использован на различных этапах построения системы управления, включая этап проверки спецификации на управляемость и наблюдаемость, в случае частичного наблюдения событий в системе, модификацию спецификации для обеспечения управляемости, если свойство управляемости не выполнено, реализацию управления на верхнем символьном уровне. Представление ДСС с помощью ПОФ позволяет использовать в процессе построения логического вывода информацию, поступающую из окружающей среды, а также данные о функционировании самой системы. Эта функция может существенно помочь в задачах синтеза и реализации супервизоров для программируемых логических контроллеров (ПЛК) [29], где используются расширенные автоматы с конечным числом состояний (РКА). Переходы в РКА могут содержать условия (например, логические условия) на переменные и действия (например, обновление) на переменные. В ПОФ условия переходов могут быть реализованы с помощью вопросов обработки событий, служащих специальными логическими правилами. Ответы на них запускают подвыводы, в которых события, поступающие из окружающей среды, используются в вычислениях или иной обработке данных.

Отметим близкое к представляемому подходу исследование [30, 31], где для верификации ПЛК показана эффективность применения ограниченных хорновских дизьюнктов (constrained Horn clauses). Требование безопасности программы для ПЛК формулируется как свойство автоматов потоков управления (control flow automaton). С помощью известного SMT-солвера Z3 [32] оказывается возможным доказать безопасность формализованной программы для ПЛК, причем предварительно применяется процедура построения абстракции для уменьшения сложности вывода. Заметим, что исчисление ПОФ предлагается использовать для решения задач, предшествующих реализации желаемого поведения управляемой системы с помощью ПЛК, т. е. задач более высокого уровня абстракции. Однако, реализация построенного управления несомненно является важной задачей, решению которой посвящено немало работ в области ДСС. Так, в [33] представлен метод, который позволяет проектировщику эффективным образом внедрять результаты, полученные с помощью

[1] https://github.com/snigavik/bootfrost

ТСУ, для координации работы подсистем в программе для ПЛК, реализующей конкретную архитектуру супервизорного управления.

Еще одним направлением теоретического и практического развития ПОФ-исчисления является исследование темпоральной логики и применение ПОФ-исчисления для автоматизации поиска темпоральных выводов, а также использование темпоральной логики для исследования и управления ДСС. Так, в рамках ТСУ логика линейного времени (LTL) использовалась для задания и проверки свойств безопасности и живучести системы (например, [34, 35]). Формулы LTL позволяют формализовать такие утверждения как «ничего плохого никогда не произойдет» и «что-то хорошее, например, выполнение задач, будет происходить регулярно», поэтому они охватывают полезный набор спецификаций, связанных с регулярным достижением заданных целей без ущерба для безопасности системы [36]. В последующем, логика деревьев вычислений (CTL) и эпистемическая темпоральная логика начали применяться для работы с более сложными свойствами ДСС, например, свойством устойчивости, которое требует, чтобы система в конечном итоге достигла набора состояний, в которых выполняется какое-то утверждение, и оставалась там навсегда [37—39].

Близким к нашему исследованию является подход к тестированию диагностируемости ДСС на основе ее логического представления, предложенного в [40]. В [40] для изучения свойств диагностируемости ДСС используются конъюнктивные нормальные формы (КНФ). Переходы автоматов описываются как множество дизъюнктов, и затем хорошо известный метод резолюций применяется для проверки того, можно ли обнаружить события сбоя среди конечного числа наблюдаемых событий. Учитывая, что КНФ является менее выразительным, по сравнению с ПОФ, средством представления автоматов, лежащих в основе ДСС, мы оставляем проблему диагностируемости ДСС и сравнение с подходом на основе КНФ для будущих исследований.

В рамках ТСУ разработано несколько инструментов для анализа и проектирования управляемых ДСС. Среди них ТСТ [41], DESUMA/UMDES[2] [42], Supremica[3] и другие. В этом ряду следует отметить систему Supremica, имеющую удобный графический интерфейс. Supremica поддерживает РКА, в которых условия и действия переходов оперируют переменными, изменяющимися в конечных целочисленных диапазонах или имеющими перечисляемый тип [43]. Хотя программная система для решения задач ТСУ на основе ПОФ находится в стадии развития, наш подход предполагает, что как условия переходов, так и действия после них могут быть выражены в форме логических утверждений любого рода. При этом формат выходных данных прувера Bootfrost может быть организован таким образом, чтобы обеспечить интерпретацию полученных результатов в системе Supremica.

В отличие от крупномасштабных промышленных примеров в [44—46], применение исчисления ПОФ для управления ДСС предполагает использование логических инструментов для представления и обработки знаний. Конструктивная семантика ПОФ-исчисления позволяет извлекать знания (например, планы действий для автономных роботов) из построенных выводов, а немонотонность вывода и возможный учет времени могут применяться для планирования действий в динамично меняющихся предметных областях.

Цель данной статьи — дать представление о ПОФ-исчислении и его применимости для решения основных задач ТСУ, таких как проверка управляемости заданной спецификации, построение управляемого подъязыка заданного языка спецификации и реализация супервизорного управления. В статье описываются основные элементы предлагаемого подхода.

[2] https://gitlab.eecs.umich.edu/wikis/desuma
[3] https://supremica.org

1. Предварительные сведения

1.1. Исчисление позитивно-образованных формул

Рассмотрим язык логики первого порядка, состоящий из первопорядковых логических формул (ПЛФ), построенных из атомарных формул с помощью операторов $\&$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, кванторных символов $\forall$ и $\exists$ и констант *true* и *false*. Понятия термин, атом, литер определяются обычным образом. Далее неатомарные формулы и подформулы будут обозначаться прописными каллиграфическими буквами ($\mathcal{F}, \mathcal{P}, \mathcal{Q}$ и т. д.), возможно, с индексами. Множества формул будут обозначаться прописными греческими буквами (Φ, Ψ и т. д.), возможно, с индексами.

Пусть $X = \{x_1, \ldots, x_k\}$ — множество переменных, $A = \{A_1, \ldots, A_m\}$ — множество атомарных формул, называемое *конъюнктом*, а $\Phi = \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$ — множество ПЛФ. Формулы $\forall x_1 \ldots \forall x_k (A_1 \& \ldots \& A_m) \rightarrow (\mathcal{F}_1 \vee \ldots \vee \mathcal{F}_n)$ и $\exists x_1 \ldots \exists x_k (A_1 \& \ldots \& A_m) \& (\mathcal{F}_1 \& \ldots \& \mathcal{F}_n)$ обозначаются как $\forall x_1, \ldots, x_k A_1, \ldots, A_m\{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$ и $\exists x_1, \ldots, x_k\, A_1, \ldots, A_m\{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$, соответственно. Их можно сокращенно обозначать $\forall_X A\ \Phi$ и $\exists_X A\ \Phi$, имея в виду, что $\forall$-квантор соответствует $\rightarrow \Phi^{\vee}$, где $\Phi^{\vee}$ означает дизъюнкцию всех формул из Φ, а $\exists$-квантор соответствует $\& \Phi^{\&}$, где $\Phi^{\&}$ означает конъюнкцию всех формул из Φ. Любое из множеств X, A, Φ может быть пустым, и в этом случае при формализации формул их можно опустить. Таким образом, если $Q \in \{\forall, \exists\}$, то $Q_X A\ \varnothing \equiv Q_X A$, $Q_X \varnothing\ \Phi \equiv Q_X\ \Phi$ и $Q_{\varnothing} A\ \Phi \equiv QA\ \Phi$. Поскольку пустая дизъюнкция тождественна *false*, а пустая конъюнкция тождественна *true*, то корректны следующие эквивалентности: $\forall_X A\ \varnothing \equiv \forall_X A \rightarrow false \equiv \forall_X A$ и $\exists_X A\ \varnothing \equiv \exists_X A \& true \equiv \exists_X A$ и $\forall \varnothing\ \Phi \equiv true \rightarrow \Phi \equiv \forall\ \Phi$ и $\exists \varnothing\ \Phi \equiv true \& \Phi \equiv \exists\ \Phi$. Кванторы без переменных называются *фиктивными* и не опускаются, так как позволяют корректно интерпретировать множество ПЛФ Φ и делают структуру формулы регулярной. Если $\mathcal{P} = \forall A\ \Phi$, то формула $\mathcal{P}$ интерпретируется как $A \rightarrow \Phi^{\vee}$. А если $\mathcal{P} = \exists A\ \Phi$, то $\mathcal{P}$ интерпретируем как $A \& \Phi^{\&}$.

Фразы $\forall_X A\ \Phi$ и $\exists_X A\ \Phi$ обычно обозначают такие высказывания на естественном языке, как, например, «для всех X, удовлетворяющих свойству A, существует Φ», «для всех целых x, y, z и $n > 2$ существует $x^n + y^n \neq z^n$», и т. д. Первоначально конструкции $\forall_{XA}$ и $\exists_{XA}$ назывались *ти́повыми кванторами* и были введены Н. Бурбаки [47] как часть нотации для формализации математики.

Определение 1 (Позитивно-образованные формулы (ПОФ))**.** Синтаксис языка ПОФ в нотации Бэкуса-Наура можно представить следующим образом:

<pcf–formula>::= ∀ {<e–formulas>}
<a–formulas> ::= "" | <a–formula> | <a–formula>, <a–formulas>
<e–formulas> ::= "" | <e–formula> | <e–formula>, <e–formulas>
<e–formula> ::= ∃ <vars> <conjunct> {<a–formulas>}
<a–formula> ::= ∀ <vars> <conjunct> {<e–formulas>}
<vars> ::= "" | <symbol> | <symbol> , <vars>
<conjunct> ::= "" | <atom> | <atom> , <conjunct>
<atom> :: <symbol> (<terms>)
<terms> ::= "" | <term> | <term> , <terms>
<term> :: <symbol> | <symbol> (<terms>)
<symbol> ::= [a–zA–Z0–9_]+

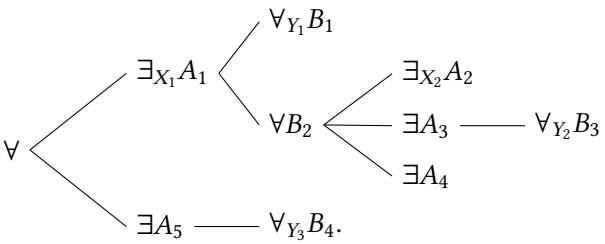
Термин «позитивные» обусловлен тем, что согласно определению, ПОФ не содержат оператор отрицания ($\neg$). Это справедливо и для семантики языка: отрицание «скрыто» в импликациях, которые подразумеваются после кванторов всеобщности. Отметим, что любая ПЛФ может быть представлена в виде ПОФ, поскольку ПОФ-язык — это специальный способ записи классических ПЛФ, так же как, например, конъюнктивные нормальные формы, дизъюнктивные нормальные формы и т. д. Алгоритм преобразования ПЛФ в ПОФ и обратно представлен в [48].

Если ПОФ имеет вид $\forall\ \Phi$, то такая ПОФ называется *канонической ПОФ* (это самый верхний нетерминальный символ в определении, `pcf-formula`). Очевидно, что любая ПОФ может быть приведена к канонической форме. Если $\mathcal{P}$ — неканоническая $\exists$-ПОФ, то $\mathcal{P} \equiv \forall\ \mathcal{P}$, так как $\forall\ \mathcal{P} \equiv true \rightarrow \mathcal{P} \equiv \mathcal{P}$. Если $\mathcal{P}$ — неканоническая $\forall$-ПОФ, то $\mathcal{P} \equiv \forall\ \{\exists\ \mathcal{P}\}$, так как $\forall\ \{\exists\ \mathcal{P}\} \equiv true \rightarrow (true \& \mathcal{P}) \equiv \mathcal{P}$.

