Д. А. Мордвинов, Ю. В. Литвинов, Обзор применения формальных методов в робототехнике,
Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление, 2016, выпуск 1, 84–107

https://www.mathnet.ru/ntitu148

DOI: 10.5862/JCSTCS.236.8
УДК 51-74

Д.А. Мордвинов, Ю.В. Литвинов

ОБЗОР ПРИМЕНЕНИЯ ФОРМАЛЬНЫХ МЕТОДОВ В РОБОТОТЕХНИКЕ

D.A. Mordvinov, Yu.V. Litvinov

SURVEY ON FORMAL METHODS IN ROBOTICS

Представлен обзор применения формальных методов в контексте робототехники. Рассмотрены недавние работы, посвященные спецификациям поведения роботов в терминах темпоральных логик, применению идей подхода model checking к таким системам. Также рассмотрено применение формальных методов анализа сетей Петри и моделирования поведения робототехнических систем с их помощью. Отдельное внимание уделено верификации гибридных систем, применению алгебр процессов для спецификации поведения параллельных систем, а также использованию других подходов для верификации и синтеза программ поведения роботов.

ФОРМАЛЬНЫЕ МЕТОДЫ; РОБОТОТЕХНИКА; ТЕМПОРАЛЬНЫЕ ЛОГИКИ; ФОРМАЛЬНАЯ ВЕРИФИКАЦИЯ; СИНТЕЗ ФОРМАЛЬНЫХ СИСТЕМ.

This paper is a survey of applying formal methods in the robotics field. We consider a number of recent works on robotic behavior specification in terms of temporal logics and using the model checking approach. Formal analysis techniques for Petri nets and robotics systems modeling using those methods are also considered. Verification of hybrid systems, application of process algebras for concurrent systems and other approaches for synthesis and verification of robotics controllers are described. We survey both fundamental papers that lay a foundation for the entire branches of research and recent papers from the top conferences of the last five years hoping to cover most of the actively developed research topics.

FORMAL METHODS; ROBOTICS; TEMPORAL LOGICS; FORMAL VERIFICATION; SYNTHESIS OF FORMAL SYSTEMS.

Разговор о формальных методах в большинстве случаев сопряжен с понятием опасности отказа системы или некорректного ее поведения, особенно когда речь идет о дорогостоящих системах и ошибках, приводящих к причинению вреда людям. Чаще всего формальные методы находят применение в военных, космических, медицинских, промышленных технологиях, протоколах общения узлов в компьютерных сетях, о чем написано немало литературы.

Естественным образом аналогичные проблемы появляются и в области робототехники: отказ дорогостоящих робототехнических систем может привести к весьма неприятным последствиям (особенно если речь идет о боевых, медицинских или космических роботах). Неудивительно, что применению формальных методов посвящены целые секции на крупнейших конференциях по робототехнике (таких как ICRA[1] и IROS[2]), на каждой из которых ежегодно публикуются десятки работ. Однако стоит отметить, что далеко не всегда они касаются тематики безопасности поведения роботов. Проблемы, решаемые авторами таких статей, чаще относятся к методам

[1] IEEE International Conference on Robotics and Automation, URL: http://icra2015.org (дата обращения: 30.09.2015).

[2] IEEE/RSJ International Conference on Intelligent Robots and Systems, URL: http://www.iros2015.org (дата обращения: 30.09.2015).

планирования действий и рассуждений, симуляции, управления роботами, а также их групповой координации.

Многие формализмы, построенные в контексте применения формальных методов в робототехнике, выходят за рамки апробации на компьютерных симуляторах, их воплощения работают на дорогостоящих роботах. Например, Verifiable Robotics Research Group[3], авторы одного из наиболее обсуждаемых в настоящее время подходов к планированию действий робота для задачи, поставленной в терминах темпоральных логик, применяют результаты своих работ на одном из крупнейших робототехнических состязаний DARPA Challenge[4] в составе команды Vigir[5]. Известно, что различные методы формальной верификации были применены для марсохода Curiosity, причем, как выяснилось в 2012 г., весьма успешно [1]. Существует и будет рассмотрено применение формальных методов в реабилитационной робототехнике, к созданию автономных промышленных робототехнических систем, робофутболу и т. д.

Задача данной статьи – обзор и классификация применения формальных методов к проблемам робототехники. Будут рассмотрены как самые известные работы середины 90-х гг., положившие начало целым областям исследований, так и работы с последних робототехнических конференций вплоть до 2015 г. Мы не претендуем на полноту обзора как со стороны формальных методов, так и со стороны задач робототехники, тем не менее, многие из самых обсуждаемых идей последних годов, а также самые разработанные за два десятилетия области будут здесь рассмотрены.

Статья представляет собой обзор сразу двух научных областей: формальных методов разработки и верификации, и робототехники. Структура работы отдает приоритет формальным методам, т. е. одна глава соответствует применению конкретного класса формальных методов во всей области робототехники. Рассматриваемые методы можно разделить на два класса: статические и динамические. Статические методы используются для решения двух задач: верификации и синтеза формальных систем. В свою очередь, существует великое множество формальных систем и исчислений, для которых применимы эти методы, из них в работе будут рассмотрены шесть больших классов: конечные системы переходов, сети Петри, алгебры процессов, гибридные системы, марковские модели и формальные логики.

Темпоральные логики и синтез конечных систем переходов

В 1995 г. в статье [2] был предложен новаторский подход к синтезу управления шагающей системой. В работе строится модель искусственной ноги робота в виде конечного автомата с начальным состоянием (start) и состояниями «под нагрузкой» (load), «толчок» (drive), «без нагрузки» (unload), «переносится вперед» (recover) и «соскользнула» (slipped). Далее, рассматривая *полное произведение* (shuffle product) четырех автоматов, авторы получают модель четырехногой шагающей системы (состоящей из 1296 состояний и 5184 переходов). Очевидно, что пространство состояний результирующей системы содержит множество некорректных состояний (например, для двуногой системы не может быть состояния, когда обе ноги переносят свой вес в одно и то же время, или во время переноса одной ноги вторая не может перейти в состояние «без нагрузки»). Для спецификации таких инвариантов (свойств живучести системы, если выражаться в терминологии [3]) авторы используют темпоральную логику ветвящегося времени (CTL, [4]). Исторически это первый пример успешного использования темпоральной логики в контексте робототехнических систем. Техники проверки модели (model checking применены в работе в своем «стандартном» виде: по заданной модели поведения системы и ее

[3] URL: http://verifiablerobotics.com (дата обращения: 30.09.2015).

[4] URL: http://www.theroboticschallenge.org (дата обращения: 30.09.2015).

[5] URL: http://www.teamvigir.org (дата обращения: 30.09.2015).

инварианту проведена формальная верификация соответствия поведения системы требованиям.

Читателю рекомендуется ознакомится с [5] или с [6] для понимания материала данного раздела. Введем понятия и обозначения, используемые далее в статье. Для спецификации формул темпоральной логики мы будем использовать стандартную нотацию: $\bigcirc\varphi$ для оператора *next*, $\Box\varphi$ для оператора *always*, $\Diamond\varphi$ для оператора *eventually* и $\varphi\mathcal{U}\psi$ для оператора *until*. Множество формул темпоральной логики линейного времени (LTL, [7]) вида $(\Box\Diamond p_1 \wedge \ldots \wedge \Box\Diamond p_m) \Rightarrow (\Box\Diamond q_1 \wedge \ldots \wedge \Box\Diamond q_n)$ будем называть, в соответствии с [8], классом *General Reactivity* (1), или *GR*(1). Класс *GR*(1) часто используется в контексте решения задачи синтеза, цель которой – построение программного формализма, удовлетворяющего какой-либо формальной спецификации (в нашем случае, спецификации в терминах темпоральной логики) [9]. По представителю класса *GR*(1) φ можно синтезировать автомат (при условии его существования), все траектории которого удовлетворяют φ, за $O(n^3)$, где n – количество вхождений атомарных предикатов в φ (размер формулы), в то время как для общего случая известны лишь алгоритмы, делающие это с двойной экспоненциальной сложностью [9]. Структурой Крипке [10] будем называть пятерку (S, s_0, R, AP, L), где S – непустое конечное множество состояний, $s_0 \in S$ – начальное состояние, $R \subseteq S \times S$ – тотальное отношение на S, AP – конечное множество атомарных предикатов, L: $S \to 2^{AP}$ – функция пометок.

В 2005, 2007 и 2009 гг. авторы из лаборатории GRASP Пенсильванского университета представили работы, определившие текущее состояние области применения формальных методов в робототехнике. За последние три года (с 2013 по 2015) более половины работ, публикуемых на секциях формальных методов крупнейших робототехнических конференций (таких как ICRA и IROS), так или иначе затрагивают обсуждаемую ниже область, дополняя ее теоретическую базу или проводя эксперименты с применением теоретических наработок этой области.

В статье [11] 2005 г. обсуждается задача планирования траектории движения мобильного робота, формально удовлетворяющей спецификации LTL. Предполагается, что робот представляет собой точку в плоской области, ограниченной многоугольником – окружении робота, внешней среде. Окружение разбивается на n многоугольников (ячеек, регионов), некоторые из которых считаются *интересными*. Интересные регионы могут символизировать комнаты в здании, либо препятствия, с которыми робот не должен сталкиваться. Разбиение геометрического пространства на регионы превращает непрерывную задачу планирования движения в дискретную (*дискретизирует* ее). Каждый регион получает в соответствие свой номер, а также вводится множество предикатов $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, где π_i означает, что «робот находится внутри региона $i \in \{1..n\}$». «Карта» окружения в таком случае будет представлять собой структуру Крипке $D = (S, s_0, R, \Pi, L)$, где $S = \{s_1, s_2, \ldots, s_n\}$ – множество состояний, соответствующих регионам разбиения, s_0 – состояние, соответствующее региону, в котором робот находится в начальный момент времени, R – отношение перехода, $(s_i, s_j) \in R$ тогда и только тогда, когда регион i геометрически смежен региону j, $L(s_i) := \pi_i$.

Основная идея работы – постановка задачи роботу декларативно в виде LTL-спецификации. Например, формула $\varphi = \Diamond(\pi_2 \wedge \Diamond(\pi_3 \wedge \Diamond(\pi_4 \wedge (\neg\pi_2 \wedge \neg\pi_3) \cup \pi_1)))$ для стартового состояния π_1 может означать «посетить регион π_2, затем π_3, затем π_4 и, наконец, вернуться в регион π_1, избегая регионов π_2 и π_3». Для обнаружения траектории, удовлетворяющей спецификации φ на структуре Крипке D окружения, достаточно построить контрпример для свойства $\neg\varphi$ на модели D. В статье проводятся эксперименты с использованием верификаторов NuSMV [12] и SPIN [13]. Полученная дискретная траектория робота затем преобразуется в непрерывное управление при помощи методов теории управления, описанных в [14].

Подход, описанный в [11], может показаться непрактичным и чересчур усложнен-

ным для решаемой задачи, однако идеи, на основе которых он строится, легли в основу действительно полезного метода. Речь идет о статье 2007 г. [15]. Развитие подхода, предложенное в ней, в основном, происходит в двух направлениях: работа с датчиками и мультиагентность. Множество атомарных предикатов теперь имеет вид $AP = \Pi \cup X$, где Π — предикаты принадлежности робота региону, а $X = \{x_1, ..., x_m\}$ — конечное множество предикатов, отражающих информацию с датчика о внешнем мире. В такой модели становится возможным построение поведения с реакцией на внешний мир, например, такая неформальная спецификация: «Стартуя в регионе 1, проверять, не плачет ли ребенок в регионах 2 или 3. Если плачущий ребенок обнаружен, необходимо отыскать родителей в регионах 4, 5 или 6». Расширение модели на мультиагентные системы происходит естественным образом, т. к. каждый робот представляет собой часть окружения другого.

