Формальные методы в робототехнике

Д.А. Мордвинов
Санкт-Петербургский государственный университет
Кафедра системного программирования
Email: mordvinov.dmitry@gmail.com

Ю.В. Литвинов
Санкт-Петербургский государственный университет
Кафедра системного программирования
Email: y.litvinov@spbu.ru

***Аннотация*—В статье дается обзор применения формальных методов в контексте робототехники. Рассматриваются недавние работы, посвященные спецификациям поведения роботов в терминах темпоральных логик, применению идей подхода model checking к таким системам. Также рассматриваются применения формальных методов анализа сетей Петри и моделирования поведения робототехнических систем с их помощью. Отдельное внимание уделяется верификации гибридных систем, применению хоаровского исчисления процессов для спецификации поведения параллельных систем, а также использованию других подходов для верификации программ поведения роботов.**

I. Введение

Разговор о формальных методах чаще всего 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 каждом 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 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]. Будут также рассмотрены случаи применения формальных методов в реабилитационной робототехнике, к созданию автономных промышленных робототехнических систем, робофутболу и т.д.

Задачей данной статьи является обзор и классификация применения формальных методов к проблемам робототехники. Будут рассмотрены как самые известные работы середины девяностых годов, положившие начало целым областям исследований, так и работы с последних робототехнических конференций вплоть до 2015 года. Авторы не претендуют 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.

II. Model Checking

В 1995 году в статье [1] 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

[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)

[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)

(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 [2]) авторы используют темпоральную логику ветвящегося времени (CTL, [3]). Исторически это первый пример успешного использования темпоральной логики в контексте робототехнических систем. Техники model checking были применены в работе в своем «стандартном» виде: по заданной модели поведения системы и ее инварианту была проведена формальная верификация соответствия поведения системы требованиям.

Авторы рекомендуют читателю быть ознакомленным с [4] или с [2] для понимания материала данного раздела. Введем понятия и обозначения, используемые далее в статье. Для спецификации формул темпоральной логики мы будем использовать стандартную нотацию: $\bigcirc\phi$ для оператора *next*, $\Box\phi$ для оператора *always*, $\Diamond\phi$ для оператора *eventually* и $\phi\mathcal{U}\psi$ для оператора *until*. Множество формул темпоральной логики линейного времени (LTL, [5]) вида

$$(\Box\Diamond p_1 \wedge \cdots \wedge \Box\Diamond p_m) \Rightarrow (\Box\Diamond q_1 \wedge \cdots \wedge \Box\Diamond q_n)$$

будем называть, в соответствии с [6], классом *General Reactivity (1)*, или *GR(1)*. Данный класс формул LTL примечателен тем, что по его представителю ϕ можно построить 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 [7]. Структурой Крипке [8] 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 году авторы из лаборатории GRASP Пенсильванского Университета представили работы, определившие текущее состояние области применения формальных методов в робототехнике. В последние 3 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 [9] 2005 года 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 «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, $L(s_i) =^{def} \pi_i$.

Основная идея работы — постановка задачи роботу декларативно в виде LTL-спецификации. К примеру, формула $\phi = \Diamond(\pi_2 \wedge \Diamond(\pi_3 \wedge \Diamond(\pi_4 \wedge (\neg\pi_2 \wedge \neg\pi_3) \cup \pi_1)))$ 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 $\neg\phi$ на модели D. В статье проводятся эксперименты с использованием верификаторов NuSMV [10] и SPIN [11]. Полученная дискретная траектория робота затем преобразуется в непрерывное управление при помощи методов теории управления, описанных в [12].

Подход, описанный в [9] может показаться непрактичным и чересчур усложненным для решаемой задачи, однако идеи, на основе которых он строится, легли в основу действительно полезного метода. Речь идет о статье 2007 года [13]. 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 $X = \{x_1, \ldots, x_m\}$ — 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, так как каждый робот представляет собой часть окружения другого.

Очевидно, что дискретное управление системой в случае такой модели уже не описывается одной траекторией на «карте» окружения, так как необходимо реагировать на разное поведение среды по-разному. Ав-

торы 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 $\phi = \phi_e \Rightarrow \phi_s$, 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 авторами такой модели. Утверждается, что можно посмотреть на процесс синтеза дискретного управления как на игру 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 обязана «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 года [14] 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 [15]. Другой результат, полученный в [14] — формализация действий робота в общем виде. Множество атомарных предикатов расширяется множеством $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$ 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 $\mathcal{A}$ вместе с такими действиями, как, например, захват предмета клешней или включение/выключение камеры.