Для удобства чтения мы представляем ПОФ в виде деревьев, узлами которых являются типовые кванторы, и используем привычные для них наименования: *узел, корень, лист, ветвь.* Например, ПОФ:

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

представляется в виде дерева следующим образом:



Если дана ПОФ $\mathcal{P} = \forall\{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$ и $\mathcal{F}_i = \exists_{X_i} B_i\{\mathcal{Q}_{i1}, \ldots, \mathcal{Q}_{im}\}$, $i = \overline{1,n}$, то $\mathcal{F}_i$ будут называться *базовыми подформулами* $\mathcal{P}$, B_i называются *базами фактов* или просто *базами*, $\mathcal{Q}_{ij}$ называются *подформулами-вопросами*, а корни подформул-вопросов называются *вопросами* к базе B_i, $i = \overline{1,n}$. Вопрос вида $\forall_X A$ (без дочерних элементов) называется *целевым вопросом.* Будем считать, что внутри базовых подформул любая переменная не может быть одновременно свободной и связанной, более того, она не может быть связана разными кванторами одновременно.

В процессе рассуждений часто доказывают утверждение $\mathcal{F}$, опровергая его отрицание. Мы намерены действовать аналогичным образом. Метод применяется только к каноническим ПОФ.

Определение 2 (Ответ)**.** Рассмотрим некоторую базовую подформулу $\exists_X A\ \Psi$ ПОФ. Вопрос подформулы $\mathcal{Q} = \forall_Y B\ \Phi$, $Q \in \Psi$ имеет ответ θ тогда и только тогда, когда θ является подстановкой $Y \rightarrow H^\infty \cup X$ и $B\theta \subseteq A$, где H^∞ — эрбрановский универсум в основе которого лежат символы констант и функций, встречающиеся в соответствующей базовой подформуле.

Определение 3 (Merge)**.** Пусть $\mathcal{P}_1 = \exists_X A\ \Psi$ и $\mathcal{P}_2 = \exists_Y B\ \Phi$, тогда $merge(\mathcal{P}_1, \mathcal{P}_2) = \exists_{X \cup Y} A \cup B\ \Psi \cup \Phi$.

Определение 4 (Split)**.** Рассмотрим некоторую базовую подформулу $\mathcal{B} = \exists_X A\ \Psi$. Пусть подформула-вопрос $\mathcal{Q} \in \Psi$ имеет вид $\forall_Y D\ \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, где $\mathcal{P}_i = \exists_{Z_i} C_i\ \Gamma_i, i = \overline{1,n}$, тогда $split(B, Q) = \{merge(B, \mathcal{P}_1'), \ldots,\ merge(B, \mathcal{P}_n')\}$, где $'$ — оператор переименования переменных. Будем говорить, что $\mathcal{B}$ расщепляется с помощью $\mathcal{Q}$, а $split(\mathcal{B}, \mathcal{Q})$ — результат расщепления $\mathcal{B}$. Очевидно, что $split(\mathcal{B}, \forall_Y D) = \varnothing$.

Определение 5 (Правило вывода ω)**.** Рассмотрим некоторую каноническую ПОФ $\mathcal{F} = \forall\ \Phi$. Если существует базовая подформула $\mathcal{B} = \exists_X A\ \Psi$, $\mathcal{B} \in \Phi$ и существует подформула-вопрос $\mathcal{Q} \in \Psi$, и вопрос $\mathcal{Q}$ имеет ответ θ на $\mathcal{B}$, тогда $\omega(\mathcal{F}) = \forall\ \Phi \setminus \{\mathcal{B}\} \cup split(\mathcal{B}, \mathcal{Q}\theta)$.

Обратим внимание, что если после применения правила ω множество Φ становится пустым, а ПОФ — лишь квантором $\forall$, то можно сделать вывод, что отрицание исходной формулы невыполнимо, следовательно, сама формула общезначима.

Любая конечная последовательность ПОФ $\mathcal{F}, \omega\mathcal{F}, \omega^2\mathcal{F}, \ldots, \omega^n\mathcal{F}$, где $\omega^s\mathcal{F} = \omega(\omega^{s-1}\mathcal{F})$, $\omega^1 = \omega$, $\omega^n\mathcal{F} = \forall$, называется *выводом* $\mathcal{F}$ в ПОФ-исчислении (с аксиомой $\forall$).

Предположим, что стандартная стратегия поиска выводов проверяет вопросы в последовательном порядке, повторяя проверку только тогда, когда список вопросов заканчивается, и не использует повторное применение ω к вопросу с той же подстановкой θ (вопросно-ответный метод автоматического поиска вывода). Подробные сведения о ПОФ-исчислении изложены в [26], [49].

Пример 1. Покажем вывод ПОФ $\mathcal{F}_1$.

$$\mathcal{F}_1 = \forall - \exists S(e) \begin{cases} \forall x\ S(x) \text{ —— } \exists A(x) \\ \forall x, y\ C(x,y) \\ \forall x\ A(x) \begin{cases} \exists y\ C(y, f(x)) \\ \exists \text{ —— } \forall x\ S(x), A(x) \text{ — } \exists C(x, f(x)). \end{cases} \end{cases}$$

На первом шаге можно найти только одну ответную подстановку: $\{x \to e\}$ для первого вопроса (обозначим вопросы $Q_i, i = \overline{1,3}$, нумеруя по возрастанию сверху вниз). После применения правила вывода ω с этим ответом к единственной базе $\mathcal{F}_1$ формула трансформируется и принимает следующий вид:

$$\mathcal{F}_2 = \forall - \exists S(e), A(e) \begin{cases} \forall x\ S(x) \text{ —— } \exists A(x) \\ \forall x, y\ C(x,y) \\ \forall x\ A(x) \begin{cases} \exists y\ C(y, f(x)) \\ \exists \text{ —— } \forall x\ S(x), A(x) \text{ — } \exists C(x, f(x)). \end{cases} \end{cases}$$

На втором шаге также доступен только один ответ на вопрос Q_3 с подстановкой $\{x \to e\}$. После применения ω формула расщепляется, поскольку после Q_3 имеется дизъюнктивное ветвление. Формула примет следующий вид:

$$\mathcal{F}_3 = \forall \begin{cases} \mathcal{F}_3^1 \\ \mathcal{F}_3^2, \end{cases}$$

где $\mathcal{F}_3^1$ имеет вид:

$$\mathcal{F}_3^1 \text{ —— } \exists y_1\ S(e), A(e), C(y_1, f(e)) \begin{cases} \forall x\ S(x) \text{ —— } \exists A(x) \\ \forall x, y\ C(x,y) \\ \forall x\ A(x) \begin{cases} \exists y\ C(y, f(x)) \\ \exists \text{ —— } \forall x\ S(x), A(x) \text{ — } \exists C(x, f(x)), \end{cases} \end{cases}$$

а $\mathcal{F}_3^2$ есть:

$$\mathcal{F}_3^2 \text{ —— } \exists S(e), A(e) \begin{cases} \forall x\ S(x) \text{ —— } \exists A(x) \\ \forall x, y\ C(x,y) \\ \forall x\ A(x) \begin{cases} \exists y\ C(y, f(x)) \\ \exists \text{ —— } \forall x\ S(x), A(x) \text{ — } \exists C(x, f(x)) \end{cases} \\ \forall x\ S(x), A(x) \text{ - } \exists C(x, f(x)). \end{cases}$$

На третьем шаге первая база может быть опровергнута ответом на целевой вопрос Q_2 с подстановкой $\{x \to y_1, y \to f(e)\}$. Затем опровергнутая база, как и вся базовая подформула $\mathcal{F}_3^1$, должны быть удалены из списка базовых подформул формулы $\mathcal{F}_3$.

На четвертом шаге можно найти ответ на четвертый, добавленный вопрос базовой подформулы $\mathcal{F}_3^2$. После ответа на него с подстановкой $\{x \to e, y \to e\}$ к базе $\mathcal{F}_3^2$ добавится атом $C(e, f(e))$.

На пятом шаге используется вновь добавленный атом для ответа на целевой вопрос Q_2 с подстановкой $\{x \to e; y \to f(e)\}$, что опровергает базу $\mathcal{F}_3^2$ и завершает вывод, так как все базы были опровергнуты.

1.2. Логическая ДСС как генератор формального языка

Рассмотрим логическую дискретно-событийную систему (ДСС), описывающую функционирование некоторой реальной системы на абстрактно-символическом уровне последовательностями событий, происходящих в неопределенные моменты времени. Такая ДСС может быть представлена [4] конечным автоматом $\boldsymbol{G} = (\boldsymbol{Q}, \Sigma, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{Q_m})$, играющим роль генератора регулярного языка. Здесь $\boldsymbol{Q}$ — множество состояний $\boldsymbol{q}$; Σ — множество событий; $\boldsymbol{\delta}$: $\Sigma \times \boldsymbol{Q} \to \boldsymbol{Q}$ — функция перехода; $\boldsymbol{q_0} \in \boldsymbol{Q}$ — начальное состояние; $\boldsymbol{Q_m} \subset \boldsymbol{Q}$ — множество выделенных состояний. $\boldsymbol{G}$ также называется *объектом управления* в теории автоматического управления. Пусть Σ^* обозначает замыкание Клини, $\boldsymbol{\varepsilon}$ — пустая строка. $\boldsymbol{\delta}$ легко распространяется на строки из Σ^*. Язык, генерируемый $\boldsymbol{G}$ — это $\boldsymbol{L(G)} = \{\boldsymbol{s} : \boldsymbol{s} \in \Sigma^*$ и $\boldsymbol{\delta(s, q_0)}$ определен$\}$, а язык, маркируемый $\boldsymbol{G}$, это – $\boldsymbol{L_m(G)} = \{\boldsymbol{s} : \boldsymbol{s} \in \boldsymbol{L(G)}$ и $\boldsymbol{\delta(s, q_0)} \in \boldsymbol{Q_m}\}$. Маркировка необходима для того, чтобы выделить строки, которые в системе считаются «завершенными», например, окончание процесса сборки на производственной линии или завершение последовательности действий, формирующих миссию мобильного робота. При этом элементы множества $\boldsymbol{Q_m}$ не рассматриваются как терминальные состояния. В связи с этим к генераторам также применяется термин «машина конечных состояний». Однако, начиная с основополагающей работы [4], термин «автомат» стал стандартным термином, используемым в ТСУ. Обозначим через $\overline{\boldsymbol{K}}$ множество всех префиксов слов из $\boldsymbol{K}$. Язык $\boldsymbol{K}$ называется префиксно-замкнутым, если $\overline{\boldsymbol{K}} = \boldsymbol{K}$. Для любого генератора $\boldsymbol{G}$ справедливо $\boldsymbol{L(G)} = \overline{\boldsymbol{L(G)}}$.