Очевидно, что дискретное управление системой в случае такой модели уже не описывается одной траекторией на «карте» окружения, а представляет собой автомат, учитывающий информацию с датчиков и описывающий все возможные траектории робота, удовлетворяющие LTL-спецификации. Авторы предлагают проводить синтез автомата, описывающего всевозможные поведения робота, удовлетворяющие LTL-спецификации из класса *GR*(1). Структура Крипке, описывающая «карту» окружения, в данном случае становится не нужна и заменяется описанием в терминах LTL: в общую спецификацию системы добавляются утверждения о геометрически смежных регионах вида $\square(\pi_1 \Rightarrow (\bigcirc\pi_2 \wedge \bigcirc\pi_3))$, а также утверждения о состоятельности окружения, например, что в один момент времени робот должен находиться точно в одном регионе. Общий вид LTL-требования имеет вид $\varphi = \varphi_e \Rightarrow \varphi_s$, где φ_e — требования к поведению окружения, φ_s — требования к системе (группе роботов). Говоря неформально, такая спецификация подразумевает, что, если среда «играет не по правилам», т. е. в случае нарушения ожидаемых условий окружения, система не обязана вести себя в соответствии с требованиями к ней.

Интересна трактовка синтеза дискретного управления как игры в математическом смысле между средой и системой роботов. Начиная в стартовом состоянии, среда и система по очереди делают ходы. Среда делает ход первой. Задача среды — опровергнуть спецификацию φ (поэтому она должна «играть по правилам» для того, чтобы импликация была ложной), задача системы — удовлетворять спецификации вне зависимости от того, что делает среда. Если система выигрывает, то синтез требуемого дискретного управления может быть произведен, в случае если выигрывает среда — полностью корректной стратегии не существует.

В статье 2009 г. [16] те же авторы производят синтез непрерывного управления роботом с обратной связью по дискретной модели (т. е. с учетом реакции системы на значения датчиков) с использованием подхода, описанного в [17]. Другой результат, полученный в [16], — формализация действий робота в общем виде. Множество атомарных предикатов расширяется множеством $A = \{a_1, ..., a_k\}$ действий, которые могут выполняться роботом. Каждый из них принимает значение истины в определенном состоянии тогда и только тогда, когда робот выполняет действие, находясь в этом состоянии. Передвижение робота из региона в регион является теперь элементом множества *A* вместе с такими действиями, как, например, захват предмета клешней или включение/выключение камеры.

Достоинства описанного подхода очевидны: по декларативной спецификации требований к системе в выразительном формализме LTL автоматически синтезируется математически корректное «по построению» непрерывное управление группой роботов, которое гарантированно приведет к решению задачи. Тем не менее ограничений у такого подхода достаточно много. Во-первых, система может функционировать только в досконально известной среде, которая ведет себя только так, как ожидается при постановке требований. Во-вторых, хоть сложность синтеза полиноми-

альна от размера пространства состояний, сам размер пространства состояний может зависеть экспоненциально от количества входов и выходов (датчиков и действий). Однако в результате получается готовый алгоритм поведения системы, т. е. синтез можно произвести лишь один раз.

Шаги к решению как минимум второй проблемы были сделаны в работе [18] 2013 г., главный вклад которой заключается в описании подхода синтезирования дискретной стратегии робота в онлайн-режиме, «на лету». Получая на вход все ту же LTL-спецификацию в форме *GR*(1), описывающую как «карту» окружения, так и ограничения и цели системы, алгоритм выдает на выходе не автомат, описывающий корректное поведение робота, а величину *горизонта*. Интуитивно *горизонт* – это глубина, на которую нужно просчитывать всевозможные состояния, получаемые из текущего, иначе говоря, это высота дерева игры, количество ходов, которые нужно просчитать в контексте математической игры между средой и роботом для того, чтобы локальные решения, принятые по такому неполному обсчету, гарантированно приводили бы к глобально корректному поведению. В худшем случае трудоемкость принятия решения на определенном шаге сравнится с трудоемкостью алгоритма построения автомата синтеза полной стратегии, применяемого ранее. Работоспособность такого подхода объясняется следующим фактом. В статье [8] построение автомата по LTL-спецификации производилось за два шага: итеративное построение множества «выигрышных» для системы состояний (вычисление неподвижной точки состояний с инвариантом «выигрышности»), и далее сам синтез автомата по результатам каждого шага вычисления неподвижной точки. Подсчет неподвижной точки оперирует пространствами состояний, допуская применение символических структур данных, которые показали себя на практике очень лаконично представляющими множества состояний. Интуитивно обсуждаемый алгоритм принятия решения в реальном времени использует только вычисление неподвижной точки, не проводя самого синтеза автомата, что и дает выигрыш во времени.

Схема работы системы теперь принимает следующий вид. В каждый момент времени робот производит наблюдение за поведением среды, т. е. определяет «ход» оппонента. Это может быть сдвиг передвигающегося препятствия или сигнал «батарея почти разряжена». Среди всех «выигрышных» состояний, полученных в результате вычисления неподвижной точки, выбирается наиболее близкое в смысле пути на графе состояний с одной эвристикой исключения уже посещенных состояний, чтобы не случилось «зацикливания» системы в среде. Такая эвристика должна быть проигнорирована, если среда «нарушает правила», это повлечет за собой повторные попытки выполнения задачи роботом.

Кроме возможности планирования поведения робота в режиме реального времени, такой подход позволяет системе принять полностью реактивный характер. Другими словами, система получает возможность функционирования в изменяющемся со временем окружении (но только если характер изменений среды полностью известен на момент синтеза управления). В частности, становится возможным объезд движущегося препятствия, что ранее было слабо затронуто в статьях на тему планирования поведения по LTL-спецификации. Работа [8] содержит результаты экспериментов по синтезу поведения робота при наличии подвижного препятствия.

Существует множество статей с экспериментами, поставленными на теоретической основе обсуждаемого подхода. Например, в [19] рассказывается о применении LTL-спецификаций для планирования движений манипулятора робота-бармена за прилавком суши-ресторана. Применение именно LTL-подхода дало значительный прирост в «интеллектуальности» системы. Например, робот «понимал», что предметы, которые лежат на месте других или не дают перенести банку на другое место, должны быть временно отодвинуты в другую сторону, что раньше достигалось только методом *бэктрекинга*, описанном в статье [20]. Однако бэктрекинг работал не во всех ситуациях и был крайне неэффективен

во временном и ресурсном отношении). Работа [21] описывает планирование для мультиагентной системы раздельного сбора отходов. В [22] обсуждается планирование поведения робота-спасателя, исследующего место катастрофы с целью тушения пожаров и предоставления первой помощи выжившим.

Для формальных спецификаций поведения робота используются и другие фрагменты логики LTL. Например, в [23] предлагается использование специфичного подкласса LTL для эффективного синтеза стратегии поведения робота. Главная особенность предложенного фрагмента LTL состоит в том, что он позволяет проводить синтез стратегии прямо на изначальной модели Крипке, без использования теоретико-автоматного подхода, что может сильно уменьшить время синтеза, т. к. отпадает необходимость рассматривать композицию с автоматами для LTL-ограничения, значительно увеличивающую размер итоговой системы. Например, в тексте статьи описана ситуация, когда подход с использованием предложенного фрагмента LTL синтезирует стратегию за время порядка минуты на системе из 300 000 состояний, в то время как реализация подхода с использованием *GR*(1)-фрагмента не закончила работу на системе из 100 000 состояний. Однако сравнение проводилось с реализацией, не использующей символические методы представления пространства состояний *GR*(1)-системы, что является сильной стороной *GR*(1)-подхода.

Существуют и работы, в которых для спецификации поведения роботов используются другие темпоральные логики. Например, в [24] предложен язык MASL, расширяющий LTL временными ограничениями и кванторами коалиций, что полезно для спецификации поведения мультиагентных систем. Там же можно найти подробный обзор применения различных темпоральных логик в теории мультиагентных систем.

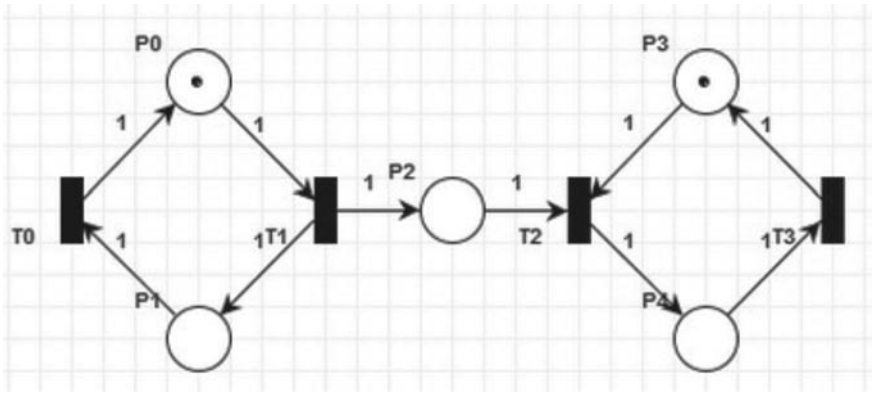
Следует отметить, что количество опубликованных работ на тему планирования поведения робота при помощи спецификаций в темпоральных логиках на момент 2015 г. имеет порядок сотен или даже тысяч, поэтому мы считаем уместным ограничиться здесь лишь рассмотренными статьями, т. к. они дают общую характеристику подхода. Актуальные статьи на данную тему можно найти, в частности, в списках докладов секций формальных методов конференций ICRA и IROS.

Сети Петри

Один из самых старых и некогда один из самых популярных методов формального анализа программ – сети Петри. Впервые предложенные Карлом Петри в 1962 г., они предназначались прежде всего для моделирования и анализа параллельных и распределенных систем. Они интересны тем, что имеют наглядную графическую нотацию, и многие их свойства, имеющие важное практическое значение, можно установить аналитически, зная топологию и начальное состояние сети. Сети Петри быстро стали популярны: например, в статье [25], опубликованной в 1991 г., приводится библиография из 4099 статей, рассматривающих сети Петри или их применение. Мы здесь рассмотрим лишь некоторое применение сетей Петри в робототехнике.

Сетью Петри мы, следуя [26], будем называть тройку (P, T, φ), где P – конечное множество позиций, T – конечное множество переходов, а $\varphi : (P \times T) \cup (T \times P) \to \mathbb{N}$ – функция потока. Маркировкой называется отображение $\mu : P \to \mathbb{N}$, ставящее в соответствие позиции количество маркеров, находящихся в данной позиции. Графически сеть Петри представляется как двудольный граф, в котором позиции обозначаются кругами (и маркеры – точками в позициях), переходы – прямоугольниками, функция потока рисуется как ребра графа (каждое ребро может быть входным или выходным и иметь вес, который пишется над ребром, см. рисунок).

Переход $t \in T$ называется активным при маркировке μ, если $\forall p \in P\ \varphi(p,t) \le \mu(p)$. Любой активный переход сети может сработать, в этом случае из каждой входной позиции p перехода удаляются $\varphi(p,t)$ маркеров и для каждой выходной позиции перехода p' в нее добавляются $\varphi(t, p')$



Визуализация сети Петри

маркеров. Можно записать срабатывание перехода как $\mu \xrightarrow{t} \mu'$, где $\mu'(p) = \mu(p) - \varphi(p,t) + \varphi(t,p)\ \forall p \in P$.