Достоинства описанного подхода очевидны: по декларативной спецификации требований к системе в достаточно 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, скорее всего, в описанном виде модель не применима в случае большого количества регионов, роботов, датчиков и действий, выполняемых системой, так как синтез управления может происходить очень долго (зато в результате получается готовый алгоритм поведения системы, т.е. синтез можно произвести лишь один раз).

Шаги к решению как минимум второй проблемы были проделаны в работе [16] 2013 года, 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 шагу сравнится с трудоемкостью алгоритма построения автомата синтеза полной стратегии, применяемым раннее. Работоспособность такого подхода объясняется следующим фактом. В статье [6] 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. При этом трудоемким является второй шаг, в то время как первый работает быстро. Обсуждаемый алгоритм принятия решения в реальном времени использует только вычисление неподвижной точки, не проводя синтеза автомата, что и дает значительный 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 роботом в режиме реального времени, такой подход позво-

ляет системе принять полностью реактивный характер. В частности, становится возможным объезд движущегося препятствия, что раннее было слабо затронуто в статьях на тему планирования поведения по LTL-спецификации.

Существует множество статей с экспериментами, поставленными на теоретической основе обсуждаемого подхода. Например, в [17] 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 [18] (однако бэктрекинг работал не во всех ситуациях и был крайне неэффективен во временном и ресурсном отношении). Работа [19] описывает планирование для мультиагентной системы раздельного сбора отходов. В работе [20] обсуждается планирование поведения робота-спасателя, исследующего место катастрофы с целью тушения пожаров и предоставления первой помощи выжившим.

Следует отметить, что количество опубликованных работ на тему планирования поведения робота при помощи LTL-спецификации на момент 2015 года имеет порядок сотен, поэтому авторы считают уместным ограничиться здесь лишь рассмотренными статьями, так как они дают общую характеристику подхода. Актуальные статьи на данную тему можно найти, в частности, в списках докладов секций формальных методов конференций ICRA и IROS.

III. Сети Петри

Один из самых старых и некогда один из самых популярных методов формального анализа программ — сети Петри. Впервые предложенные Карлом Петри в 1962 году, они предназначались прежде всего для моделирования и анализа параллельных и распределённых 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 [21], опубликованной в 1991 году, приводится библиография из 4099 статей, рассматривающих сети Петри или их применения. Мы здесь рассмотрим лишь некоторые применения сетей Петри в робототехнике.

Сетью Петри мы, следуя [22], будем называть тройку (P, T, ϕ), где P — конечное множество позиций, T — конечное множество переходов, а $\phi : (P \times T) \cup (T \times P) \to N$ — функция потока. Маркировкой называется отображение $\mu : P \to N$, ставящее в соответствие позиции количество маркеров, которые находятся 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 рёбра графа (каждое ребро может быть входным или выходным и иметь вес, который пишется над ребром, см. рис. 1).

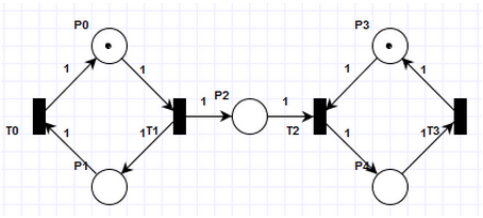

Рис. 1: Визуализация сети Петри.

Переход $t \in T$ называется активным при маркировке μ, если $\forall p \in P\ \phi(p,t) \le \mu(p)$. Любой активный переход сети может сработать, в этом случае из каждой входной позиции p перехода удаляются $\phi(p,t)$ маркеров и для каждой выходной позиции перехода p' в неё добавляются $\phi(t,p')$ маркеров. Можно записать срабатывание перехода как $\mu \xrightarrow{t} \mu'$, где $\mu'(p) = \mu(p) - \phi(p,t) + \phi(t,p)\ \forall p \in P$.

Практический смысл приведённых 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 определённый 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 — неопределённость, 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 наперёд заданное число k такое, что в любой достижимой маркировке сети в каждой позиции находится не более k маркеров. Практический смысл — в системе никогда не произойдёт 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 данной маркировки, в которой заданный переход может сработать. Живость вводится как для конкретного перехода, так и для сети в целом, при этом определяется 6 уровней живости, от 0-го (переход не может сработать никогда) до 5-го (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 мёртвого кода.