ТСУ предполагает, что возникновение некоторых событий в системе может быть запрещено внешним средством управления, называемым супервизором. Пусть $\Sigma = \Sigma_c \cup \Sigma_{uc}$, где Σ_c — множество управляемых событий, $\Sigma_{uc} = \Sigma \backslash \Sigma_c$, события в множестве Σ_{uc} являются неуправляемыми, $\Sigma_c \cap \Sigma_{uc} = \varnothing$. Чтобы система не вышла за рамки поведения, определенного неким регулярным языком, например, $\boldsymbol{K}$, супервизор запрещает возникновение событий из $\boldsymbol{\Sigma_c}$. $\boldsymbol{K}$ называется спецификацией поведения системы. Обозначим через $\boldsymbol{L(J/G)}$ язык, генерируемый ДСС $\boldsymbol{G}$ под управлением супервизора $\boldsymbol{J}$. Пусть $\boldsymbol{L_m(J/G)}$ обозначает язык, маркированный супервизором: $\boldsymbol{L_m(J/G)} = \boldsymbol{L(J/G)} \cap \boldsymbol{L_m(G)}$. Основная задача супервизорного управления состоит в построении такого супервизора $\boldsymbol{J}$, что $\boldsymbol{L(J/G)} = \overline{\boldsymbol{K}}$ и $\boldsymbol{L_m(J/G)} = \boldsymbol{K}$. Второе равенство важно для задач, связанных с обеспечением неблокирования системы.

Критерием существования супервизора, обеспечивающего решение основной задачи супервизорного управления, является управляемость языка спецификации $\boldsymbol{K}$. Язык $\boldsymbol{K}$ называется *управляемым* (относительно $\boldsymbol{L(G)}$ и Σ_{uc}), если $\overline{\boldsymbol{K}}\boldsymbol{\Sigma_{uc}} \cap \boldsymbol{L(G)} \subseteq \overline{\boldsymbol{K}}$. Здесь $\overline{\boldsymbol{K}}\boldsymbol{\Sigma_{uc}}$ — сокращенное выражение, обозначающее конкатенацию всех строк из $\overline{\boldsymbol{K}}$ с любым из символов из множества $\boldsymbol{\Sigma_{uc}}$. Проверка управляемости является первым этапом процедуры построения супервизорного управления по заданной спецификации. В случае, когда управляемость не имеет места, можно построить гарантированно управляемый подъязык спецификации, обеспечивающий наименьшее ограничение возможностей функционирования системы. Доказано, что наибольший управляемый подъязык $\boldsymbol{K}^{\uparrow C}$ языка $\boldsymbol{K}$ всегда существует. Если построенный подъязык оказывается удовлетворительным ограничением на систему, то в качестве новой спецификации выбирается $\boldsymbol{K}^{\uparrow C}$, и для его обеспечения строится соответствующее супервизорное управление. Ниже будет показано, как логический

вывод ПОФ применяется для анализа управляемости, построения минимально ограничивающих спецификаций и реализации супервизорного управления.

2. Супервизорное управление с помощью ПОФ

2.1. Представление ДСС в виде ПОФ

Для представления некоторой ДСС G с помощью ПОФ-исчисления мы используем следующие предикаты. Пусть предикат $L(s, q)$ обозначает, что «последовательность событий s приводит систему в состояние q». Аналогично, предикат $L^m(s, q)$ обозначает факт «последовательность событий s приводит систему в маркированное состояние q». Первые аргументы этих предикатов будут накапливать строки языков, генерируемого и маркированного автоматом G. Пусть предикат $\delta(q_1, \sigma, q_2)$ интерпретируется как переход из состояния q_1 в состояние q_2 по событию σ. Аналогично, предикат $\delta^m(q_1, \sigma, q_2)$ соответствует переходу в маркированное состояние q_2. Как обычно, функциональный символ «$\cdot$» обозначает конкатенацию строк, а символ «ε» соответствует пустой строке.

$$\exists\, B_G \left\langle \begin{array}{ll} \forall s, q, \sigma, q'\ L(\sigma, s), \delta(q, \sigma, q') & \text{——} \ \exists L(s \cdot \sigma, q') \\ \forall s, q, \sigma, q'\ L(\sigma, s), \delta_m(q, \sigma, q') & \text{——} \ \exists L^m(s \cdot \sigma, q') \end{array} \right.$$

Fig. 1. PCF $\mathcal{F}_G$ generating $L(G)$ and $L_m(G)$ for DES G

Рис. 1. Общий вид ПОФ $\mathcal{F}_G$, строящей $L(G)$ и $L_m(G)$ для ДСС G

Если дан генератор G, процесс построения языков $L(G)$ и $L_m(G)$ можно реализовать с помощью ПОФ $\mathcal{F}_G$ на рис. 1. Ее базой является множество $B_G = \{L(\varepsilon, S_0),\ L_m(\varepsilon, S_0),\ \delta(S_1^i, \sigma^i, S_2^i),\ \delta^m(S_1^j, \sigma^j, S_2^j)\}$, $i, j \in \{1, \ldots, n\}$, где n — количество событий G. База содержит атомы, описывающие переходы между состояниями G, включая маркированные, и начальные атомы $L(\varepsilon, S_0)$, $L_m(\varepsilon, S_0)$, которые будут использоваться для построения языков G. В зависимости от выбранного на текущем шаге вывода вопроса использование правила вывода ω ПОФ-исчисления влечет за собой генерацию нового факта $L(s, q)$ или $L^m(s, q)$ и добавление его в базу, причем аргументом s является слово языка $L(G)$ или $L_m(G)$ соответственно. Для различных задач ТСУ управляемые и неуправляемые события будут представлены атомами, которые являются предикатами $\Sigma_c(\_)$ и $\Sigma_{uc}(\_)$ соответственно. Вся доступная информация о системе может храниться в базе в виде фактов для дальнейшего использования. Расширенной мы называем базу, содержащую информацию, которая не является необходимой для вывода текущей формулы.

Пример 2. Рассмотрим ДСС G на рис. 2. Пусть $\Sigma_{uc} = \{a_1, a_2, a_3, a_4\}$. Для этого автомата общая форма ПОФ $\mathcal{F}_G$ на рис. 1 имеет расширенную базу, представленную множеством $B_G = \{Q_0(A), Q(A), Q(B), Q(C), Q(D), Q(E), \Sigma_{uc}(a_i), L(\varepsilon, A), L^m(\varepsilon, A), \delta(A, a_1, B), \delta(B, p_3, A), \ldots, \delta^m(C, p_2, A), \delta^m(B, p_3, A), \ldots\}$, $i = \overline{1, 4}$.

Следует заметить, что в ПОФ $\mathcal{F}_G$ отсутствует целевой вопрос. Отсутствие целевого вопроса в ПОФ означает ее выполнимость, т. е. такая ПОФ никогда не может быть опровергнута. В этом случае вывод завершается только из-за исчерпания подстановок, позволяющих применить правило вывода ω. В зависимости от рассматриваемой задачи ТСУ нас может интересовать построение вывода формулы без цели ее опровержения. При выводе $\mathcal{F}_G$, если это позволяет атом $\delta(q_1^i, \sigma^i, q_2^i)$, с помощью функционального символа $\cdot$ создается новый атом, который добавляется к базе на каждом этапе вывода, поэтому невозможно исчерпать все возможные подстановки. Таким образом, бесконечность вывода позволяет реализовать ДСС как генератор регулярного языка, включающего слова любой длины. Если атомы $\delta(q_1^i, \sigma^i, q_2^i)$ не допускают никаких подстановок, это означает, что все слова $L(G)$, $L_m(G)$ уже построены.

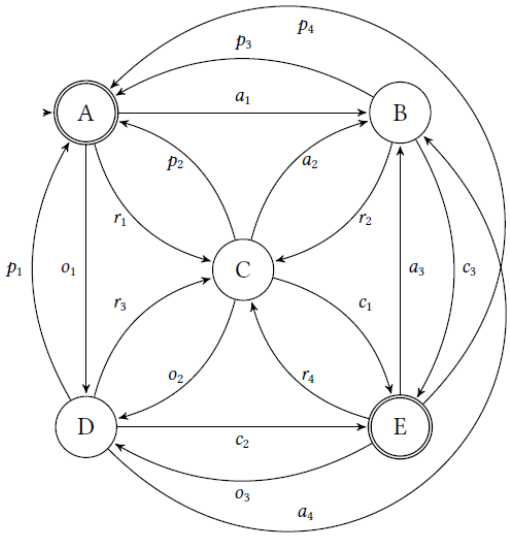


Fig. 2. Generator G **Рис. 2.** Генератор G

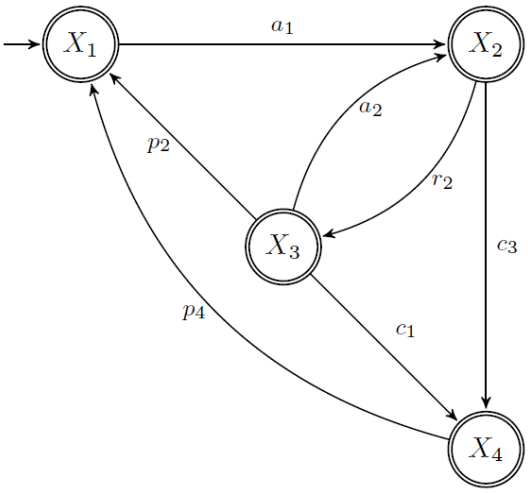


Fig. 3. Automaton H recognizing specification language K **Рис. 3.** Автомат H, распознающий язык спецификации K

Далее в данной статье для простоты изложения не рассматриваются маркированный язык и связанные с ним задачи, такие как проверка блокирования или построение неблокирующего супервизорного управления.

2.2. Анализ языка спецификации на ДСС

Пусть спецификация языка K на ДСС G распознается автоматом H. Как говорилось ранее, критерием существования супервизорного управления является управляемость K. Согласно парадигме АДТ, для проверки некоторого свойства ищется опровержение отрицания этого свойства. Таким образом, будем доказывать отсутствие неуправляемости спецификации K, т. е. что возникновение неуправляемого события не приводит к строке, которая принадлежит языку $L(G)$, но не является префиксом слова из K. Для этого воспользуемся операцией произведения автоматов, соответствующих системе и спецификации $G \times H$, поскольку такой автомат генерирует только строки, общие для $L(G)$ и K, $L(G \times H) = L(G) \cap K$.