Практический смысл приведенных определений можно представить себе так: позиция – это состояние системы, либо предусловие для выполнения некоторого действия. Переход – это действие, выполняемое системой. Функция потока показывает поток управления или поток данных в системе, маркировка показывает состояние, в котором в определенный момент времени находится система (при этом маркер можно понимать как указатель на текущую исполняемую инструкцию, хотя, конечно, это далеко не всегда так), активный переход – действие, которое может быть исполнено, тот факт, что может сработать любой активный переход в сети – неопределенность, возникающую при параллельном исполнении (гонки), срабатывание перехода – исполнение действия и передачу управления (или данных) дальше.

Про сеть Петри с заданной топологией и маркировкой можно аналитически установить следующие свойства:

Достижимость. Существует ли последовательность срабатываний переходов из маркировки μ в маркировку μ'. Практический смысл – может ли система в принципе решить поставленную задачу, или наоборот, оказаться в нежелательном состоянии.

Ограниченность. Существует наперед заданное число k такое, что в любой достижимой маркировке сети в каждой позиции находится не более k маркеров. Практический смысл – в системе никогда не произойдет переполнения буфера.

Безопасность. В каждой позиции сети в каждой достижимой маркировке находится не более одного маркера. Практический смысл – возможность не использовать буферы вообще, проверка корректности моделирования (например, движения собираемой детали по конвейеру).

Живость. Достижимость из заданной маркировки такой маркировки, в которой заданный переход может сработать. Живость вводится как для конкретного перехода, так и для сети в целом, при этом определяется шесть уровней живости, от нулевого (переход не может сработать никогда) до пятого (в любой маркировке, достижимой из заданной, существует последовательность переходов, в которой переход сработает хотя бы один раз). Практический смысл — анализ системы на возможность тупиковых ситуаций, поиск мертвого кода.

Существует несколько аналитических методов анализа, которые будут лишь упомянуты здесь в силу ограниченности объема статьи:

Анализ уравнения состояния. Сеть Петри представляется в виде матрицы инцидентности, тогда текущее состояние выражается в виде линейного уравнения, решения которого показывают количество срабатываний переходов, необходимых для перевода сети из начального состояния в некоторое заданное. Если решений нет, заданное состояние недостижимо, но если решения

есть, это еще не значит, что заданное состояние достижимо. Методы, связанные с уравнением состояния, имеют низкую вычислительную сложность, но некоторые важные свойства установить не могут.

Анализ графа (или дерева) достижимости. Вершинами такого графа являются достижимые состояния сети, ребрами – срабатывающие переходы. Такие методы позволяют установить любое из перечисленных свойств сети, но для неограниченных сетей граф будет бесконечным. Для анализа неограниченных сетей вводится граф покрытия, в котором потенциально бесконечное количество маркеров обозначается специальным символом ω, но в таком случае теряется информация о переходах сети и, например, достижимость установить уже не удастся. В любом случае, граф достижимости (покрытия) имеет размер, в общем случае экспоненциально зависящий от количества позиций в сети, так что алгоритмы анализа, как правило, имеют экспоненциальную трудоемкость.

Структурный анализ. Исследование топологии сети в поиске структур с известными свойствами. Этот метод тоже не позволяет установить все свойства.

Как видно, в общем случае доказательство практически важных свойств сети Петри имеет большую временную трудоемкость, но, тем не менее, алгоритмически разрешимо. Поэтому сети Петри имеют меньшую выразительную силу, чем машины Тьюринга, это их недостаток (поскольку сетью Петри нельзя смоделировать произвольную программу) и достоинство (для машин Тьюринга перечисленные выше свойства в общем случае алгоритмически неразрешимы). Вводятся различные расширения сетей Петри, такие как ингибиторные сети, цветные сети, сети с приоритетами, временные сети и т. д., большинство из них тьюринг-полны, так что не поддаются формальному анализу. Они, тем не менее, интересны тем, что имеют формальную семантику и могут использоваться для динамической верификации путем симуляции сети. На этом теоретическое введение заканчивается, подробнее про сети Петри можно прочитать в [27], оценки трудоемкости анализа сетей приводятся в [26], подробный обзор работ о разрешимости некоторых проблем в контексте сетей Петри до 1994 г. можно найти в [28], хорошее введение в сети Петри с точки зрения их приложений в промышленности приводится в [29], пример работы о применении в технологическом процессе – работа [30].

В робототехнике сети Петри используются разными способами. В работах [31, 32] сеть Петри используется для моделирования роботов-футболистов и их окружения. При этом инструментальные средства, предлагаемые авторами, позволяют сначала оценить качественные свойства системы (например, возможно ли вообще в данных условиях забить мяч в ворота), рассматривая модель как классическую сеть Петри, затем, с помощью симуляции, количественные свойства (такие как вероятность забить гол и оценка времени для решения этой задачи), рассматривая модель как временную сеть. Аналогично, статья [33] описывает симулятор для разных видов сетей Петри (временных, ингибиторных, с приоритетами), используемый для верификации и оптимизации разных технологических процессов (например, добычи руды в шахте с использованием полуавтономных погрузочно-транспортных машин). Для верификации робототехнических систем расширенные сети Петри используются и в работах [34, 35], причем в последнем случае предлагается новое расширение сетей Петри, и для их формального анализа используются техники проверки моделей.

Следующий важный класс применений сетей Петри в робототехнике – в качестве языка программирования. Все такие работы используют расширенные сети, что делает невозможным их формальный анализ, но наличие четкой семантики и наглядность упрощают разработку программ и написание для них интерпретаторов. Примеры таких работ – [36] (визуальный язык со вставками на языке Python для программирования мультиагентных систем), [37] (интерпретация ингибиторных сетей Петри для управления группой роботов, с визуальным редактором сетей и посылкой команд роботам по радиоканалу), [38] (программи-

рование одновременно нескольких роботов с помощью расширенной сети, где задания для каждого робота «нарезаются» из общей программы, с приложением опять-таки в робофутболе).

Сети Петри также используются для планирования и распределения задач между роботами в группе, хороший пример такой работы – [39], здесь действия каждого робота в группе моделируются ингибиторной сетью, определяется, может ли один робот в группе заменить другого для решения той или иной подзадачи, предлагается алгоритм построения оптимального с точки зрения времени выполнения плана работы группы.

Несколько менее типичный пример использования сети Петри – работа [40], где сеть используется для представления знаний робота при обучении по демонстрации. Человек-учитель перемещает предметы в поле зрения робота, при этом стационарные состояния предметов становятся позициями сети, а сами перемещения – переходами. После этого робот анализирует граф достижимости сети, чтобы найти кратчайшую последовательность перемещений, которые привели бы предметы из одного состояния в другое.

Communicating Sequential Processes

Речь в данном разделе пойдет о формальном языке описания параллельных систем CSP (Communicating Sequential Processes), языке occam[6], построенном на формализме CSP, и о некоторых работах, посвященных их применению в робототехнике. CSP принадлежит к классу математических теорий под общим названием *алгебры процессов* (*process algebras*) [41, 42]. В простейшем виде алгебра процессов для множества атомарных действий A определяется как замыкание A относительно операций альтернативы («+»), последовательной композиции («·») и параллельной композиции («‖»).

Алгебра процессов CSP состоит из множества атомарных событий и элементарных процессов STOP и SKIP. Первый – процесс над пустым алфавитом, т. е. процесс, не взаимодействующий с другими процессами и окружением и означающий аварийную остановку работы (иначе его называют *тупиком*). Второй означает нормальное завершение работы системы. Каждое событие является атомарной сущностью, соответствующей какому-либо явлению моделируемой предметной области. Все события в модели CSP являются мгновенными, при этом не важен интервал времени, разделяющий события, важна лишь упорядоченность событий.

Для подробного ознакомления с формализмом мы отсылаем читателя к оригинальным работам [43–45]. Здесь будут неформально определены основные операции в алгебре CSP.

- *Префикс* («→»). Выражение $Q = a \to P$ конструирует новый процесс Q, который ждет события a, а затем ведет себя как P. В качестве примера приведем простейший бесконечный процесс работы часов: $CLOCK = tick \to CLOCK$. Такая запись может быть развернута бесконечной подстановкой выражения для $CLOCK$ в правую часть в последовательность $tick \to tick \to tick \to ....$ Запись $Q = (x : B \to P(x))$ предлагает выбрать любое событие x из множества событий B, и далее процесс ведет себя как $P(x)$. В таком случае B называют *меню* процесса P.
- *Недетерминированный выбор* («⊓»). Выражение $P = Q \sqcap R$ конструирует процесс, который ведет себя либо как Q, либо как R, при этом выбор имеет внутреннюю природу, которая не определяется, и среда не может влиять на то, в чью пользу будет сделан этот выбор. Другими словами, процесс $(a \to Q) \sqcap (b \to R)$ может отказаться принимать a или b, и природа этого отказа носит внутренний характер.
- Детерминированный выбор («□»). $P = (a \to Q) \square (b \to R)$ конструирует процесс P, который ведет себя как P или Q в зависимости от события, выработанного окружением. В случае, когда окружение произвело событие, принимаемое обоими операндами, выбор между ними будет про-

[6] Страница occam на сайте WoTUG, URL: http://www.wotug.org/occam (дата обращения: 30.09.2015).

изведен недетерминированно.

• *Параллельная композиция по пересечению* («‖»). Данный оператор позволяет процессам исполняться параллельно, но только в случае, когда оба процесса к этому готовы. Более формально, $(x : B \to P(x)) \parallel (y : C \to Q(y)) = (z : (B \cap C) \to (P(z) \parallel Q(z)))$. Оператор параллельной композиции по пересечению часто применяется для описания синхронизации процессов в формализме CSP.

• *Параллельная композиция чередованием* (*interleaving*, «|||»). В отличие от композиции по пересечению, чередование позволяет исполняться параллельно независимо друг от друга, т. е. каждое событие требует участия одного процесса вместо двух. Формально, если B – меню процесса P, а C – меню процесса Q, то $P \,|||\, Q = (x : B \to (P(x) \,|||\, Q)) \square (y : C \to (P \,|||\, Q(y)))$, то есть события P и Q «чередуются» во времени, откуда и произошло название операции.

• *Последовательная композиция* («;»). Позволяет вести себя процессу (P; Q) как процессу P до тех пор, пока тот не закончит свою работу, а затем как Q. В частности,

```
SKIP;P = P,
STOP;P = STOP.
```

• *Сокрытие* (*concealment*, «\»). Процесс $P \setminus b$ ведет себя как P, за исключением того, что все вхождения b удаляются из всех траекторий исполнения P. В частности, $(b \to P) \setminus b = P \setminus b$.

• *Образ и прообраз.* Пусть f – тотальное отношение из множества событий во множество событий, причем прообраз каждого события конечен. Тогда $f(P)$ определяет процесс, который выполняет $f(a)$ тогда и только тогда, когда P выполняет a, $f^{-1}(P)$ — процесс, который выполняет a тогда и только тогда, когда P выполняет $f(a)$.

Существует множество инструментов для анализа CSP, например, FDR2[7] и PAT[8]. FDR2, применение которого мы рассмотрим ниже, является коммерческим инструментом проверки моделей для CSP (*refinement checker*), позволяющим формально доказать выполнимость свойств процесса, описанного на входном языке FDR2, практически полностью совпадающем с версией языка CSP самого Ч.Э. Хоара (автора CSP) из [45]. Также CSP повлиял на такие языки программирования, как occam-π [46] и Go[9].