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

- **Анализ уравнения состояния** — сеть Петри представляется в виде матрицы инциденции, 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 ещё 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, рёбрами — 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 трудоёмкость.
- **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 трудоёмкость, 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 быть использованы для динамической верификации путём симуляции сети. На этом теоретическое введение заканчивается, подробнее про сети Петри можно прочитать в статье [23], оценки трудоёмкости анализа сетей приводятся в статье [22], хорошее введение в сети Петри с точки зрения их приложений в промышленности приводится в [24], пример работы о применении в технологическом процессе — [25].

В робототехнике сети Петри используются разными способами. В работах [26] и [27] 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 [28] описывает симулятор для разных видов сетей Петри (временных, ингибиторных, с приоритетами), который используется 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 [29] и [30], причём в последнем случае предлагается новое расширение сетей Петри, и для их формального анализа используются техники model checking.

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 чёткой семантики и наглядность упрощают разработку программ и написание для них интерпретаторов. Примеры таких работ — [31] (визуальный язык со вставками на языке Python для программирования мультиагентных систем), [32] (интерпретация ингибиторных сетей Петри для управления группой роботов, с визуальным редактором сетей и посылкой команд роботам по радиоканалу), [33] (программирование 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 — [34], здесь действия каждого робота в группе моделируются ингибиторной сетью, определяется, может ли один робот в группе заменить другой для решения той или иной подзадачи, предлагается алгоритм построения оптимального с точки зрения времени выполнения плана работы роботов в группе.

Несколько менее типичный пример использования сети Петри — работа [35], где сеть используется для представления знаний робота при обучении по демонстрации. Человек-учитель перемещает некоторые 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.

IV. Communicating Sequential Processes

Речь в данном разделе пойдет о формальном языке описания параллельных систем CSP (Communicating Sequential Processes) и языке occam[6], построенном на формализме CSP и о некоторых работах, посвященных применению их в робототехнике. CSP принадлежит к классу математических теорий под общим названием *алгебр процессов* (*process algebras*) [36], [37]. В простейшем виде алгебра процессов для множества атомарных действий A определяется как замыкание множества 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, *deadlock*). 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 [38], [39], [40]. Здесь будут неформально определены основные операции в алгебре CSP.

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

- **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 $tick \to tick \to tick \to \ldots$. 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** («□»). $P = (a \to Q) \Box (b \to R)$ 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 произведен недетерминировано.
- **Параллельная композиция по пересечению** («||»). Данный оператор позволяет процессам исполняться параллельно, но только в случае, когда оба процесса к этому готовы. Более формально,

$$(x : B \to P(x)) \,||\, (y : C \to Q(y)) = (z : (B \cap C) \to (P(z) \,||\, Q(z))). \quad (1)$$

  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

$$P \,|||\, Q = (x : B \to (P(x) \,|||\, Q)) \Box (y : C \to (P \,|||\, Q(y))), \quad (2)$$

  т.е. 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 в 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 а тогда и только тогда, когда P выполняет $f(a)$.

Существует множество инструментов для анализа CSP, например, FDR2[7] и PAT[8]. FDR2, применения которого мы рассмотрим ниже, является коммерческим инструментом проверки моделей для CSP (*refinement checker*), позволяющим формально доказать выполнимость свойств процесса, описанного на входном языке FDR2, который практически полностью совпадает с версией языка CSP самого Ч.Э.Хоара (автора CSP) из [40]. Также CSP повлиял на такие языки программирования, как *occam*-π [41] и Go[9].

Перейдем к рассмотрению работ применения описанного формализма к задачам робототехники. В 1998 году была опубликована статья [42], в которой обсуждается применение формальных методов в цеях 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 [43], [44] 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). Некоторые узлы могут представлять внешние опасности, в таком случае они не раскрываются и не рассматриваются в процессе верификации и нужны лишь для полноты спецификации опасностей. 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. Утверждается, что такой подход более выразителен, чем просто постановка требований в терминах LTL (см. раздел II), 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 обсуждаются другие возможные применения для подхода с деревом опасностей.

Работа [45] интересна в контексте образовательной робототехники, а также в контексте визуального моделирования. В ней обсуждается разработка визуального языка для программирования роботов Surveyor SRV-1 в целях обучения студентов Университета Кента параллельному программированию, для чего раннее использовался язык *occam*-π [41], основанный на формализмах π-исчисления [41] и CSP.