Рассмотрим ПОФ $\mathcal{F}_{Ctl\_ch}$, реализующую проверку управляемости спецификации K:

$$\mathcal{F}_{Ctl\_ch} = \exists B_{Ctl\_ch} \begin{cases} R_1 \\ R_2 \\ \dots \\ R_6, \end{cases}$$

$$R_1 : \ \forall q_1^1, q_2^1, q_1^3, q_2^3, \sigma \ Q_0^1(q_1^1), Q_0^3(q_1^3), \delta_1(q_1^1, \sigma, q_2^1), \delta_3(q_1^3, \sigma, q_2^3)^- \exists \ N(q_1^1, q_1^3),$$
$$R_2 : \ \forall q_1^1, q_2^1, q^3, \sigma \ N(q_1^1, q^3), \delta_1(q_1^1, \sigma, q_2^1), E_{uc}(\sigma)^- \exists \ Chk(q^3, \sigma, 0),$$
$$R_3 : \ \forall q_1^3, q_2^3, \sigma \ Chk^*(q_1^3, \sigma, 0), \delta_3(q_1^3, \sigma, q_2^3)^- \exists \ Chk(q_1^3, \sigma, 1),$$
$$R_4 : \ \forall q^3, \sigma \ Chk(q^3, \sigma, 0) - \exists \ UC(q^3, \sigma),$$
$$R_5 : \forall p, \sigma \ UC(p, \sigma),$$
$$R_6 : \ \forall q_1^1, q_2^1, q_1^3, q_2^3, \sigma \ N(q_1^1, q_1^3), \delta_1(q_1^1, \sigma, q_2^1), \delta_3(q_1^3, \sigma, q_2^3)^- \exists \ N(q_2^1, q_2^3).$$

Предполагается, что база B_{Ctl_ch} ПОФ $\mathcal{F}_{Ctl\_ch}$ содержит предикатные описания переходов $\boldsymbol{G}$, $\boldsymbol{H}$ и $\boldsymbol{G} \times \boldsymbol{H}$. Заметим, что синхронную композицию автоматов и их произведение можно построить и с использованием подхода на основе ПОФ. В правилах $R_1 - R_6$ используются вспомогательные предикаты. Предикат $N(\_, \_)$ будет хранить пары состояний, которые одновременно достигаются из состояний $\boldsymbol{G}$ и $\boldsymbol{G}\times\boldsymbol{H}$ по одному и тому же событию. Такие состояния называются *соседними* состояниями. Остальные новые предикаты будут объясняться по мере их появления. Правило R_1 добавляет в базу начальные состояния $\boldsymbol{G}$ и $\boldsymbol{G} \times \boldsymbol{H}$, которые являются стартовыми для процедуры проверки нарушения управляемости. Правило R_2 обнаруживает в $\boldsymbol{G}$ переход по неуправляемому событию $\boldsymbol{\sigma}$. Если ответ на этот вопрос успешен, то в базу добавляется атом $Chk(q^3, \sigma, 0)$ для проверки состояния q^3 автомата $\boldsymbol{G} \times \boldsymbol{H}$ на существование в нем, а значит и в $\boldsymbol{H}$, перехода по событию $\boldsymbol{\sigma}$. Наличие такого перехода означает, что условие управляемости не нарушается, и флаг 0 в $Chk(q^3, \sigma, 0)$ меняется на 1. Затем вопрос R_6 используется для добавления следующей пары проверяемых состояний. Если флаг в $Chk(q^3, \sigma, 0)$ остаётся 0, то имеет место нарушение управляемости, и вопросом R_4 в базу добавляется атом $UC(p, \sigma)$, который позволяет ответить на целевой вопрос R_5. Если вывод завершился исчерпанием вариантов поиска замен и ответ на целевой вопрос не найден, то язык спецификации является управляемым. Мы формализовали задачу таким образом, что если спецификация неуправляема, то успешное завершение вывода обеспечивает набор переменных, нарушающих управляемость спецификации, в рамках ответа $UC(p, \sigma)$ на вопрос R_5. Поскольку рассматриваются только конечные автоматы, пространство поиска вывода в этой формализации является конечным. Следовательно, вывод заканчивается как в случае управляемости спецификации, так и в случае ее неуправляемости.

В случае нарушения управляемости спецификации нас интересует наибольший управляемый подъязык $\boldsymbol{K}^{\uparrow C}$, который можно выбрать в качестве альтернативной спецификации. Предложена ПОФ $\mathcal{F}_{Sub}$, обеспечивающая построение $\boldsymbol{K}^{\uparrow C}$ в процессе проверки управляемости спецификации $\boldsymbol{K}$, таким образом позволяя объединить анализ спецификаций и построение супервизора:

$$\mathcal{F}_{Sub} = \exists B_{Sub} \left\langle \begin{array}{l} R_1 \\ R_2 \\ \dots \\ R_7, \end{array} \right.$$

$$R_4 : \ \forall q^3, \sigma \ Chk(q^3, \sigma, 0) - \exists \ Del(q^3),$$
$$R_5 : \ \forall q_1^1, q_2^1, q_1^3, q_2^3, \sigma \ N(q_1^1, q_1^3), \delta_1(q_1^1, \sigma, q_2^1), \delta_3(q_1^3, \sigma, q_2^3)^- \exists \ N(q_2^1, q_2^3),$$
$$R_6 : \ \forall q_{11}, q_{12}, q_{21}, q_{22}, \sigma \ Del(q_{11} \cdot q_{12}), \delta_2^*(q_{12}, \sigma, q_{22}), \delta_3^*(q_{11} \cdot q_{12}, \sigma, q_{21} \cdot q_{22})^- \exists \ Deleted(q_{12}, \sigma, q_{22}),$$
$$R_7 : \ \forall q_{11}, q_{12}, q_{21}, q_{22}, \sigma \ Del(q_{21} \cdot q_{22}), \delta_2^*(q_{12}, \sigma, q_{22}), \delta_3^*(q_{11} \cdot q_{12}, \sigma, q_{21} \cdot q_{22})^- \exists \ Deleted(q_{12}, \sigma, q_{22}).$$

База B_{Sub} совпадает с базой B_{Ctl_ch}, и теми же остаются правила R_1–R_3. Вопрос R_5 используется для проверки следующего состояния $\boldsymbol{G}$. Если флаг в $Chk(q^3, \sigma, 0)$ остаётся 0, то вспомогательный предикат $Del()$ в вопросе R_4 запускает процесс удаления переходов, нарушающих управляемость спецификации, с помощью вопросов R_6 и R_7. Затем процесс повторяется до тех пор, пока поиск вывода не остановится, исчерпав все возможные подстановки. В результате вывода база ПОФ будет содержать атомы, описывающие генератор $\boldsymbol{H'}$ наибольшего управляемого подъязыка $\boldsymbol{K}^{\uparrow C}$ спецификации $\boldsymbol{K}$.

Одной из существенных особенностей исчисления ПОФ, которая будет использоваться для анализа ДСС, является тот факт, что мы можем построить *немонотонный* вывод, слегка изменив определение правила вывода. Для этого введем оператор *, которым могут быть помечены атомы в вопросах. Теперь, если на вопрос с атомами, отмеченными оператором *, есть ответ, то после применения правила вывода атомы в базе, участвовавшие в поиске совпадений с отмеченными

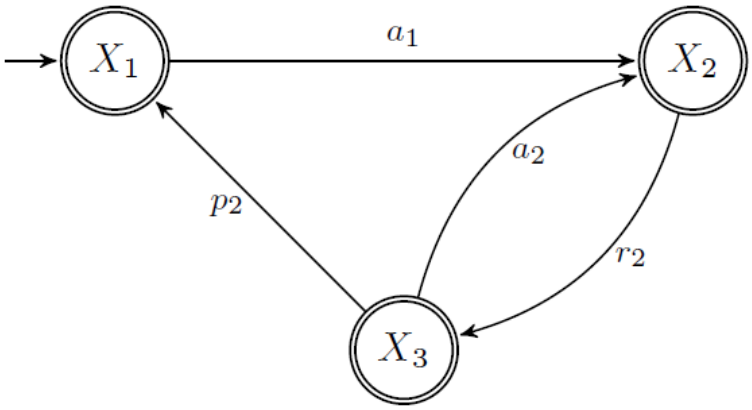


Fig. 4. Automaton H' recognizing $K^{\uparrow C}$ **Рис. 4.** Автомат H', распознающий $K^{\uparrow C}$

атомами, должны быть удалены из базы. В целом оператор * влияет на свойство полноты ПОФ-исчисления, но для рассматриваемых в статье задач благодаря правильной формализации вывод с использованием * всегда корректен.

Чтобы построить $K^{\uparrow C}$, мы используем следующую стратегию. Вывод строится путем проверки применимости правила вывода к вопросам в порядке от R_1 до R_7. Если при применении правил $R_1 - R_5$ был найден ответ на вопрос R_4, к базе добавляется атом $Del()$, обозначающий неуправляемость K. Затем применяем правила R_6, R_7, игнорируя правила $R_1 - R_5$, пока не будут исчерпаны возможные замены для применения правила вывода к R_6 и R_7. Далее процесс повторяется до тех пор, пока вывод не остановится, исчерпав все возможные замены. Например, если для данной спецификации управляемый подъязык равен только пустой строке, то при выводе все переходы в H будут удалены. Рассмотрим эту стратегию на примере.

Пример 3. Для ДСС G на рисунке 2 рассмотрим язык спецификации K, определенный автоматом H на рисунке 3. Можно заметить, что неуправляемое событие a_3 делает $K = L(H)$ неуправляемым относительно G и Σ_{uc}. Для проверки управляемости K логическими методами воспользуемся ПОФ $\mathcal{F}_{Sub}$. Для этого примера база $B_{Sub} = \{Q_0^1(A), Q_0^2(X_1), Q_0^3(A \cdot X_1), \Sigma_{uc}(a_i), \delta_1(A, a_1, B), \ldots, \delta_2(X_1, a_1, X_2), \ldots, \delta_3(A \cdot X_1, a_1, B \cdot X_2), \ldots\}$ включает группу атомов $\delta_1(\_, \_, \_)$, соответствующую переходам автомата G, группу атомов $\delta_2(\_, \_, \_)$, соответствующую переходам автомата H, и $\delta_3(\_, \_, \_)$, соответствующую переходам автомата-произведения $G \times H$. В таблице 1 показаны шаги вывода, строящего автомат H'. Таблица иллюстрирует минимальный вывод, построенный прувером Bootfrost, разработанным для опровержения ПОФ. Вывод был построен за 17 шагов. В таблице не показаны шаги, которые добавляют уже отмеченные для проверки пары состояний, поглощаются текущей базой, и те, которые приводят к успешной проверке управляемости. Заметим, что в этом примере вывод прекращается из-за исчерпания всех возможных подстановок. После окончания вывода переходы H' можно извлечь из базы $\mathcal{F}_{Sub}$ как атомы $\delta_2(\_, \_, \_)$. Получившийся автомат показан на рисунке 4.

2.3. Реализация супервизорного управления

Если H — автомат, распознающий управляемый язык спецификации K, то H можно использовать в качестве супервизора J для G и справедливо равенство $L(J/G) = J||G$ [5]. Таким образом, операция построения синхронной композиции автоматов может быть использована для реализации супервизорного управления. Поскольку множества событий H и G совпадают, вместо $H||G$ можно использовать операцию произведения $H \times G$. Заметим, что если спецификация K не является управляемой, но $K^{\uparrow C} \neq \emptyset$, то язык $K^{\uparrow C}$ может рассматриваться как новая, наименее ограничительная спецификация, а автомат H', распознающий $K^{\uparrow C}$, может использоваться в качестве супервизора J.