Перейдем к рассмотрению применения описанного формализма к задачам робототехники. В 1998 г. была опубликована статья [47], в которой обсуждается применение формальных методов в целях безопасности на примере реабилитационной робототехники. Работа ведется в контексте бременского автономного кресла – интеллектуальной системы для передвижения людей преклонного возраста и людей с ограниченными возможностями. Предлагается использование подхода с моделированием дерева опасностей [48, 49] посредством CSP. Дерево опасностей представляет собой иерархическое представление всех рисков, которым подвергается система, начиная от самого абстрактного описания в корневых узлах до самого конкретного в листьях. Узлы-дети конкретизируют опасности родителя, объединяясь либо логическим И (родительская опасность случается тогда и только тогда, когда случаются все опасности, соответствующие узлам-детям), либо логическим ИЛИ (родительская опасность случается, когда случается хотя бы одна дочерняя опасность). Полное дерево опасностей для бременского автономного кресла имеет 170 узлов и охватывает опасности от угроз столкновения (активного и пассивного, т. е. произошедших по вине самого кресла и по причине внешних обстоятельств) до угрозы потери связи между программными и аппаратными компонентами кресла. Дерево опасностей (или его поддеревья) далее описывается на входном языке FDR2, происходит моделирование окружения и системы со случайными выработками событий опасностей. Утверж-

[7] Домашняя страница FDR2, URL: http://www.fsel.com (дата обращения: 30.09.2015).

[8] Домашняя страница PAT, URL: http://pat.comp.nus.edu.sg (дата обращения:30.09.2015).

[9] Домашняя страница Go, URL: https://golang.org (дата обращения: 30.09.2015).

дается, что такой подход более выразителен, чем просто постановка требований в терминах LTL (см. выше в данной статье), т. к. в общем случае дерево опасностей не выражается в LTL. Наконец, производится верификация сетевых протоколов взаимодействия компонент кресла с блокировками и двойной буферизацией, а также обсуждается другое возможное применение для подхода с деревом опасностей.

Работа [50] интересна в контексте образовательной робототехники, а также в контексте визуального моделирования [51]. В ней обсуждается разработка визуального языка для программирования роботов Surveyor SRV-1 в целях обучения студентов Университета Кента параллельному программированию, для чего ранее использовался язык occam-π [46], основанный на формализмах π-исчисления [46] и CSP.

В работе [52] обсуждается применение языка occam-π для построения роевой [53] системы роботов сбора отходов. Для реализации системы выбрана архитектура Брукса [54], в которой система представляется как набор уровней ответственности, при этом работоспособность верхних уровней напрямую зависит от нижних, которые могут, например, контролировать работу датчиков или обеспечивать передвижение мобильного робота, избегая при этом столкновений с препятствиями. Характерной чертой такой архитектуры является то, что отказ работы верхних уровней не влечет отказа всей системы, робот все еще в состоянии делать более «примитивные» действия. Использование occam-π позволило представить каждый уровень ответственности как отдельный процесс, что оказалось удобным при программировании системы сбора отходов группой роботов.

Работа [55] изучает подход к планированию передвижения роботов с использованием алгебр процессов. Общая схема подхода примерно такая же, как у описанного выше в данной статье, однако вместо спецификаций LTL используются алгебры процессов.

Гибридные системы

Робототехническая система, действующая в реальном мире, всегда является *гибридной системой* в том смысле, что совмещает дискретную логику поведения, отражающую внутреннее состояние системы, с непрерывной динамикой (управление скоростями моторов, углом соединения деталей, длина тормозного пути и т. д.). При этом класс гибридных систем не ограничивается одними лишь роботами. Их использование довольно естественно в автомобиле-, авиа- и судостроении, а также в различных отраслях науки (включая такие, как экология хищников-жертв).

На тему теории гибридных систем выходят целые сборники (например, [56, 57]), создано множество инструментов анализа, включая HyTech [58], PHAVer [59], KeYmaera [60]. Мы рекомендуем работу [61] для ознакомления с общими принципами работы верификаторов гибридных систем и эмпирическое исследование о верификации и валидации в гибридных киберфизических системах [62] как хороший обзор научных работ и индустриальной практики в этой области. Интересно, что согласно результатам обзора, почти все участники опроса используют методы проверки моделей, но только для очень ограниченных частей разрабатываемых систем, тогда как основная доля работ по обеспечению качества приходится все-таки на ручное тестирование.

Рассмотрение робота как гибридной системы помогает уточнить и улучшить результаты применения других формальных методов. Например, в работе [63] вводится формализация робота как гибридной системы со сложной динамикой, к которой применяется аппарат темпоральных логик для планирования траектории, близкий к [15]. Планирование выполняется на двух уровнях: высоком (строящем примерный маршрут, максимально «хорошо» удовлетворяющий LTL-спецификации) и низком, где учитывается непрерывная динамика робота. При этом планировщик низкого уровня может влиять на решения планировщика высокого уровня, например, если проложенный маршрут физически невоз-

можен для данного робота. Предложенный авторами подход позволяет не перестраивать LTL-спецификацию при обнаружении препятствий, тем избегая вычислительно сложных операций.

В работе [64] как гибридная система рассматривается промышленная роботизированная сборочная линия. Поведение системы из манипуляторов и контейнеров описывается в виде набора дифференциальных уравнений, решается задача планирования работы системы. Подход позволяет синтезировать непрерывное управление, проводящее систему через набор дискретных состояний (таких как «взять деталь», «передвинуть контейнер» и т. д.) с учетом показаний сенсоров, при этом могут быть удобно выражены неожиданные события, например, появление препятствия в зоне действия робота. После устранения препятствия система может продолжить исполнение плана без необходимости перепланирования. Для достижения этого применяется исключение переменной времени из уравнений и ввод искусственной переменной, с которой ведется работа в уравнениях (reference variable), в которой кодируется и непрерывное, и дискретное состояние системы (пространственное положение манипулятора и задача, которую он в данный момент выполняет). Для удобного выражения последовательных и параллельных процессов применяется max-plus algebra: над множеством $\mathrm{R} \cup \infty$ вводятся операции $\oplus$ и $\otimes$:

$$a \otimes b = a + b$$
$$a \oplus b = \max(a, b).$$

Уравнения, описывающие поведение системы из нескольких манипуляторов и конвейера, выражаются в этой алгебре, после чего их относительно несложно анализировать.

Также для анализа гибридных систем используется дифференциальная динамическая логика ($d\mathcal{L}$). В работе [65] представляется инструмент Sphinx, позволяющий описывать модель гибридной системы на визуальном языке (профиле для диаграмм активностей и диаграмм классов языка UML), после чего модель может быть переведена в текстовую форму и проанализирована инструментом KeYmaera. Средство реализовано на платформе Eclipse Modeling Project[10], поддерживает коллаборативную работу над моделью и доказательствами корректности (используются возможности Eclipse, в частности, EMF Compare[11]), средства контроля версий. Возможности по обеспечению коллаборативной работы позволяют распределять задачи по формализации и верификации системы, эффективно используя знания членов команды, а использование визуальных языков позволяет сделать модель более наглядной и понятной. Кроме того, авторы упоминают, что возможности по реализации предметно-ориентированных языков платформы Eclipse Modeling Project позволяют описывать систему на разных языках, привычных для разных групп экспертов, и автоматически транслировать их в язык $d\mathcal{L}$.

Марковские модели

Еще одной активно развивающейся областью применения математического аппарата к задачам робототехники является теория марковских процессов принятия решений. О марковских процессах принятия решений написано довольно много литературы, в том числе на русском языке, например, книга [67]. Применение теории марковских процессов принятия решений к задаче планирования, что ближе к материалу данной статьи, довольно подробно рассмотрено в [68].

Марковским процессом принятия решений (*Markov Decision Process – MDP*) назовем шестерку $M = (S, A, D_0, P, R, \gamma)$, где S – конечное множество состояний; A – конечное множество действий; $D_0 : S \to \mathbb{R}$ – начальное распределение вероятности по состояниям; $P : S \times A \times S \to [0,1]$ — функция вероятности перехода. По данным s, $s' \in S$

[10] Обзор платформы на русском языке можно найти в работе [66].

[11] Домашняя страница средства визуального сравнения моделей EMF Compare, URL: https://www.eclipse.org/emf/compare/ (дата обращения: 04.02.2016).

и $a \in A$ величина $P(s, a, s')$ означает вероятность перехода из состояния s в состояние s' при совершении системой действия a; $R : S \times A \times S \to \mathbb{R}$ – функция награды. Значение $P(s, a, s')$ есть величина награды, полученной системой немедленно после перехода из состояния s в состояние s' при выполнении действия $a(s, s' \in S, a \in A)$; $\gamma \in [0,1]$ – фактор скидки, означающий важность ближайших наград относительно будущих наград.

Политикой процесса M с конечной памятью назовем отображение $f : S^* \to D(A)$, где S^* – множество конечных кортежей, все элементы которого принадлежат S, $D(X)$ – множество функций распределения вероятности над множеством X. Политика f с конечной памятью возвращает по «истории» передвижения по состояниям марковского процесса распределение вероятностей очередного действия системы. *Детерминированная политика* – политика марковского процесса, отображающая конечный путь в одно конкретное действие. В контексте робототехники обычно говорят о детерминированных политиках. Часто используется понятие *политики без памяти*, или *стационарной политики*, когда в действие отображается только текущее состояние: $f : S \to A$.

В классическом виде задачей анализа MDP является поиск политики f, максимизирующей потенциальную награду над бесконечным горизонтом, иначе говоря, ставится экстремальная задача:

$$\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \to \max,$$

где действие a_t выбирается по состоянию s_t в соответствие с политикой f. Значения γ, близкие к нулю, заставляют систему пренебрегать более удаленными во времени наградами, близкие к единице – учитывать их.

MDP применяются для моделирования поведения дискретных систем, поведение которых носит стохастический характер, поэтому часто используются в робототехнике. С их помощью моделируется поведение роботов и их частей (вероятности переходов в таком случае означают вероятности успешного выполнения задачи, например, захвата предмета клешней), элементов окружения и даже поведения человека [69–71] (например, когнитивного состояния оператора робота или человека как движущегося препятствия).

В контексте формальных методов интересны расширения марковских процессов множеством атомарных предикатов AP и функцией пометок $L : S \to 2^{AP}$. Такая структура носит название *марковского процесса принятия решений с метками* (*Labeled Markov Decision Process – LMDP*). Для систем, моделируемых при помощи LMDP, часто ставится задача *проверки вероятностных моделей* (*probabilistic model checking*) [72]. Методы, применяемые для проверки вероятностных моделей, в целом аналогичны методам классической проверки моделей. Нередко ограничения ставятся на языке логики PCTL[12] или PCTL*, хотя практически во всех упомянутых ниже работах для постановки задачи роботу используется темпоральная логика линейного времени.

Рассмотрим некоторое применение методов проверки вероятностных моделей к задачам робототехники. В работе [73] описан метод синтеза политики, гарантирующей выполнение требований в форме произвольной формулы LTL с вероятностью равной единице, при этом такая политика будет оптимальной с точки зрения награды (или стоимости, что эквивалентно). Методы синтеза такой политики без учета LTL-ограничений могут быть найдены, например, в [74]. Дополнительные ограничения в форме LTL заставляют рассматривать синхронное произведение MDP системы и автомата Рабина, соответствующего LTL-ограничению, что может сильно увеличивать размер всей системы. Результаты применяются к решению задачи передвижения мобильного робота. Анализ системы в ситуации произвольных формул LTL в худшем случае имеет двойную экспоненциальную сложность, к тому же результирующая политика в общем случае не обязана иметь конечную память (последнее

[12] Probabilistic Computation Tree Logic.

все же не играет особой роли на практике). Существуют и более практичные подходы, использующие фрагменты LTL, что нередко снижает трудоемкость синтеза и анализа вероятностных систем.

Работа [75] описывает простые методы для синтеза стационарной политики передвижения мобильного робота в среде, описываемой марковским процессом принятия решений. При этом ограничения ставятся в форме логики co-safe LTL, описывающей только конечные траектории системы. Модель не поддерживает возможности задать реакцию робота на значения с датчиков. Статья [23], уже упомянутая выше, содержит описание подхода к синтезу политики с конечной памятью для ограничений в форме специфичного фрагмента LTL за полиномиальное время без использования теоретико-автоматного подхода.