Table 1. Constructive inference providing H' as a recognizer of $K^{\uparrow C}$

Таблица 1. Конструктивный вывод, строящий H' как распознаватель $K^{\uparrow C}$

Шаг	Использованные атомы	Ответ	Добавленные атомы
1	$Q_0^1(A), Q_0^3(A \cdot X_1)$, $\delta_1(A, a_1, B)$, $\delta_3(A \cdot X_1, a1, B \cdot X_2)$	$\{q_1^1 \to A, q_2^1 \to B, q_1^3 \to A \cdot X_1$, $q_2^3 \to B \cdot X_2, \sigma \to a_1\}$	$N(A, A \cdot X_1)$
На первом этапе поиска вывода был найден ответ на вопрос R_1, что означает, что пара состояний A и $A \cdot X_1$ в автоматах G и $G \times H$ подлежит проверке на нарушение управляемости.			
2	$N(A, A \cdot X_1), \delta_1(A, a_1, B)$, $\Sigma_{uc}(a_1)$	$\{q_1^1 \to A, q_1^2 \to B, q^3 \to A \cdot X_1, \sigma \to a_1\}$	$Chk(A \cdot X_1, a_1, 0)$
Ранее добавленный атом $N(A, A \cdot X_1)$ помогает найти ответ на вопрос R_2. Ответ означает, что в автомате G найдена неконтролируемый переход по событию a_1, поэтому необходима проверка соответствующего перехода в автомате $G \times H$.			
3	$Chk(A \cdot X_1, a_1, 0)(deleted)$, $\delta_3(A \cdot X_1, a_1, B \cdot X_2)$	$\{q_1^3 \to A \cdot X_1, q_2^3 \to OA \cdot X_2, \sigma \to a_1\}$	$Chk(A \cdot X_1, a_1, 1)$
На вопрос R_3 получен ответ, значит, проверка, назначенная на предыдущем шаге, пройдена, т. е. нарушение управляемости не обнаружено. Заметим, что поиск вывода дает несколько возможных ответов на вопросы R_2 и R_3. Мы опускаем дополнительные варианты, чтобы показать минимальный вывод.			
4	$N(A, A \cdot X_1), \delta_1(A, a_1, B)$, $\delta_3(A \cdot X_1, a_1, B \cdot X_2)$	$\{q_1^1 \to A, q_2^1 \to B, q_1^3 \to A \cdot X_1, q_2^3 \to$ $B \cdot X_2, \sigma \to a_1\}$	$N(B, B \cdot X_2)$
Ответ на вопрос R_5 получен. Ответ аналогичен первому шагу данного вывода и добавляет следующую пару проверяемых состояний.			
5	$N(B, B \cdot X_2), \delta_1(B, r_2, C)$, $\delta_3(B \cdot X_2, r_2, B \cdot X_2)$	$\{q_1^1 \to B, q_2^1 \to C, q_1^3 \to B \cdot X_2$, $q_2^3 \to C \cdot X_3, \sigma \to r_2\}$	$N(C, C \cdot X_3)$
Ответ на вопрос R_5 получен. В базу для последующей проверки на нарушение управляемости добавляется еще одна пара состояний.			
6	$N(B, B \cdot X_2), \delta_1(B, c_3, E)$, $\delta_3(B \cdot X_2, c_3, E \cdot X_4)$	$\{q_1^1 \to B, q_2^1 \to E, q_1^3 \to B \cdot X_2$, $q_2^3 \to E \cdot X_4, \sigma \to c_3\}$	$N(E, E \cdot X_4)$
На вопрос R_5 дан ответ. В базу добавлена пара состояний $N(E, E \cdot X_4)$. Состояние E в автомате G имеет исходящий переход, помеченный неуправляемым событием a_3.			
7	$N(E, E \cdot X_4), \delta_1(E, a_3, B), \Sigma_{uc}(a_3)$	$\{q_1^1 \to E, q_2^1 \to B, q^3 \to E \cdot X_4, \sigma \to a_3\}$	$Chk(E \cdot X_4, a_3, 0)$
Получен ответ на вопрос R_2. Результат аналогичен второму шагу этого умозаключения, но событие, которое будет проверяться следующим, — a_3 в состоянии E.			
8	$Chk(E \cdot X_4, a_3, 0)$	$\{q^3 \to E \cdot E, \sigma \to a_3\}$	$Del(E \cdot X_4)$
Ответ на вопрос R_4 означает, что проверка, назначенная на предыдущем шаге, провалена. Таким образом, спецификация, соответствующая автомату H, является неуправляемой. Правила R_6, R_7 удаления переходов, связанных с состоянием E, будут срабатывать благодаря добавляемым атомам $Del(E \cdot X_4)$.			
9	$Del(E \cdot X_4), \delta_2(X_2, c_3, X_4)$, $\delta_3(B \cdot X_2, c_3, E \cdot X_4)$	$\{q_{11} \to B, q_{12} \to X_2, q_{21} \to E$, $q_{22} \to X_4, \sigma \to c_3\}$	$Deleted(X_2, c_1, X_4)$
Ответ на вопрос R_6 получен. Переход $\delta_2(X_2, c_3, X_4)$ автомата H удален.			
10	$Del(E \cdot X_4), \delta_2(X_4, p_4, X_1)$, $\delta_3(E \cdot X_3, p_4, A \cdot X_1)$	$\{q_{11} \to E, q_{12} \to X_3, q_{21} \to A$, $q_{22} \to X_1, \sigma \to p_4\}$	$Deleted(X_4, p_4, X_1)$
Ответ на вопрос R_6 получен. Переход $\delta_2(E, p_4, A)$ автомата H удален.			
11	$Del(E \cdot X_4), \delta_2(X_3, c_1, X_4)$, $\delta_3(C \cdot X_3, c_1, E \cdot X_4)$	$\{q_{11} \to C, q_{12} \to X_3, q_{21} \to E$, $q_{22} \to X_4, \sigma \to c_1\}$	$Deleted(C, c_1, E)$
Ответ на вопрос R_7 получен. Переход $\delta_2(X_3, c_1, X_4)$ автомата H удален.			

В предлагаемом подходе ПОФ $\mathcal{F}_{SupC}$ на рисунке 5 генерирует язык $L(J/G)$. Здесь множество $B_{J/G}$ включает в себя базы B_G и B_H, а также содержит атом $L^{J/G}(\varepsilon, S_0^G)$ в качестве начального атома для конструкции языка $L(J/G)$. Также в базе находится атом $E(\sigma^{\#})$, где $\sigma^{\#}$ — это результат вызова функции $get\_random\_event()$, которая предоставляет случайное событие, которое может произойти

$$\exists\ B_{J/G} \cup \{E(\sigma^{\#})\} \text{ — } \begin{array}{r} \forall\ s, s', q, \sigma, q'\ L^{J/G}(s,q), \\ L^J(s',q), E^*(\sigma), \\ \delta_J(q,\sigma,q') \end{array} \text{ — } \begin{array}{r} \exists L^J(s\cdot\sigma, q') \\ E(\sigma^{\#}) \end{array} \text{ — } \begin{array}{r} \forall\ L^{J/G}(s',q), \\ \delta_G(q,\sigma,q') \end{array} \text{ — } \exists L^{J/G}(s'\cdot\sigma, q')$$

Fig. 5. PCF $\mathcal{F}_{SupC}$ generating $\boldsymbol{L(J/G)}$

Рис. 5. Общая форма ПОФ $\mathcal{F}_{SupC}$, генерирующей $\boldsymbol{L(J/G)}$

$$\exists\ B_{J/G} \cup \{E(\sigma^{\#}), L^J(\varepsilon\cdot a_1, B)\} \left\langle \begin{array}{l} \forall\ s, s', q, \sigma, q'\ L^{J/G}(s,q), \\ L^J(s',q), E^*(\sigma), \\ \delta_J(q,\sigma,q') \end{array} \right. \text{ — } \begin{array}{r} \exists L^J(s\cdot\sigma, q') \\ E(\sigma^{\#}) \end{array} \text{ — } \begin{array}{r} \forall\ L^{J/G}(s',q), \\ \delta_G(q,\sigma,q') \end{array} \text{ — } \exists L^{J/G}(s'\cdot\sigma, q')$$
$$\left. \begin{array}{l} \\ \forall\ L^{J/G}(\varepsilon, A), \delta_G(A, a_1, B) \end{array} \right. \text{ — } \exists L^{J/G}(\varepsilon\cdot a_1, B)$$

Fig. 6. Generation of $\boldsymbol{L(J/G)}$, step 1, event a_1 occurred

Рис. 6. Генерация $\boldsymbol{L(J/G)}$, шаг 1, произошло событие a_1

$$\exists\ B_{J/G} \cup \{E(\sigma^{\#}), L^J(\varepsilon\cdot a_1, B), L^{J/G}(\varepsilon\cdot a_1, B)\} \text{ — } \begin{array}{l} \forall\ s, s', q, \sigma, q'\ L^{J/G}(s,q), \\ L^J(s',q), E^*(\sigma), \\ \delta_J(q,\sigma,q') \end{array} \text{ — } \begin{array}{r} \exists L^J(s\cdot\sigma, q') \\ E(\sigma^{\#}) \end{array} \text{ — } \begin{array}{r} \forall\ L^{J/G}(s',q), \\ \delta_G(q,\sigma,q') \end{array} \text{ — } \exists L^{J/G}(s'\cdot\sigma, q')$$

Fig. 7. Generation of $\boldsymbol{L(J/G)}$, step 2

Рис. 7. Генерация $\boldsymbol{L(J/G)}$, шаг 2

$$\exists\ B_{J/G} \cup \{E(\sigma^{\#}), L^J(\varepsilon\cdot a_1, B), L^{J/G}(\varepsilon\cdot a_1, B), L^J(\varepsilon\cdot a_1\cdot r_2, C)\} \left\langle \begin{array}{l} \forall\ s, s', q, \sigma, q'\ L^{J/G}(s,q), \\ L^J(s',q), E^*(\sigma), \\ \delta_J(q,\sigma,q') \end{array} \right. \text{ — } \begin{array}{r} \exists L^J(s\cdot\sigma, q') \\ E(\sigma^{\#}) \end{array} \text{ — } \begin{array}{r} \forall\ L^{J/G}(s',q), \\ \delta_G(q,\sigma,q') \end{array} \text{ — } \exists L^{J/G}(s'\cdot\sigma, q')$$
$$\left. \begin{array}{l} \\ \forall\ L^{J/G}(\varepsilon\cdot a_1, B), \delta_G(B, r_2, C) \end{array} \right. \text{ — } \exists L^{J/G}(\varepsilon\cdot a_1\cdot r_2, C)$$

Fig. 8. Generation of $\boldsymbol{L(J/G)}$, step 3, event r_2 occurred

Рис. 8. Генерация $\boldsymbol{L(J/G)}$, шаг 3, произошло событие r_2

в текущем состоянии. $\delta_J(\_,\_,\_)$ соответствует переходам $\boldsymbol{H}$, играющего роль супервизора для $\boldsymbol{G}$. Поскольку $\boldsymbol{H}$ является подавтоматом $\boldsymbol{G}$, предполагается, что состояния $\boldsymbol{H}$ переименованы соответственно состояниям $\boldsymbol{G}$.

Пример 4. Рисунки 6-8 и таблица 2 представляет первые шаги вывода ПОФ $\mathcal{F}_{SupC}$ для ДСС из примера 2 с языком спецификации $\boldsymbol{K}^{\uparrow C}$, распознаваемым автоматом на рисунке 4. Текущее состояние системы определяется вторым аргументом последнего добавленного атома $L^{J/G}(\_,\_)$. Первый столбец таблицы 2 — это номер вопроса, на который ищется ответ. Следующие пять столбцов показывают части найденной подстановки, например, в первой строке представлена подстановка: $\{\sigma \to a_1, q \to A, q1 \to B, s \to \varepsilon, s' \to \varepsilon\}$. Колонка **Добавления** содержит информацию о том, что было добавлено в базу на этом этапе, атомы и/или вопросы. Колонка **Удаления** содержит информацию об удалениях, произошедших в формуле на этом этапе.

Заметим, что из формальных определений ПОФ-исчисления следует, что если у вопроса нет кванторных переменных и типовое условие содержит только константы, такой вопрос можно удалять после ответа, т. к. на него можно ответить только один раз с пустой подстановкой. Такие вопросы называются *тривиальными*. На первом шаге вывода (рисунок 6) в формулу добавляется второй вопрос, являющийся тривиальным. После ответа на него на втором шаге он удаляется из ПОФ. Аналогично происходит на третьем и последующих шагах.