В [76] приводится интересное приложение теории проверки вероятностных моделей к задаче *гибридного управления роботом* (*shared autonomy robot*). Для робота, у которого есть как подсистема операторского контроля, так и подсистема автономного поведения, строится такая модель поведения системы, которая находится в состоянии парето-оптимума между максимизацией вероятности выполнения задания, поставленного LTL-спецификацией, и стоимостью работы оператора. Для синтеза такой стратегии поведение робота и когнитивное состояние оператора представляется марковским процессом.

Другая часто применяемая в контексте робототехники математическая структура – *частично наблюдаемые марковские процессы принятия решений* (*Partially Observable Markov Decision Process – POMDP*). От обычных MDP POMDP отличаются тем, что состояние, в котором находится система, неизвестно, и система может только поддерживать набор вероятностей нахождения в определенном состоянии, который она получает из наблюдений о поведении окружающей среды в ответ на выполнение какого-либо действия. Формально, частично наблюдаемый марковский процесс принятия решений – кортеж (S, A, D_0, P, R, Z, O, γ), где S, A, D_0, P, R и γ соответствуют аналогичным элементам из определения MDP, Z – конечное множество наблюдений, $O : S \to Z$ – отображение, сопоставляющее состоянию наблюдение. Очевидно, что в реальном мире часто приходится сталкиваться с ситуациями, описывающимися POMDP, что подтверждается большим числом работ, посвященных применению POMDP в робототехнике.

Формальный анализ POMDP разделяется на *количественный* и *качественный*. Задача качественного анализа POMDP – нахождение политики, удовлетворяющей цели с вероятностью, равной единице. Количественный анализ задается вопросом об удовлетворении цели с вероятностью $\lambda \in (0,1)$. В [77] доказывается неразрешимость задачи количественного анализа. В [78] подробно описываются текущие результаты разрешимости задач анализа POMDP, в частности, приводится конструктивное доказательство разрешимости EXPTIME-полноты качественного анализа POMDP.

В [79] исследуется применение качественного анализа POMDP совместно с LTL-требованиями для классических задач планирования поведения робота. Работа [80] описывает применение темпоральной логики линейного времени и POMDP к управлению квадрокоптером, наблюдающим за ситуацией на земле по видеокамере.

Другие используемые формализмы

Process Algebra for Robot Schemas. Существуют формализмы, которые не попали ни в одну из категорий выше. Один из таких формализмов – вариант алгебры процессов – PARS (Process Algebra for Robot Schemas), описанный в работах [81, 82]. Этот подход рассматривает модель программы робота, аппаратного обеспечения робота и среды как набор процессов, каждый из которых имеет набор входных и выходных переменных, преобразует входные переменные в выходные, используя данные, которые он может читать из входных портов, при этом процесс может писать в выходные порты. Каждый процесс может завершиться в двух состояниях – stop и abort, или не завершиться вовсе. Процессы комбинируются с помощью операторов «;»

(последовательное выполнение), «|» (параллельное выполнение до тех пор, пока все процессы не завершатся) и «#» (параллельное выполнение до тех пор, пока хоть один из процессов не завершится). Последовательное выполнение передает управление следующему процессу, только если предыдущий завершился в состоянии stop, что дает возможность моделировать условный оператор:

$$T = (Eq\langle a, b\rangle; P \mid Neq\langle a, b\rangle; Q),$$

где *Eq* – процесс, который заканчивается в состоянии stop, если его входные переменные *a* и *b* равны, и в состоянии abort в противном случае; *Neq* – аналогично моделирует неравенство. Циклическое исполнение моделируется через хвостовую рекурсию $T\langle a\rangle\langle b\rangle = P\langle a\rangle\langle b\rangle; T\langle b\rangle$.

Таким образом, выразительной мощности алгебры процессов достаточно, чтобы выразить любую программу. Временной аспект вводится в модель посредством процесса $Delay\langle t\rangle$, который останавливается в состоянии stop по истечении времени *t*, коммуникации между параллельно исполняемыми частями системы – через процессы $In\langle p\rangle\langle x\rangle$ и $Out\langle p, x\rangle$.

Ограничение на систему вводится как выражение на PARS, дополненное логическим условием на переменные, используемые процессами в выражении. Для того чтобы моделировать неопределенность, имеющуюся в реальном мире, переменные рассматриваются как случайные величины с некоторой функцией распределения. Вводится процесс $Ran\langle\Phi\rangle\langle\nu\rangle$, возвращающий случайную величину ν с распределением Φ.

Цель введения такого формализма – подавление комбинаторного взрыва, которому подвержены многие другие методы анализа (сети Петри, подходы, основанные на темпоральных логиках). Достигается это так: рассматриваются только системы, представляющиеся в виде совокупности параллельных циклических процессов (выражаемых с помощью хвостовой рекурсии). Для них считается функция потока, показывающая, как входные параметры преобразуются в выходные за одну итерацию. Это не накладывает серьезных ограничений на моделируемые системы, поскольку представление в требуемом виде во многих случаях может быть построено автоматически (причем, полиномиальным относительно количества процессов алгоритмом). Далее переменные, используемые процессами, «пропускаются» через динамическую байесовскую сеть, чтобы определить вероятность удовлетворения ограничения. Таким образом, можно априори определить вероятность успешного выполнения миссии. Авторы поставили более сотни экспериментов, в результате которых установлено, что реальные значения вероятности выполнения миссии статистически соответствуют предсказанным.

Integrated Behavior-Based Control. Еще один интересный формализм, используемый для программирования роботов, – Integrated Behavior-Based Control (i2BC), формализм и архитектура, описанные в [83]. Архитектура базируется на известной в среде робототехников работе [54] и представляет программу в виде сети взаимодействующих *поведений*. Поведение преобразует входные данные в выходные, кроме того, имеет входы *активации* и *ингибиции*. Входы принимают сигнал в виде числа от нуля до единицы, при этом смысл их таков: если ингибиция равна единице, то поведение не должно работать, если нулю, то должно работать полностью. Соответственно, если активация равна единице, то наоборот, поведение должно работать, если нулю, то не должно (обычно их комбинируют как $a^*(1 - i)$, где *a* – это активация, *i* – ингибиция). Помимо выхода для данных есть еще выходы активности и целевого рейтинга. Это тоже числа от нуля до единицы; активность показывает, насколько поведение активно и готово влиять на состояние системы (с учетом активации и ингибиции), целевой рейтинг – насколько поведение «удовлетворено» тем, что происходит. Наличие такой структуры позволяет разделить поток данных и поток управления и реализовать сеть из влияющих друг на друга поведений без нужды в централизованной координации. Благодаря активностям и ингибициям поведения могут перехватывать управление (например, подсистема уклонения от столкновений может подавлять

управляющее воздействие от поведения более высокого уровня, обрабатывающего действия оператора пульта дистанционного управления). Возможна и более сложная логика комбинирования поведений, для этого применяется fusion behavior, специальный вид поведения, имеющий несколько входов данных, активаций и ингибиций, и выдающий результат согласно некоторой внутренней функции комбинирования этих сигналов.

Как архитектура i2BC интересна тем, что позволяет разрабатывать сложные системы, сохраняя их модульность, и интегрировать существующие алгоритмы, «оборачивая» их в поведение. Как формализм она интересна тем, что сводима к набору конечных автоматов, для которых оказываются применимы методы проверки моделей, как описано в работе [84]. Каждое поведение представляется в виде набора из пяти автоматов, каждый из которых отвечает за обработку своего входа или выхода. Автоматы связаны друг с другом синхронизационными каналами, по которым они могут обмениваться сообщениями. Автоматное представление системы может быть получено автоматически по модели в виде сети состояний. Авторы [84] проводили эксперименты по оценке влияния отказов датчиков на свойства системы управления мобильным роботом, в результате чего выяснилось, что некоторые свойства доказываются относительно эффективно (порядка секунд на обычном настольном компьютере), но некоторые вообще не доказываются за «разумное» время (за два дня на процессоре Intel XEON 2.7 ГГц с 1024 Гб оперативной памяти).

Формальная логика. Помимо верификации уже существующих систем, формальные методы применяются при нахождении корректных «по построению» и оптимальных решений. Еще один пример такого применения (помимо описанных выше в данной статье) – решение задачи о нахождении оптимального плана реконфигурации модульных роботов путем сведения ее к задаче SAT методами матлогики [85]. Оптимальный (в смысле количества шагов соединения-рассоединения) план для групп размером примерно в 400 роботов находился существующими решателями SAT за время порядка десятков минут.

Динамическая верификация

Рассмотренные ранее методы предназначаются прежде всего для верификации программы до запуска на реальном роботе или создания корректных «по построению» программ, но в реальных условиях этого оказывается недостаточно. Даже если удалось полностью верифицировать модель, необходимо еще убедиться, что модель соответствует реальному миру, в котором будет функционировать робот. Доказать в общем случае невозможно в силу непредсказуемости реального мира, но можно проверить во время выполнения и активировать аварийный механизм поведения, если данные о реальном мире перестали соответствовать предсказанным моделью.

Пример такой работы – средство ModelPlex, описанное в [86]. Верифицированная модель робота, программы управления и окружающей среды здесь уже даны, цель работы – сгенерировать код верификатора, который бы запускался на роботе каждый раз, когда вырабатывается новое управляющее воздействие, проверял, что параметры соответствуют предсказанным моделью и запускал бы аварийный режим управления, если не соответствуют.

Система гибридная, в том смысле, что робот непрерывно взаимодействует с окружающей реальностью, но управление и проверки происходят в дискретные моменты времени. Авторы используют дифференциальную динамическую логику для описания моделей и записи условия корректности. Управление системой рассматривается как циклическое исполнение программы управления и верификатора в некоторые моменты времени. Верификатор проверяет сначала переход из предыдущего состояния системы в текущее (описывается ли моделью изменение показаний датчиков за время, прошедшее с последнего запуска верификатора), затем корректность выработанного управляющего воздействия, затем предсказание на момент следующего запуска верификатора: будет ли выполнен

инвариант корректности модели с учетом выработанного управляющего воздействия и возможных отклонений реальности от модели. Разумеется, отклонения от модели должны быть ограничены. Например, если под роботом провалится пол, никакой верификатор не сможет гарантировать его безопасность. В случае, если вероятное отклонение модели от реального мира получается оценить, можно формально доказать корректность и своевременность переключения на аварийный сценарий. При этом сгенерированные верификаторы оказываются, по утверждению авторов, достаточно вычислительно просты, чтобы работать на роботе в реальном времени

Менее формальный метод обеспечения корректности предложен для системы ROS13 в [87]. Система ROS представляет программу в виде модулей (узлов), связанных каналами, по которым узлы обмениваются сообщениями. В работе предлагается вставлять между двумя взаимодействующими узлами монитор, который бы верифицировал сообщения, передающиеся по каналу, и в случае, если сообщения нарушают заданные ограничения, инициировать аварийное поведение. Монитор описывается на специальном языке, после чего генерируется код для ROS. В работе приводится пример с контролем угла наклона пейнтбольной пушки на роботе, чтобы подавить сигнал о выстреле, если робот может попасть в себя.

В настоящей статье приведен обзор использования формальных методов в робототехнике. Обзор проводился по материалам конференций ICRA и IROS последних лет, а также по данным Scopus и Google Scholar. Наиболее популярный и исследуемый подход на данный момент – постановка задач посредством LTL-спецификаций. Также активно ведутся исследования применения сетей Петри и марковских моделей. Есть и ряд альтернативных направлений, призванных избежать вычислительной сложности более популярных подходов, либо сложности специфицирования систем в таких формализмах. Также в контексте робототехники актуальна задача динамической верификации, когда корректность поведения робота контролируется в процессе его работы в реальном времени.

Следует отметить, что часто формальные методы применяются не для верификации, а для синтеза программы поведения робота по заданной модели среды и желаемым результатам. Это позволяет задавать поведение робота декларативно, и результирующая программа будет корректна по построению, однако такое построение часто требует больших вычислительных ресурсов.

Работа выполнена в рамках «Межвузовской проектной лаборатории робототехники» – совместного проекта компаний JetBrains и КиберТех Лабс.

СПИСОК ЛИТЕРАТУРЫ

1. **Kornfeld R.P., Prakash R., Devereaux A.S., et al.** Verification and validation of the Mars science laboratory/curiosity rover entry, descent, and landing system // J. of Spacecraft and Rockets. 2014. Vol. 51. No. 4. Pp. 1251–1269.

2. **Antoniotti M., Mishra B.** Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers // Robotics and Automation. Proc. IEEE Internat. Conf. 1995. Vol. 2. Pp. 1441–1446.

3. **Alpern B., Schneider F.B.** Defining liveness // Information processing letters. 1985. Vol. 21. No. 4. Pp. 181–185.

4. **Clarke E.M., Emerson E.A., Sistla A.P**. Automatic verification of finite-state concurrent systems using temporal logic specifications // ACM Transactions on Programming Languages and Systems. 1986. Vol. 8. No. 2. Pp. 244–263.

5. **Карпов Ю.Г.** Model Checking. Верификация параллельных и распределенных программных систем. СПб.: БХВ-Петербург, 2010.

6. **Clarke E.M, Grumberg O., Peled D.** Model checking. MIT press, 1999.

7. **Emerson E.A.** Temporal and modal logic // Handbook of Theoretical Computer Science. Vol. B: Formal Models and Semantics. 1990. Vol. 995. No. 1072. P. 5.

8. **Piterman N., Pnueli A., Sa'ar Ya.** Synthesis of reactive (1) designs // Verification, Model Checking, and Abstract Interpretation. Springer, 2006. Pp. 364–380.

9. **Pnueli A., Rosner R.** On the synthesis of a reactive module // Proc. of the 16th ACM SIGPLAN-SIGACT Symp. on Principles of

Programming Languages. 1989. Pp. 179–190.

10. **Kripke S.A.** Semantical analysis of modal logic i normal modal propositional calculi // Mathematical Logic Quarterly. 1963. Vol. 9. No. 5-6. Pp. 67–96.

11. **Fainekos G.E., Kress-Gazit H., Pappas G.J.** Temporal logic motion planning for mobile robots // Robotics and Automation. Proc. IEEE Internat. Conf. 2005. Pp. 2020–2025.

12. **Cimatti A., Clarke E., Giunchiglia E., et al.** NuSMV 2: An opensource tool for symbolic model checking // Computer Aided Verification. Springer, 2002. Pp. 359–364.

13. **Holzmann G.J.** The SPIN model checker: Primer and reference manual. Addison-Wesley Reading, 2003.

14. **Belta C., Habets L.C.G.J.M.** Constructing decidable hybrid systems with velocity bounds // Decision and Control. 43rd IEEE Conf. on. 2004. Vol. 1. Pp. 467–472.

15. **Kress-Gazit H., Fainekos G.E., Pappas G.J.** Where's Waldo? Sensor-based temporal logic motion planning // Robotics and Automation. IEEE Internat. Conf. on. 2007. Pp. 3116–3121.

16. **Kress-Gazit H., Fainekos G.E., Pappas G.J.** Temporal logic-based reactive mission and motion planning // Robotics. IEEE Transactions on. 2009. Vol. 25. No. 6. Pp. 1370–1381.

17. **Conner D.C., Rizzi A., Choset H., et al.** Composition of local potential functions for global robot control and navigation // Intelligent Robots and Systems. IEEE/RSJ Internat. Conf. on. 2003. Vol. 4. Pp. 3546–3551.

18. **Livingston S.C., Murray R.M.** Just-in-time synthesis for reactive motion planning with temporal logic // Robotics and Automation. IEEE Internat. Conf. on. 2013. Pp. 5048–5053.

19. **He K., Lahijanian M., Kavraki L.E., et al.** Towards manipulation planning with temporal logic specifications // Robotics and Automation. IEEE Internat. Conf. on. 2015. Pp. 346–352.

20. **Srivastava S., Fang E., Riano L., et al.** Combined task and motion planning through an extensible planner independent interface layer // Robotics and Automation. IEEE Internat. Conf. on. 2014. Pp. 639–646.

21. **Raman V., Kress-Gazit H.** Synthesis for multi-robot controllers with interleaved motion // Robotics and Automation. IEEE Internat. Conf. on. 2014. Pp. 4316–4321.

22. **Vasile Cristian Ioan, Belta Calin.** Reactive sampling-based temporal logic path planning // Robotics and Automation. IEEE Internat. Conf. on. 2014. Pp. 4310–4315.

23. **Wolff E.M., Topcu U., Murray R.M.** Efficient reactive controller synthesis for a fragment of linear temporal logic // Robotics and Automation. IEEE Internat. Conf. on. 2013. Pp. 5033–5040.

24. **Бугайченко Д.Ю.** Разработка и реализация методов формально-логической спецификации самонастраивающихся мультиагентных систем с временными ограничениями // Сайт математико-механического факультета СПбГУ. СПб.: СПбГУ, 2007. Т. 2010.

25. **Plünnecke H., Reisig W.** Bibliography of Petri nets 1990. Advances on I'rtri Net.s 1991 // Lecture Notes in Computer Science. 1991. Vol. 524. Pp. 312–572.

26. **Yen Hsu Chun.** Introduction to Petri net theory // Studies in Computational Intelligence. 2006. Vol. 25. Pp. 343–373.

27. **Murata Tadao.** Petri nets: Properties, analysis and applications // Proc. of the IEEE. 1989. Vol. 77. No. 4. Pp. 541–580.

28. **Esparza J., Nielsen M.** Decidability Issues for Petri Nets-a Survey // Bulletin of the European Association for Theoretical Computer Science. 1994. Vol. 52. Pp. 245–262.

29. **Zurawski R., Zhou Meng Chu.** Petri nets and industrial applications: A tutorial // IEEE Transactions on Industrial Electronics. 1994. Vol. 41. No. 6. Pp. 567–583.

30. **Мартынов В.Г., Масягин В.Б.** Применение сетей Петри при моделировании управления технологическими процессами сборочного производства // Омский научный вестник. 2014. № 1 (127). С. 134–137.

31. **Costelha H., Lima P.** Robotic Tasks Modeling and Analysis Based on Petri Nets. 2003.

32. **Costelha H., Lima P** Petri net robotic task plan representation: Modelling, analysis and execution // Autonomous Agents. 2010. Pp. 65–91.

33. **Конюх В.Л.** Опыт применения сетей Петри для имитации поведения систем // Имитационное моделирование. Теория и практика. ИММОД-2009. С. 27–37.

34. **Aguiar A.J.C., Villani E.** Petri Nets and Graphic Simulation for the Validation of Collaborative Robotic Cells in Aircraft Industry // ABCM Symp. Ser. in Mechatronics. 2010. Vol. 4. Pp. 364–373.

35. **Fu Yujian, Drabo Mebougna**. Formal Modeling and Verification of Dynamic Reconfiguration of Autonomous Robotics Systems // Proc. of the Internat. Conf. on Embedded Systems and Applications (ESA). 2014.

36. **Simon Jochen, Moldt Daniel**. PyTri, a Visual Agent Programming Language // Algorithmen und Werkzeuge für Petrinetze. 2010. Vol. 643. Pp. 106–111.

37. **Kashima Hideharu, Masuda Ryosuke**.

Cooperative Control of Mobile Roots Based on Petri Net // The 10th Internat. Conf. on Advanced Robotics. 2001. Pp. 399–404.

38. **Palamara F., Ziparo V.A. Locchi L., et al.** A Robotic Soccer Passing Task Using Petri Net Plans (Demo Paper) // Proc. of the 7th Internat. Joint Conf. on Autonomous Agents and Multiagent Systems. 2008. Pp. 1711–1712.

39. **Kotb Y.T., Beauchemin S.S., Barron J.L.** Petri Net-Based Cooperation In Multi-Agent Systems // Computer and Robot Vision. 4th Canadian Conf. on. 2007. Pp. 123–130.

40. **Chang Guoting Jane, Kulirc Dana.** Robot Task Learning from Demonstration Using Petri Nets // The 22nd IEEE Internat. Symp. on Robot and Human Interactive Communication. Gyeongju, Korea. 2013. Pp. 31–36.

41. **Baeten J.C.M.** A brief history of process algebra // Theoretical Computer Science. 2005. Vol. 335. No. 2. Pp. 131–146.

42. **Fokkink Wan.** Introduction to process algebra. Springer Science & Business Media, 2013.

43. **Hoare C.A.R.** Communicating sequential processes // Communications of the ACM. 1978. Vol. 21. No. 8. Pp. 666–677.

44. **Brookes S.D., Hoare C.A.R., Roscoe A.W.** A theory of communicating sequential processes // J. of the ACM. 1984. Vol. 31. No. 3. Pp. 560–599.

45. **Hoare C.A.R.** Communicating Sequential Processes. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1985.

46. **Welch P.H., Barnes F.R.M.** Communicating mobile processes // Communicating Sequential Processes. The 1st 25 Years. Springer, 2005. Pp. 175–210.

47. **Lankenau A., Meyer O.** Formal methods in robotics: Fault tree based verification // Proc. of Quality Week. Citeseer, 1999.

48. **Vesely W.E., Goldberg F.F., Roberts N.H., et al.** Fault tree handbook: Tech. Rep. DTIC Document, 1981.

49. **Hansen K.M.** Linking safety analysis to safety requirements: exemplified by railway interlocking systems. Institut for Informationsteknologi. DTU, 1996.

50. **Simpson J., Jacobsen C.L.** Visual Process-Oriented Programming for Robotics // CPA. 2008. Pp. 365–380.

51. **Кознов Д.В.** Основы визуального моделирования. М.: БИНОМ, Лаборатория знаний, 2008.

52. **Posso J.C., Sampson A.T., Simpson J., et al.** Process-Oriented Subsumption Architectures in Swarm Robotic Systems // CPA. 2011. Pp. 303–316.

53. **Sahin Erol.** Swarm robotics: From sources of inspiration to domains of application // Swarm robotics. Springer, 2005. Pp. 10–20.

54. **Brooks R.** A robust layered control system for a mobile robot // IEEE Journal on Robotics and Automation. 1986. Vol. 2. No. 1. Pp. 14–23.

55. **Varricchio V., Chaudhari P., Frazzoli E.** Sampling-based algorithms for optimal motion planning using process algebra specifications // Robotics and Automation. IEEE Internat. Conf. on. 2014. Pp. 5326–5332.

56. **Vaandrager F.W, van Schuppen J.H.** Hybrid Systems: Computation and Control // Proc. of the 2nd Internat. Workshop. HSCC'99. Berg en Dal, The Netherlands. Springer Science & Business Media, 1999. No. 1569.

57. **Lynch N., Krogh B.** Hybrid Systems: Computation and Control // Proc. of the 3rd Internat. Workshop. HSCC 2000. Pittsburgh, PA, USA. Springer Science & Business Media, 2007.

58. **Henzinger T.A., Ho Pei-Hsin, Wong-Toi H.** HyTech: A model checker for hybrid systems // Computer aided verification. Springer, 1997. Pp. 460–463.