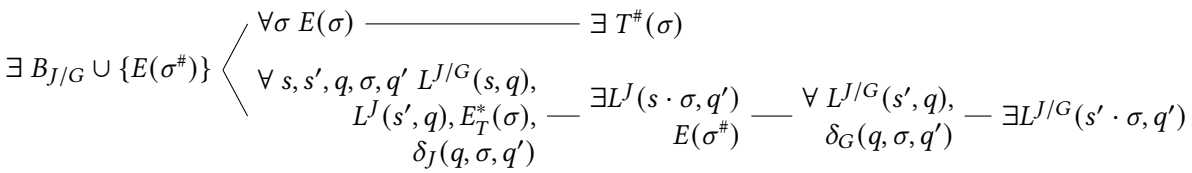


Fig. 9. PCF $\mathcal{F}_{Knl}$ realizing supervisory control using additional knowledge

Рис. 9. ПОФ $\mathcal{F}_{Knl}$, реализующая супервизорное управление с использованием дополнительных знаний

Table 2. Comments to the inference of $\mathcal{F}_{SupC}$ in Example 3

Таблица 2. Пояснения к выводу ПОФ $\mathcal{F}_{SupC}$ в примере 3

$Q\#$	σ	q	q'	s	s'	Добавления	Удаления
1	a_1	A	B	ε	ε	$L^J(\varepsilon\cdot a_1, B)$ $\forall\, L^{J/G}(\varepsilon, A), \delta_G(A, a_1, B)-$ $\exists L^{J/G}(\varepsilon\cdot a_1, B)$	$E(a_1)$
2	—	—	—	—	—	$L^{J/G}(\varepsilon\cdot a_1, B)$	вопрос 2 удален
1	r_2	B	C	a_1	a_1	$L^J(\varepsilon\cdot a_1\cdot r_2, C)$ $\forall\, L^{J/G}(\varepsilon\cdot a_1, B), \delta_G(B, r_2, C)-$ $\exists L^{J/G}(\varepsilon\cdot a_1\cdot r_2, C)$	$E(r_2)$
2	—	—	—	—	—	$L^{J/G}(\varepsilon\cdot a_1\cdot r_2, C)$	вопрос 2 удален

Из сравнения первых аргументов атомов $L^{J/G}(\_,\_)$ и $L^J(\_,\_)$ очевидно, что $\boldsymbol{L(J/G) = K^{\uparrow C}}$. Таким образом, реализовано требуемое управление, при этом не использовалось построение структуры $\boldsymbol{H'||G}$.

2.4. Обработка событий

Представление ДСС с помощью ПОФ позволяет использовать в процессе построения логического вывода информацию, поступающую из окружающей среды, а также данные о функционировании самой системы. Это можно реализовать с помощью специальных логических правил, представленных в ПОФ в виде вопросов обработки событий. Ответ на них запускает дополнительные выводы, в которых события, поступающие из окружающей среды, служат параметрами, используемыми в расчетах или других процедурах. На рисунке 9 показана ПОФ, генерирующая язык ДСС, ограниченной спецификацией, с возможностью обработки событий. Здесь $E(\sigma)$ — это атом, который определяет все события ДСС и каждый такт заменяется на реальное событие, произошедшее в системе.

В этой ПОФ оператор # — это символ, обозначающий вызов внешнего мира. Первый вопрос $\mathcal{F}_{Knl}$ служит для внешней обработки события, произошедшего в системе. Событие входит в вычислимую конструкцию $T^{\#}$ в качестве аргумента σ. $T^{\#}$ может иметь любую разумную сложность, например, это может быть некоторая логическая формула, включая ПОФ (как на рисунке 10), или любой черный ящик, выполняющий вычисления или другие процедуры обработки знаний. Атом $E^*_T(\sigma)$ в этом вопросе является результатом обработки события σ с помощью конструкции $T^{\#}$. Здесь мы используем оператор $*$ для удаления использованных атомов.

Конъюнкт *Trigger* в первом вопросе ПОФ на рисунке 10 может задавать некоторое условие. Ответ на такой заранее подготовленный вопрос вызывает процесс обработки $\sigma^{\#}$, результатом которого

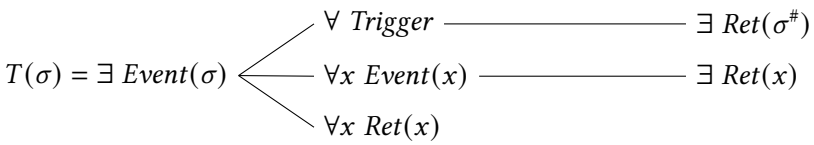


Fig. 10. PCF processing predicate T **Рис. 10.** ПОФ, обрабатывающая предикат T

будет добавление в базу атома $Ret(\sigma)$ с конкретизированным σ. Это может быть, например, «подмена» события, принудительная смена состояния или другая реконфигурация системы. Второй вопрос отрабатывает поведение системы по умолчанию, добавляя атом $Ret(\sigma)$ с тем же событием, которое поступило в подвывод, чтобы ответить на третий, целевой, вопрос и вернуть управление поиском вывода вызывающему потоку. Стратегия вывода должна быть настроена таким образом, что если ответ на первый, условный, вопрос, осуществился, то второй вопрос необходимо убрать из очереди, чтобы не добавить лишнее в данном случае необработанное событие.

3. Программная система Bootfrost, реализующая метод опровержения ПОФ

Прувер Bootfrost для исчисления ПОФ написан на языке программирования Rust. Исходный код проекта, документацию и примеры можно найти на странице GitHub[4]. Благодаря развитой системе типов Rust и модели владения гарантируется безопасность памяти и потокобезопасность. Кроме того, в Rust отсутствует среда исполнения и сборщик мусора. Эти особенности обеспечивают реализацию безопасных и эффективных систем.

Разработанный прувер основан на системе транзакций, которая позволяет регистрировать и откатывать любые изменения, происходящие в процессе поиска логического вывода. Также данная версия прувера специализирована для защищённых ПОФ. Хотя это и ограничивает класс решаемых задач, тем не менее, в прикладных задачах и задачах, связанных с динамическими системами, обычно используются только такие формулы. Защищёнными ПОФ называются такие формулы, в ти́повых условиях которых все переменные, управляемые квантором, встречаются в конъюнкте. Например, $\forall x, yA(x), B(y)$ — это защищённая подформула, поскольку переменные x, y встречаются в конъюнкте $A(x), B(y)$, при этом сами переменные также называются защищёнными, а $\forall x, yA(x), B(x)$ — это незащищённая подформула, поскольку переменная y не встречается в конъюнкте $A(x), B(x)$ и также называется незащищённой.

Стратегии, используемые в Bootfrost, делятся на три основные группы: стратегии выбора вопроса, стратегии выбора ответа и вычислимые термы и команды.

3.1. Стратегии выбора вопроса

На каждом шаге вывода прувер должен выбрать вопрос, на который он будет искать ответ. В Bootfrost для выбора вопроса используется специальная процедура. Она оценивает каждый вопрос по нескольким критериям. Эти критерии могут быть настроены пользователем, но по умолчанию используется стратегия, при которой вопрос оценивается следующим образом:

1. Является ли вопрос целевым? Целевые вопросы получают наивысшую оценку, поскольку ответ на них завершает вывод.
2. Сколько шагов вывода прошло с момента последнего ответа на вопрос? Наиболее давние вопросы получают наивысшую оценку. Такой подход обеспечивает высокий уровень разнообразия, ограничивая использование одного и того же вопроса несколько раз подряд.
3. Сколько раз на этот вопрос уже отвечали? Вопросы с наименьшим значением получают наивысшую оценку.

[4] https://github.com/snigavik/bootfrost

4. Каков уровень ветвления вопроса? Вопросы с наименьшим значением оцениваются выше.

Эта общая стратегия сортирует вопросы в порядке убывания баллов от самого высокого до самого низкого. Затем специальный метод пытается ответить на вопросы из списка, используя стратегию выбора ответа (см. следующий подраздел). Если ответ найден, то формула преобразуется по правилу вывода ω, и осуществляется переход к следующему шагу.

3.2. Стратегии выбора ответа

В прувере Bootfrost существует две стратегии выбора ответа: «Первый подходящий ответ» и «Лучший ответ».

Стратегия «Первый подходящий ответ» выбирает первый найденный ответ, удовлетворяющий заданному критерию. Такой подход позволяет эффективно использовать ресурсы памяти и процессора. Критерии могут быть настроены, по умолчанию используется тривиальная функция, возвращающая *true*, что приводит к выбору первого найденного ответа.

Стратегия «Лучший ответ» выбирает лучший ответ из множества всех возможных ответов. То есть эта стратегия находит все возможные ответы, а затем выбирает лучший из них в соответствии со специальной функцией выбора, которая может быть настроена пользователем. По умолчанию, функция просто выбирает ответ с наименьшим суммарным весом всех задействованных термов. Вес терма — это количество узлов в дереве, представляющем терм, например, терм $A(e)$ имеет вес 2, а терм $A(e, f(e))$ имеет вес 4.

Другой вспомогательной процедурой, используемой при поиске ответа, является выбор начальной точки. Существует два варианта того, с чего начинать поиск: «от последнего» и «с нуля». Метод «от последнего» означает, что следующий поиск ответа на вопрос будет начинаться в той точке, где прувер остановился во время предыдущей процедуры поиска. Метод «с нуля» означает, что процедура поиска ответа начинается с нуля, т. е. без учета предыдущих результатов поиска. Такой подход применим для немонотонных выводов и в аналогичных ситуациях, когда предыдущая история вывода может меняться с течением времени и становиться неактуальной. Например, метод «с нуля» используется для построения подъязыка $\boldsymbol{K}^{\uparrow C}$ спецификации. В ПОФ $\mathcal{F}_{Sub}$, обеспечивающей построение подъязыка, используется специальный оператор *. Если вопрос с атомом, помеченным оператором *, имеет ответ, то после применения правила вывода из базы должны быть удалены те атомы, которые участвовали в поиске подстановки с помеченным атомом. Оператор * также может быть смоделирован командой *remove-fact* (см. ниже).

3.3. Вычислимые термы и команды

Вычислимые термы (ВТермы) — это термы, обрабатываемые прувером не как синтаксические структуры, а как функции, которые должны быть вычислены. На данный момент в прувере реализованы такие ВТермы, как арифметические операции, операции сравнения, операции со списком (*in*, *notin*, *first*, *last*, конкатенация, *length*), решения (*solve*). Основная схема вычисления термов такова: специальная процедура извлекает из среды термы по их ID; затем эти термы исполняются; затем для результирующего терма, используя структуру идеального разделения (perfect sharing structure), вычисляется ID и этот ID возвращается в качестве результата.

Команды — это вычислимые термы, которые выполняются сразу после применения правила вывода ω. Например, *remove-fact* является командой. ВТермы и команды относятся к категории стратегий, поскольку их основное назначение — модификация процесса поиска логических выводов и расширение логических возможностей ПОФ. С помощью команд можно моделировать немонотонный вывод.