59. **Frehse Goran**. PHAVer: Algorithmic verification of hybrid systems past HyTech // Hybrid Systems: Computation and Control. Springer, 2005. Pp. 258–273.

60. **Platzer A., Quesel J.-D.** KeYmaera: A hybrid theorem prover for hybrid systems (system description) // Automated Reasoning. Springer, 2008. Pp. 171–178.

61. **Alur R., Courcoubetis C., Halbwachs N., et al.** The algorithmic analysis of hybrid systems // Theoretical computer science. 1995. Vol. 138. No. 1. Pp. 3–34.

62. **Xi Zheng, Christine Julien, Miryung Kim, et al.** Perceptions on the State of the Art in Verification and Validation in Cyber-Physical Systems // Systems Journal, IEEE. Los-Alamitos, CA. Pp. 1–14.

63. **Matthew R.M., Lahijanian M., Kavraki L.E., et al.** Iterative temporal motion planning for hybrid systems in partially unknown environments // Proc. of the 16th Internat. Conf. on Hybrid Systems: Computation and Control. NY, USA, 2013. Pp. 353–362.

64. **Song M., Tarn T.-J., Xi N.** Integration of task scheduling, action planning, and control in robotic manufacturing systems // Proc. of the IEEE. Los-Alamitos, CA, 2000. Vol. 88. No. 7. Pp. 1097–1107.

65. **Mitsch S., Passmore G.O., Platzer A.** Collaborative Verification-Driven Engineering of Hybrid Systems // Mathematics in Computer Science. Basel, 2014. Vol. 8. Pp. 71–97.

66. **Сорокин А.В., Кознов Д.В.** Обзор Eclipse

Modeling Project // Системное программирование. СПб., 2010. Т. 5. № 1. С. 6–32.

67. **Майн Х.** Марковские процессы принятия решений. М., 1977.

68. **Kolobov A.** Planning with Markov decision processes: An AI perspective // Synthesis Lectures on Artificial Intelligence and Machine Learning. 2012. Vol. 6. No. 1. Pp. 1–210.

69. **Pentland A., Liu A.** Modeling and prediction of human behavior // Neural computation. 1999. Vol. 11. No. 1. Pp. 229–242.

70. **Rothkopf C.A, Ballard D.H.** Modular inverse reinforcement learning for visuomotor behavior // Biological cybernetics. 2013. Vol. 107. No. 4. Pp. 477–490.

71. **McGhan C.L.R., Nasir A., Atkins E.M.** Human intent prediction using markov decision processes // J. of Aerospace Information Systems. 2015. Pp. 1–5.

72. **Baier C., Katoen J.-P., et al.** Principles of model checking. Cambridge: MIT press, 2008. Pp. 2620–2649.

73. **Svorenova M., Cerna I., Belta C.** Optimal control of MDPs with temporal logic constraints // Decision and Control. IEEE 52nd Annual Conf. on. 2013. Pp. 3938–3943.

74. **Bertsekas D.P., et al.** Dynamic programming and optimal control. Athena Scientific Belmont, MA, 1995. Vol. 1.

75. **Lacerda B., Parker D., Hawes N.** Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications // Intelligent Robots and Systems. IEEE/RSJ Internat. Conf on. 2014. Pp. 1511–1516.

76. **Fu Jie, Topcu Ufuk**. Pareto efficiency in synthesizing shared autonomy policies with temporal logic constraints // arXiv preprint arXiv:1412.6029. 2014.

77. **Baier C., Grosser M., Bertrand N.** Probabilistic omega-automata // J. of the ACM. 2012. Vol. 59. No. 1. P. 1.

78. **Chatterjee K., Chmelik M., Tracol M.** What is Decidable about Partially Observable Markov Decision Processes with omega-Regular Objectives // arXiv preprint arXiv:1309.2802. 2013.

79. **Chatterjee K., Chmelik M., Gupta R., et al.** Qualitative analysis of pomdps with temporal logic specifications for robotics applications // arXiv preprint arXiv:1409.3360. 2014.

80. **Svorěnov´a M., Chmelik M., Leahy K., et al.** Temporal logic motion planning using POMDPs with parity objectives: case study paper // Proc. of the 18th Internat. Conf. on Hybrid Systems: Computation and Control. ACM. 2015. Pp. 233–238.

81. **Lyons D., Arkin R., Jiang S., et al.** Performance Verification for Behavior-based Robot Missions // IEEE Transactions on Robotics. 2015. Vol. 31. No. 3. Pp. 619–636.

82. **Damian M.L., Arkin R.C., Nirmal P., et al.** A Software Tool for the Design of Critical Robot Missions with Performance Guarantees // Procedia Computer Science. 2013. Vol. 16. C. 888–897.

83. **Proetzsch M., Luksch T., Berns K.** The Behaviour-Based Control Architecture iB2C for Complex Robotic Systems. 2007.

84. **Kiekbusch L., Armbrust C., Berns K.** Formal Verification of Behaviour Networks Including Hardware Failures // Proc. of the 13th Internat. Conf. on Intelligent Autonomous Systems. 2014.

85. **Gorbenko A.A., Popov V.Yu.** Programming for modular reconfigurable robots // Programming and Computer Software. 2012. Vol. 38. No. 1. Pp. 13–23.

86. **Mitsch S., Platzer A.** ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models // Runtime Verification. 2014. Pp. 199–214.

87. **Huang J., Erdogan C., Zhang Yi, et al.** ROSRV: Runtime Verification for Robots // Runtime Verification. 2014. Pp. 247–254.

REFERENCES

1. **Kornfeld R.P., Prakash R., Devereaux A.S., et al.** Verification and validation of the Mars science laboratory/curiosity rover entry, descent, and landing system. *Journal of Spacecraft and Rockets*, 2014, Vol. 51, No. 4, Pp. 1251–1269.

2. **Antoniotti M., Mishra B.** Discrete event models + temporal logic=supervisory controller: Automatic synthesis of locomotion controllers. *Robotics and Automation, Proceedings IEEE International Conference*, 1995, Vol. 2, Pp. 1441–1446.

3. **Alpern B., Schneider F.B.** Defining liveness. *Information processing letters*, 1985, Vol. 21, No. 4, Pp. 181–185.

4. **Clarke E.M., Emerson E.A., Sistla A.P**. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 1986, Vol. 8, No. 2, Pp. 244–263.

5. **Karpov Yu.G.** *Model Checking. Verifikatsiya parallelnykh i raspredelennykh programmnykh sistem* [*Model Checking. Verification of parallel and distributed software systems*], St. Petersburg: BKhV-Petersburg Publ., 2010. (rus)

6. **Clarke E.M, Grumberg O., Peled D.** *Model*

checking. MIT press, 1999.

7. **Emerson E.A.** Temporal and modal logic. *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, 1990, Vol. 995, No. 1072, P. 5.

8. **Piterman N., Pnueli A., Sa'ar Ya.** Synthesis of reactive (1) designs. *Verification, Model Checking, and Abstract Interpretation*. Springer, 2006, Pp. 364–380.

9. **Pnueli A., Rosner R.** On the synthesis of a reactive module. *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1989, Pp. 179–190.

10. **Kripke S.A.** Semantical analysis of modal logic i normal modal propositional calculi. *Mathematical Logic Quarterly*, 1963, Vol. 9, No. 5-6, Pp. 67–96.

11. **Fainekos G.E, Kress-Gazit H., Pappas G.J.** Temporal logic motion planning for mobile robots. *Robotics and Automation, Proceedings of the IEEE International Conference*, 2005, Pp. 2020–2025.

12. **Cimatti A., Clarke E., Giunchiglia E., et al.** NuSMV 2: An opensource tool for symbolic model checking. *Computer Aided Verification*. Springer, 2002, Pp. 359–364.

13. **Holzmann G.J.** *The SPIN model checker: Primer and reference manual*. Addison-Wesley Reading, 2003.

14. **Belta C., Habets L.C.G.J.M.** Constructing decidable hybrid systems with velocity bounds. *Decision and Control. Proceedings of the 43rd IEEE Conference on*, 2004, Vol. 1, Pp. 467–472.

15. **Kress-Gazit H., Fainekos G.E., Pappas G.J.** Where'sWaldo? Sensor-based temporal logic motion planning. *Robotics and Automation, IEEE International Conference on*, 2007. Pp. 3116–3121.

16. **Kress-Gazit H., Fainekos G.E., Pappas G.J.** Temporal logic-based reactive mission and motion planning. *Robotics, IEEE Transactions on*, 2009, Vol. 25, No. 6, Pp. 1370–1381.

17. **Conner D.C., Rizzi A., Choset H., et al.** Composition of local potential functions for global robot control and navigation. *Intelligent Robots and Systems*, *Proceedings of the 2003 IEEE/ RSJ International Conference on*. 2003, Vol. 4, Pp. 3546–3551.

18. **Livingston S.C., Murray R.M.** Just-in-time synthesis for reactive motion planning with temporal logic. *Robotics and Automation*, *IEEE International Conference on*, 2013, Pp. 5048–5053.

19. **He K., Lahijanian M., Kavraki L.E., et al.** Towards manipulation planning with temporal logic specifications. *Robotics and Automation*, *IEEE International Conference on*, 2015, Pp. 346–352.

20. **Srivastava S., Fang E., Riano L., et al.** Combined task and motion planning through an extensible planner independent interface layer. *Robotics and Automation*, *IEEE International Conference on*, 2014, Pp. 639–646.

21. **Raman V., Kress-Gazit H.** Synthesis for multi-robot controllers with interleaved motion. *Robotics and Automation*, *IEEE International Conference on*, 2014, Pp. 4316–4321.

22. **Vasile Cristian Ioan, Belta C.** Reactive sampling-based temporal logic path planning. *Robotics and Automation, IEEE International Conference on*, 2014, Pp. 4310–4315.

23. **Wolff E.M., Topcu U., Murray R.M.** Efficient reactive controller synthesis for a fragment of linear temporal logic. *Robotics and Automation*, *IEEE International Conference on*, 2013, Pp. 5033–5040.

24. **Bugaychenko D.Yu.** Razrabotka i realizatsiya metodov formalno-logicheskoy spetsifikatsii samonastraivayushchikhsya multiagentnykh sistem s vremennymi ogranicheniyami [Development and implementation of methods of formal logic specification self-adjusting multi-agent systems with time constraints]. *Sayt matematiko-mekhanicheskogo fakulteta SPbGU* [*Website of the Faculty of Mathematics and Mechanics St. Petersburg State University*]. SPbGU, 2007, Vol. 2010. (rus)

25. **Plünnecke H., Reisig W.** Bibliography of Petri nets 1990. Advances on I'rtri Net.s 1991, *Lecture Notes in Cornputer Science*, 1991, Vol. 524. Pp. 312–572.

26. **Yen Hsu Chun.** Introduction to Petri net theory. *Studies in Computational Intelligence*, 2006, Vol. 25, Pp. 343–373.

27. **Murata Tadao.** Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 1989, Vol. 77, No. 4, Pp. 541–580.

28. **Esparza J., Nielsen M.** Decidability Issues for Petri Nets-a Survey. *Bulletin of the European Association for Theoretical Computer Science*, 1994, Vol. 52, Pp. 245–262.

29. **Zurawski R., Zhou Meng Chu.** Petri nets and industrial applications: A tutorial. *IEEE Transactions on Industrial Electronics*, 1994, Vol. 41, No. 6, Pp. 567–583.

30. **Martynov V.G., Masyagin V.B.** Primeneniye setey Petri pri modelirovanii upravleniya tekhnologicheskimi protsessami sborochnogo proizvodstva [Application of Petri network for modeling of process control of assembly production]. *Omskiy nauchnyy vestnik* [*Omsk Scientific Bulletin*], 2014, No. 1 (127), Pp. 134–137. (rus)

31. **Costelha H., Lima P.** *Robotic Tasks Modeling and Analysis Based on Petri Nets*, 2003.

32. **Costelha H., Lima P** Petri net robotic task plan representation: Modelling, analysis and

execution. *Autonomous Agents*, 2010, Pp. 65–91.

33. **Konyukh V.L.** Opyt primeneniya setey Petri dlya imitatsii povedeniya system [Experience of using Petri nets to simulate the behavior of systems]. *Imitatsionnoye modelirovaniye. Teoriya i praktika* [*Simulation. Theory and practice. IMMOD-2009*], 2009, Pp. 27–37. (rus)

34. **Aguiar A.J.C., Villani E.** Petri Nets and Graphic Simulation for the Validation of Collaborative Robotic Cells in Aircraft Industry. *ABCM Symposium Series in Mechatronics*, 2010, Vol. 4, Pp. 364–373.

35. **Fu Yujian, Drabo Mebougna**. Formal Modeling and Verification of Dynamic Reconfiguration of Autonomous Robotics Systems. *Proceedings of the International Conference on Embedded Systems and Applications (ESA)*, 2014.

36. **Simon Jochen, Moldt Daniel**. PyTri, a Visual Agent Programming Language. *Algorithmen und Werkzeuge für Petrinetze*, 2010, Vol. 643, Pp. 106–111.

37. **Kashima Hideharu, Masuda Ryosuke**. Cooperative Control of Mobile Roots Based on Petri Net. *The 10th International Conference on Advanced Robotics*, 2001, Pp. 399–404.

38. **Palamara F., Ziparo V.A. Locchi L., et al.** A Robotic Soccer Passing Task Using Petri Net Plans (Demo Paper). *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008, Pp. 1711–1712.

39. **Kotb Y.T., Beauchemin S.S., Barron J.L.** Petri Net-Based Cooperation In Multi-Agent Systems. *Computer and Robot Vision, Proceedings of the 4th Canadian Conference on*, 2007, Pp. 123–130.

40. **Chang Guoting Jane, Kulirc Dana.** Robot Task Learning from Demonstration Using Petri Nets. *The 22nd IEEE International Symposium on Robot and Human Interactive Communication,* Gyeongju, Korea, 2013, Pp. 31–36.

41. **Baeten J.C.M.** A brief history of process algebra. *Theoretical Computer Science*, 2005, Vol. 335, No. 2, Pp. 131–146.

42. **Fokkink Wan**. *Introduction to process algebra*. Springer Science & Business Media, 2013.

43. **Hoare C.A.R.** Communicating sequential processes, *Communications of the ACM*, 1978, Vol. 21, No. 8, Pp. 666–677.

44. **Brookes S.D., Hoare C.A.R., Roscoe A.W.** A theory of communicating sequential processes. *Journal of the ACM*, 1984, Vol. 31, No. 3, Pp. 560–599.

45. **Hoare C.A.R.** *Communicating Sequential Processes*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1985.

46. **Welch P.H., Barnes F.R.M.** Communicating mobile processes. *Communicating Sequential Processes. The 1st 25 Years*. Springer, 2005, Pp. 175–210.

47. **Lankenau A., Meyer O.** Formal methods in robotics: Fault tree based verification, *Proc. of Quality Week*, Citeseer, 1999.

48. **Vesely W.E., Goldberg F.F., Roberts N.H., et al.** *Fault tree handbook*: *Tech. Rep.*, DTIC Document, 1981.

49. **Hansen K.M.** *Linking safety analysis to safety requirements: exemplified by railway interlocking systems*. Institut for Informationsteknologi, DTU, 1996.

50. **Simpson J., Jacobsen C.L.** Visual Process-Oriented Programming for Robotics. *CPA*, 2008, Pp. 365–380.

51. **Koznov D.V.** *Osnovy vizualnogo modelirovaniya* [*Fundamentals of Visual Modeling*]. Moscow: BINOM, Laboratoriya znaniy Publ., 2008. (rus)

52. **Posso J.C., Sampson A.T., Simpson J., et al.** Process-Oriented Subsumption Architectures in Swarm Robotic Systems. *CPA,* 2011, Pp. 303–316.

53. **Sahin Erol.** Swarm robotics: From sources of inspiration to domains of application. *Swarm robotics*, Springer, 2005, Pp. 10–20.

54. **Brooks R.** A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation,* 1986, Vol. 2, No. 1, Pp. 14–23.

55. **Varricchio V., Chaudhari P., Frazzoli E.** Sampling-based algorithms for optimal motion planning using process algebra specifications. *Robotics and Automation, IEEE International Conference on,* 2014, Pp. 5326–5332.

56. **Vaandrager F.W, van Schuppen J.H.** Hybrid Systems: Computation and Control. *Proceedings of the 2nd International Workshop, HSCC'99*, Berg en Dal, The Netherlands. Springer Science & Business Media, 1999, No. 1569.

57. **Lynch N., Krogh B.** Hybrid Systems: Computation and Control. *Proceedings of the 3rd Internat. Workshop, HSCC 2000*, Pittsburgh, PA, USA. Springer Science & Business Media, 2007.

58. **Henzinger T.A., Ho Pei-Hsin, Wong-Toi H.** HyTech: A model checker for hybrid systems. *Computer aided verification*. Springer, 1997, Pp. 460–463.

59. **Frehse Goran**. PHAVer: Algorithmic verification of hybrid systems past HyTech. *Hybrid Systems: Computation and Control*. Springer, 2005, Pp. 258–273.

60. **Platzer A., Quesel J.-D.** KeYmaera: A hybrid theorem prover for hybrid systems (system description). *Automated Reasoning*. Springer, 2008, Pp. 171–178.

61. **Alur R., Courcoubetis C., Halbwachs N., et al.** The algorithmic analysis of hybrid systems. *Theoretical computer science*. 1995, Vol. 138, No. 1, Pp. 3–34.

62. **Xi Zheng, Christine Julien, Miryung Kim, et al.** Perceptions on the State of the Art in Verification and Validation in Cyber-Physical Systems. *Systems Journal, IEEE*. Los-Alamitos, CA, Pp. 1–14.

63. **Matthew R.M., Lahijanian M., Kavraki L.E., et al.** Iterative temporal motion planning for hybrid systems in partially unknown environments. *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*. New York, USA, 2013, Pp. 353–362.

64. **Song M., Tarn T.-J., Xi N.** Integration of task scheduling, action planning, and control in robotic manufacturing systems. *Proceedings of the IEEE*. Los-Alamitos, CA, 2000, Vol. 88, No. 7, Pp. 1097–1107.

65. **Mitsch S., Passmore G.O., Platzer A.** Collaborative Verification-Driven Engineering of Hybrid Systems. *Mathematics in Computer Science*. Basel, 2014, Vol. 8, Pp. 71–97.

66. **Sorokin A.V., Koznov D.V.** Obzor Eclipse Modeling Project [Eclipse Modeling Project Overview]. *Sistemnoye programmirovaniye* [*System Programming*]. St. Petersburg, 2010, Vol. 5, No. 1, Pp. 6–32. (rus)

67. **Mayn Kh.** *Markovskiye protsessy prinyatiya resheniy* [*Markov decision processes*]. Moscow, 1977. (rus)

68. **Kolobov A.** Planning with Markov decision processes: An AI perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2012, Vol. 6, No. 1, Pp. 1–210.

69. **Pentland A., Liu A.** Modeling and prediction of human behavior. *Neural computation*, 1999, Vol. 11, No. 1, Pp. 229–242.

70. **Rothkopf C.A, Ballard D.H.** Modular inverse reinforcement learning for visuomotor behavior. *Biological cybernetics*, 2013, Vol. 107, No. 4, Pp. 477–490.

71. **McGhan C.L.R., Nasir A., Atkins E.M.** Human intent prediction using markov decision processes. *Journal of Aerospace Information Systems*, 2015, Pp. 1–5.

72. **Baier C., Katoen J.-P., et al.** *Principles of model checking*. Cambridge: MIT press, 2008, Pp. 2620–2649.

73. **Svorenova M., Cerna I., Belta C.** Optimal control of MDPs with temporal logic constraints. *Decision and Control (CDC), IEEE 52nd Annual Conference on*, 2013, Pp. 3938–3943.

74. **Bertsekas D.P., et al.** *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995, Vol. 1.

75. **Lacerda B., Parker D., Hawes N.** Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. *Intelligent Robots and Systems, IEEE/RSJ International Conference on,* 2014, Pp. 1511–1516.

76. **Fu Jie, Topcu Ufuk**. *Pareto efficiency in synthesizing shared autonomy policies with temporal logic constraints,* arXiv preprint arXiv:1412.6029, 2014.

77. **Baier C., Grosser M., Bertrand N.** Probabilistic omega-automata. *Journal of the ACM*, 2012, Vol. 59, No. 1, P. 1.

78. **Chatterjee K., Chmelik M., Tracol M.** *What is Decidable about Partially Observable Markov Decision Processes with omega-Regular Objectives*, arXiv preprint arXiv:1309.2802, 2013.

79. **Chatterjee K., Chmelik M., Gupta R., et al.** *Qualitative analysis of pomdps with temporal logic specifications for robotics applications*, arXiv preprint arXiv:1409.3360, 2014.

80. **Svorěnov´a M., Chmelik M., Leahy K., et al.** Temporal logic motion planning using POMDPs with parity objectives: case study paper. *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, ACM, 2015, Pp. 233–238.

81. **Lyons D., Arkin R., Jiang S., et al.** Performance Verification for Behavior-based Robot Missions. *IEEE Transactions on Robotics*, 2015. Vol. 31, No. 3, Pp. 619–636.

82. **Lyons D.M., Arkin R.C., Nirmal P., et al.** A Software Tool for the Design of Critical Robot Missions with Performance Guarantees. *Procedia Computer Science*, 2013, Vol. 16, Pp. 888–897.

83. **Proetzsch M., Luksch T., Berns K.** *The Behaviour-Based Control Architecture iB2C for Complex Robotic Systems*, 2007.

84. **Kiekbusch L., Armbrust C., Berns K.** Formal Verification of Behaviour Networks Including Hardware Failures. *Proceedings of the 13th International Conference on Intelligent Autonomous Systems,* 2014.

85. **Gorbenko A.A., Popov V.Yu.** Programming for modular reconfigurable robots. *Programming and Computer Software*, 2012, Vol. 38, No. 1, Pp. 13–23

86. **Mitsch S., Platzer A.** ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models. *Runtime Verification*, 2014, Pp. 199–214.

87. **J. Huang, C. Erdogan, Yi Zhang, et al.** ROSRV: Runtime Verification for Robots. *Runtime Verification*, 2014, Pp. 247–254.

МОРДВИНОВ Дмитрий Александрович – *аспирант кафедры системного программирования математико-механического факультета Санкт-Петербургского государственного университета.*
199034, Россия, Санкт-Петербург, Университетская наб., д. 7-9.
E-mail: mordvinov.dmitry@gmail.com

MORDVINOV Dmitrii A. *St. Petersburg State University.*
199034, Universitetskaya emb., 7-9, St. Petersburg, Russia.
E-mail: mordvinov.dmitry@gmail.com

ЛИТВИНОВ Юрий Викторович – *старший преподаватель кафедры системного программирования математико-механического факультета Санкт-Петербургского государственного университета.*
199034, Россия, Санкт-Петербург, Университетская наб., д. 7-9.
E-mail: yurii.litvinov@gmail.com

LITVINOV Yurii V. *St. Petersburg State University.*
199034, Universitetskaya emb., 7-9, St. Petersburg, Russia.
E-mail: yurii.litvinov@gmail.com