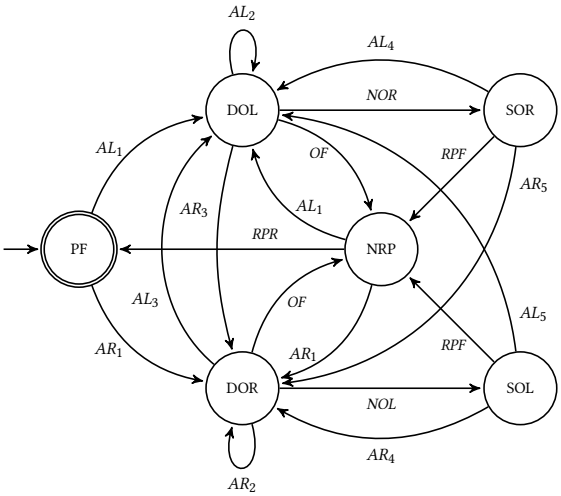


Fig. 11. Generator $\mathcal{G}$

Рис. 11. Генератор $\mathcal{G}$

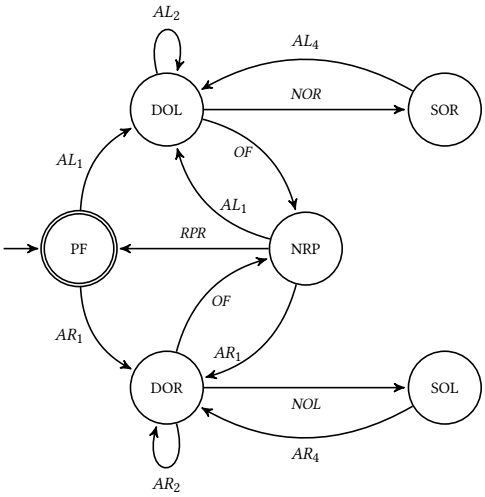


Fig. 12. Recogniser $\mathcal{H}$ of the language K

Рис. 12. Распознаватель $\mathcal{H}$ языка K

3.4. Оценка сложности

Скажем несколько слов о сложности предложенных алгоритмов на примере процедуры построения наибольшего управляемого подъязыка при проверке управляемости языка спецификации. Ответы на вопрос R_2 зависят от количества $N(\_, \_)$, добавляемых в базу. Они добавляются один раз при ответе на R_1 и столько раз, сколько переходов в произведении при ответе на R_5. Таким образом, получается не более $D+1$ копий $N(\_, \_)$. В наихудшем случае для каждого неуправляемого события будет найден ответ на R_2 с $N(\_, \_)$, так что мы получим $(D+1) * U$ шагов вывода. Каждый успешный ответ на вопрос R_2 приводит к возможности ответить на вопрос R_3 и затем R_4, т. е. всего получается $3(D+1) * U$ шагов. Если все состояния помечены на удаление, то на вопросы R_6 и R_7 можно ответить столько раз, сколько переходов в произведении. Итого, получаем $3(D+1) * U + 2D$ шагов. Таким образом, ограничив выше D по mn и U по $|\Sigma|$, найденная верхняя граница совпадает с известной оценкой $O(mn|\Sigma|)$ [5], где m и n — количество состояний автоматов $\boldsymbol{G}$ и $\boldsymbol{H}$, соответственно.

4. Иллюстративный пример

В качестве примера рассмотрим ДСС, описывающую задачу планирования пути автономного необитаемого подводного аппарата (АНПА) в неизвестной среде [50]. Предположим, что АНПА должен следовать по заданному эталонному пути, покидая его во избежание столкновения с встреченными препятствиями и возвращаясь к нему после выполнения маневров уклонения (рисунок 11). ДСС имеет следующие состояния, соответствующие режимам работы аппарата: *PF* (следование по опорному пути), *DOL* (обход обнаруженного препятствия с левой стороны), *DOR* (обход обнаруженного препятствия с правой стороны), *NRP* (навигация по опорному пути), *SOL* (поиск препятствия слева, *SOR* (поиск препятствия справа). Состояние *PF* маркировано, чтобы показать, что АНПА должен всегда возвращаться к следованию по опорному пути. События ДСС: *NOR* — «препятствий справа нет», *OF* — «обнаруженное препятствие находится далеко», *RPR* — «аппарат достиг эталонного пути» и наборы событий AL_i, AR_i, $i = \overline{1,5}$, обозначающие переключение в режимы обхода препятствий из других режимов.

Пусть спецификацию задает автомат, изображенный на рисунке 12. Стратегию поведения АНПА, предусмотренную языком спецификации $\boldsymbol{K}$, можно выразить так: стараться не попасть в место, откуда невозможно выбраться, используя стандартное препятствие. алгоритмы уклонения; не менять однажды выбранное направление обхода препятствия, пока АНПА не вернется к исходному

Table 3. DES events **Таблица 3.** События ДСС

Имя	Условие возникновения	Описание
eOLN	$R^L_{\max} < R_{\min}$	Препятствие, обнаруженное слева, находится близко
eOLNF	$R_{\min} \leqslant R^L_{\max} < R_{\min} + \Delta R$	Препятствие, обнаруженное слева, находится недалеко
eORN	$R^R_{\max} < R_{\min}$	Препятствие, обнаруженное справа, находится близко
eORNF	$R_{\min} \leqslant R^R_{\max} < R_{\min} + \Delta R$	Препятствие, обнаруженное справа, находится недалеко
eNOR	$R^R_{\max} = \infty$	Справа нет препятствий
eON	$\rho_{\min} < \rho_n$	Обнаруженное препятствие находится близко

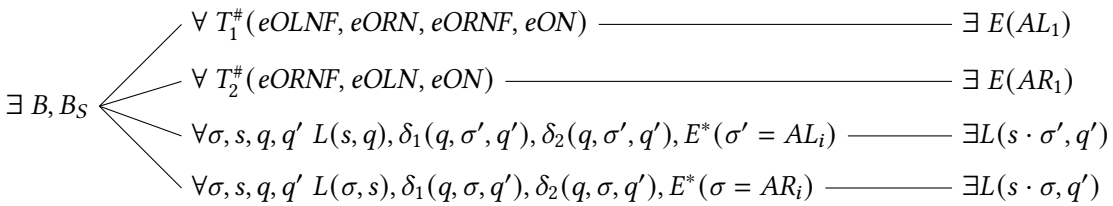


Fig. 13. PCF realizing supervisory control with composite events

Рис. 13. ПОФ, реализующая супервизорное управление с учетом композитных событий

пути (правило левой или правой руки); перестраивать путь только если это жизненно необходимо. События *RPR*, *NOR*, *NOD*, *OF* неуправляемы. Прувер Bootfrost проверяет управляемость $\boldsymbol{K}$ за 22 шага. Заметим, что любое изменение спецификации требует повторной ее проверки на управляемость, а в случае отсутствия таковой — выявления причин нарушения управляемости и построение наибольшего допустимого подъязыка спецификации, что может быть произведено с помощью представленных выше ПОФ.

После того как доказано, что спецификация управляема, можно реализовать супервизор как распознаватель языка $\boldsymbol{K}$. Его управляющее воздействие затем реализуется путем параллельной композиции автоматов, соответствующих объекту управления и супервизору. На рисунке 13 показана ПОФ, представляющая реализацию спецификации $\boldsymbol{K}$ на рисунке 12 для ДСС на рисунке 11. Здесь база представляет собой множество $\{I_1(PF), I_2(PF), \Sigma_{uc}(OF), \Sigma_{uc}(NOR), \Sigma_{uc}(NOL)\ , \Sigma_{uc}(RPR), \delta_1(PF, AL_1, DOL), \delta_1(PF, AR_1, DOR), \ldots, \delta_2(PF, AL_1, DOL)), \delta_2(PF, AR_1, DOR), \ldots\}$, состоящее из атомов объекта (множество B) и супервизора (множество B_S), и мы опускаем некоторые очевидные атомы.

Некоторые из событий AL_i, AR_i являются композиционными, т. е. определяются несколькими атомарными событиями, определяемыми условиями окружающей среды. Для учета этой информации используются предикаты T_i. Например, AL_1 как переход в режим обхода препятствия слева является результатом события *eOLNF* с одним из событий *eORN*, *eORNF* или *eON*, где *eOLNF*, *eORN*, *eORNF*, *eON* описаны в таблице 3.

Таким образом, события AL_i, AR_i могут произойти, т. е. появиться в следующих вопросах ПОФ, только в том случае, если выполняются условия их срабатывания. Оператор *, использованный с предикатом E в последних двух вопросах, показывает, что произошедшее событие должно быть удалено из базы, чтобы исключить его повторное использование машиной поиска вывода.

На рисунке 14 представлена схема реализации супервизорного управления в составе иерархической системы управления группой мобильных роботов. Система управления разрабатывается на базе роботизированного стенда, состоящего из роботов Lego Mindstorms EV3, перемещающихся по специальному полю, и набора камер, считывающих положение роботов и других объектов на поле. Программное обеспечение распространяется по беспроводной сети между роботами и сервером.

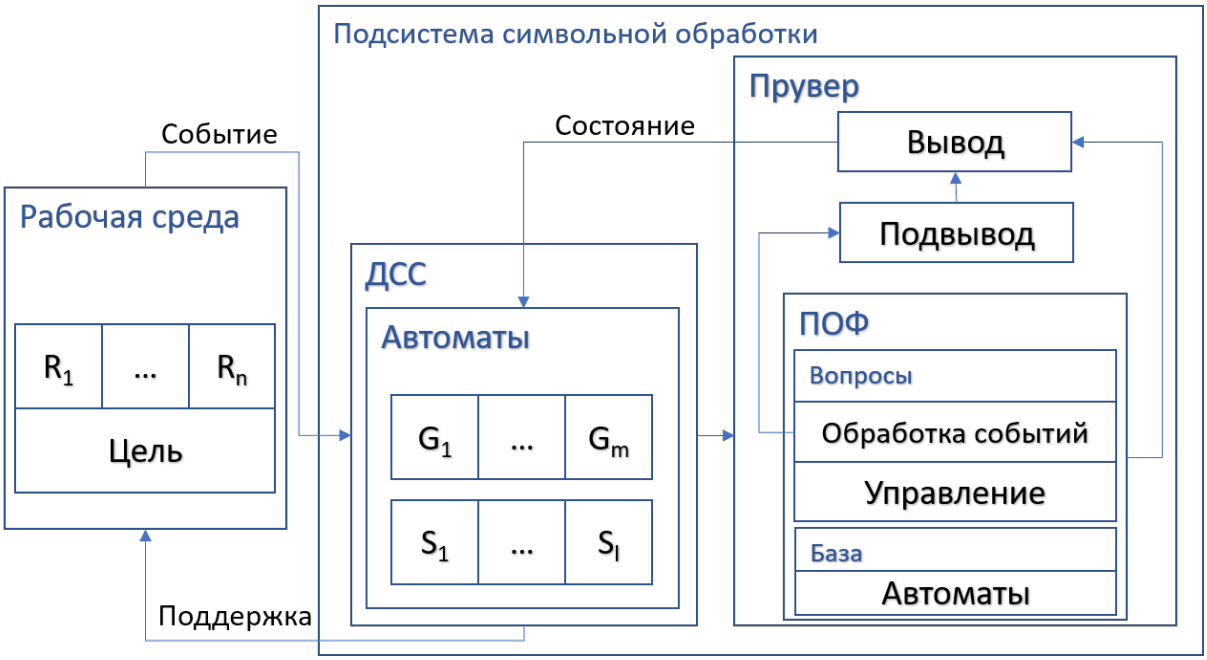


Fig. 14. Implementation of supervisory control in control system

Рис. 14. Реализация супервизорного управления как часть системы управления

На рисунке 14 реальная рабочая среда роботов представлена блоком «Рабочая среда». События, происходящие в рабочей среде, распознаются, маркируются заранее заданными символами и подаются в подсистему символьной обработки данных.

Заключение

В статье был представлен разработанный подход к решению основных задач теории супервизорного управления логическими ДСС с помощью представления таких систем в виде ПОФ и применения метода опровержения полученных ПОФ. Логический вывод в исчислении ПОФ осуществляется программной системой Bootfrost. Областью применения разработанного подхода являются системы управления техническими комплексами, в первую очередь робототехническими. Дальнейшие исследования будут направлены на решение задач управления децентрализованными и распределенными системами, в том числе частично наблюдаемыми ДСС, а также на реализацию описанного подхода в современных робототехнических системах.

References

[1] S. Lafortune, “Discrete event systems: Modeling, observation, and control”, *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 141–159, 2019. DOI: 10.1146/annurev-control-053018-023659.

[2] C. Seatzu, M. Silva, and J. H. Van Schuppen, Eds., *Control of discrete-event systems*. Springer London, 2013, 480 pp. DOI: 10.1007/978-1-4471-4276-8.

[3] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*. Springer International Publishing, 2019, 487 pp.

[4] P. J. Ramadge and W. M. Wonham, “Supervisory control of a class of discrete event processes”, *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987. DOI: 10.1137/0325013.

[5] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer Cham, 2021, 804 pp. DOI: 10.1007/978-3-030-72274-6.

[6] W. M. Wonham, K. Cai, and K. Rudie, "Supervisory control of discrete-event systems: A brief history", *Annual Reviews in Control*, vol. 45, pp. 250–256, 2018. DOI: 10.1016/j.arcontrol.2018.03.002.

[7] A. Jayasiri, G. Mann, and R. Gosine, "Behavior coordination of mobile robotics using supervisory control of fuzzy discrete event systems", *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 41, no. 5, pp. 1224–1238, 2011.

[8] C. R. Torrico, A. B. Leal, and A. T. Watanabe, "Modeling and supervisory control of mobile robots: A case of a sumo robot", *IFAC-Papers OnLine*, vol. 49, no. 32, pp. 240–245, 2016.

[9] X. Dai, L. Jiang, and Y. Zhao, "Cooperative exploration based on supervisory control of multi-robot systems", *Applied Intelligence*, vol. 45, no. 1, pp. 18–29, 2016.

[10] A. Tsalatsanis, A. Yalcin, and K. P. Valavanis, "Dynamic task allocation in cooperative robot teams", *Robotica*, vol. 30, no. 5, pp. 721–730, 2012. DOI: 10.1017/S0263574711000920.

[11] R. C. Hill and S. Lafortune, "Scaling the formal synthesis of supervisory control software for multiple robot systems", in *2017 American Control Conference (ACC)*, 2017, pp. 3840–3847. DOI: 10.23919/ACC.2017.7963543.

[12] G. W. Gamage, G. K. I. Mann, and R. G. Gosine, "Discrete event systems based formation control framework to coordinate multiple nonholonomic mobile robots", in *Proceedings of the 2009 IEEE RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 4831–4836.

[13] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, "Supervisory control theory applied to swarm robotics", *Swarm Intelligence*, vol. 10, no. 1, pp. 65–97, 2016.

[14] F. J. Mendiburu, M. R. Morais, and A. M. Lima, "Behavior coordination in multi-robot systems", in *2016 IEEE International Conference on Automatica (ICA-ACCA)*, IEEE, 2016, pp. 1–7.

[15] T. Hales *et al.*, "A formal proof of the Kepler conjecture", in *Forum of mathematics, Pi*, Cambridge University Press, vol. 5, 2017, e2. DOI: doi:10.1017/fmp.2017.1.

[16] G. Klein *et al.*, "Sel4: Formal verification of an os kernel", in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 207–220.

[17] G. Gonthier *et al.*, "Formal proof–the four-color theorem", *Notices of the AMS*, vol. 55, no. 11, pp. 1382–1393, 2008.

[18] X. Leroy, "Formal verification of a realistic compiler", *Communications of the ACM*, vol. 52, no. 7, pp. 107–115, 2009.

[19] D. A. Kondratyev and A. V. Promsky, "The complex approach of the C-lightVer system to the automated error localization in C-programs", *Automatic Control and Computer Sciences*, vol. 54, no. 7, pp. 728–739, 2020. DOI: 10.3103/S0146411620070093.

[20] J. S. Moore, "Milestones from the pure lisp theorem prover to ACL2", *Formal Aspects of Computing*, vol. 31, no. 6, pp. 699–732, 2019. DOI: 10.1007/s00165-019-00490-3.

[21] E. Karpas and D. Magazzeni, "Automated planning for robotics", *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 417–439, 2020.

[22] Z. Zombori, J. Urban, and C. E. Brown, "Prolog technology reinforcement learning prover", in *International Joint Conference on Automated Reasoning*, Springer, 2020, pp. 489–507.

[23] M. Schader and S. Luke, "Planner-guided robot swarms", in *International Conference on Practical Applications of Agents and Multi-Agent Systems*, Springer, 2020, pp. 224–237.

[24] W. Li, A. Miyazawa, P. Ribeiro, A. Cavalcanti, J. Woodcock, and J. Timmis, "From formalised state machines to implementations of robotic controllers", in *Distributed Autonomous Robotic Systems: the 13th International Symposium*, Springer, 2018, pp. 517–529.

[25] S. N. Vassilyev, "Machine synthesis of mathematical theorems", *The Journal of Logic Programming*, vol. 9, no. 2-3, pp. 235–266, 1990. DOI: 10.1016/0743-1066(90)90042-4.

[26] A. K. Zherlov, S. N. Vassilyev, E. A. Fedosov, and B. E. Fedunov, *Intelligent control of dynamic systems*. Fizmatlit, 2000, 351 pp., In Russian.

[27] S. Vassilyev and G. Ponomarev, "Automation methods for logical derivation and their application in the control of dynamic and intelligent systems", *Proceedings of the Steklov Institute of Mathematics*, vol. 276, pp. 161–179, 2012.

[28] S. Vassilyev and A. Galyaev, "Logical-optimization approach to pursuit problems for a group of targets", *Doklady Mathematics*, vol. 95, pp. 299–304, 3 2017.

[29] F. F. Reijnen, T. R. Erens, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Supervisory controller synthesis and implementation for safety PLCs", *Discrete Event Dynamic Systems*, vol. 32, no. 1, pp. 115–141, 2022.

[30] D. Bohlender and S. Kowalewski, "Compositional verification of PLC software using horn clauses and mode abstraction", *IFAC-PapersOnLine*, vol. 51, pp. 428–433, Jan. 2018. DOI: 10.1016/j.ifacol.2018.06.336.

[31] D. Bohlender and S. Kowalewski, "Leveraging horn clause solving for compositional verification of PLC software", *Discrete Event Dynamic Systems*, vol. 30, no. 1, pp. 1–24, 2020. DOI: 10.1007/s10626-019-00296-8.

[32] N. Bjørner and L. Nachmanson, "Navigating the universe of Z3 theory solvers", in *Formal Methods: Foundations and Applications*, Springer International Publishing, 2020, pp. 8–24. DOI: https://doi.org/10.1007/978-3-030-63882-5_2.

[33] A. D. Vieira, E. A. P. Santos, M. H. de Queiroz, A. B. Leal, A. D. de Paula Neto, and J. E. R. Cury, "A method for PLC implementation of supervisory control of discrete event systems", *IEEE Transactions on Control Systems Technology*, vol. 25, no. 1, pp. 175–191, 2017. DOI: 10.1109/TCST.2016.2544702.

[34] J. Thistle and W. Wonham, "Control problems in a temporal logic framework", *International Journal of Control*, vol. 44, no. 4, pp. 943–976, 1986. DOI: 10.1080/00207178608933645.

[35] B. C. Rawlings, S. Lafortune, and B. E. Ydstie, "Supervisory control of labeled transition systems subject to multiple reachability requirements via symbolic model checking", *IEEE Transactions on Control Systems Technology*, vol. 28, no. 2, pp. 644–652, 2020. DOI: 10.1109/TCST.2018.2877621.

[36] K. T. Seow, "Supervisory control of fair discrete-event systems: A canonical temporal logic foundation", *IEEE Transactions on Automatic Control*, vol. 66, no. 11, pp. 5269–5282, 2021. DOI: 10.1109/TAC.2020.3037156.

[37] S. Jiang and R. Kumar, "Supervisory control of discrete event systems with CTL* temporal logic specifications", *SIAM Journal on Control and Optimization*, vol. 44, no. 6, pp. 2079–2103, 2006. DOI: 10.1137/S0363012902409982.

[38] G. Aucher, "Supervisory control theory in epistemic temporal logic", 2014, pp. 333–340.

[39] K. Ritsuka and K. Rudie, "Do what you know: Coupling knowledge with action in discrete-event systems", *Discrete Event Dynamic Systems*, vol. 33, pp. 257–277, 2023. DOI: 10.1007/s10626-023-00381-z.

[40] X. Geng, D. Ouyang, and C. Han, "Verifying diagnosability of discrete event system with logical formula", *Chinese Journal of Electronics*, vol. 29, pp. 304–311, 2020. DOI: 10.1049/cje.2020.01.008.

[41] L. Feng and W. M. Wonham, "TCT: A computation tool for supervisory control synthesis", in *Proceedings of the 8th International Workshop on Discrete Event Systems*, IEEE, 2006, pp. 388–389.

[42] L. Ricker, S. Lafortune, and S. Genc, "DESUMA: A tool integrating GIDDES and UMDES", in *2006 8th International Workshop on Discrete Event Systems*, 2006, pp. 392–393. DOI: 10.1109/WODES.2006.382402.

[43] R. Malik, K. Åkesson, H. Flordal, and M. Fabian, "Supremica — an efficient tool for large-scale discrete event systems", *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5794–5799, 2017. DOI: 10.1016/j.ifacol.2017.08.427.

[44] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors", *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 175–180, 2002.

[45] B. Brandin, R. Malik, and P. Malik, "Incremental verification and synthesis of discrete-event systems guided by counter examples", *IEEE Transactions on Control Systems Technology*, vol. 12, no. 3, pp. 387–401, 2004.

[46] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional synthesis of modular nonblocking supervisors", *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 150–162, 2013.

[47] N. Bourbaki, *Theory of Sets*. Hermann, 1968, 414 pp.

[48] A. Larionov, A. Davydov, and E. Cherkashin, "The method for translating first-order logic formulas into positively constructed formulas", *Software & Systems*, vol. 32, no. 4, pp. 556–564, 2019, In Russian. DOI: 10.15827/0236-235X.128.556-564.

[49] A. Davydov, A. Larionov, and E. Cherkashin, "On the calculus of positively constructed formulas for automated theorem proving", *Automatic Control and Computer Sciences*, vol. 45, no. 7, pp. 402–407, 2011. DOI: 10.3103/s0146411611070054.

[50] S. Ulyanov, I. Bychkov, and N. Maksimkin, "Event-based path-planning and path-following in unknown environments for underactuated autonomous underwater vehicles", *Applied Sciences*, vol. 10, no. 21, p. 7894, 2020. DOI: 10.3390/app10217894.