В работе [46] обсуждается применение языка *occam*-π для построения роевой [47] системы роботов уборки сборки отходов. Для реализации системы выбрана архитектура Брукса [48], 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 довольно удобным при программировании системы сбора мусора группой роботов.

Работа [49] изучает подход к планированию передвижения роботов с использованием алгебр процессов. Общая схема подхода примерно такая же, как у описанного в разделе II данной работы, однако вместо спецификаций LTL используются алгебры процессов.

V. 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 различных отраслях науки (включая такие, как экология хищников-жертв).

[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)

На тему теории гибридных систем выходят целые сборники (например, [50], [51]), создано множество инструментов анализа, включая HyTech [52], PHAVer [53], KeYmaera [54]. Мы рекомендуем работу [55] для ознакомления с общими принципами работы верификаторов гибридных систем. Среди приложений теории гибридных систем к робототехнике авторам не известны какие-либо «обособленные» работы, которые не могли бы быть применены для других классов инженерных систем, поэтому ограничимся здесь лишь приведенным общим описанием области.

VI. Марковские модели

Еще одной интересной и активно развивающейся областью применения математического аппарата к задачам робототехники является теория марковских процессов принятия решений. Для подробного ознакомления с теорией марковских процессов мы отсылаем читателя к [56].

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;
- $D_0 : S \to R$ — начальное распределение вероятности по состояниям;
- $P : S \times A \times S \to [0, 1]$ — функция вероятности перехода. По данным $s, s' \in S$ и $a \in A$ $P(s, a, s')$ означает вероятность перехода из состояния s в состояние s' при совершении системой действии a;
- $R : S \times A \times S \to R$ — функция награды. Значение $R(s, a, s')$ 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 наград в будущем.

Политикой процесса M 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 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. Очевидно, что при фиксированном марковском процессе M и политике f, комбинация (M, f) ведет себя в точности как марковская цепь.

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 0, заставляют систему пренебрегать более удаленными во времени наградами, близкие к 1 — учитывать их.

MDP применяются для моделирования поведения дискретных систем, поведения которых носят стохастический характер, поэтому часто применяются в робототехнике. С их помощью моделируется поведение самих роботов и их частей (вероятности переходов в таком случае означают, вероятности успешного выполнения задачи, например, захвата предмета клешней по видеокамере), элементов окружения и даже поведения человека [57], [58], [59] (к примеру, 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 Drocess, LMDP)*. Можно проследить аналогию между LMDP с метками и структурами Крипке: первые являются, по сути, вторыми, но переходы между состояниями носят вероятностный характер и могут не произойти. Детерминированная политика LMDP будет в таком случае соответствовать траектории на структуре Крипке. Мы заимствуем терминологию и определения из раздела II и отсылаем читателя к тем же работам для лучшего понимания.

Как известно, по заданной формуле LTL можно построить соответствующий автомат Бюхи. Напоминаем, что в общей схеме алгоритма model checking для LTL по структуре Крипке K (точнее, по соответствующему ей автомату Бюхи B_K) и автомату Бюхи, распознающего неправильное поведение $B_{\neg\phi}$, строится синхронная композиция автоматов $B = B_M \otimes B\neg\phi$, в которой затем ищется допускающий проход, который, если найден, является контрпримером к требованию корректности ϕ. Аналогично для марковского процесса принятия решений M и формуле LTL ϕ может быть построена их композиция $M' = M \boxtimes B_\phi$. Формальное определение этой композиции лишь усложнит содержание данной статьи и может быть найдено, к примеру, в [60]. Интуитивно, такая композиция аналогична синхронному произведению автоматов, оставляя вероятности и награды на переходах соответствующими LMDP M, но само множество переходов между состояниями редуцируется до присутствующих в автомате B_ϕ. Полученное произведение позволяет рассуждать о синтезе политик, максимизирующих награду над бесконечным (или конечным) горизонтом и удовлетворяющих свойству

корректности ϕ.

В [60] приводится интересное приложение описанной теории 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 принятия решений.

Другая часто применяющаяся в контексте робототехники математическая структура — частично наблюдаемые марковские процессы принятия решений (*Partially Observable Markov Decision Process, POMDP*). От обычных MDP POMDP отличается 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 $(S, A, D_0, P, R, Z, O, \gamma)$, 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*. Задачей качественного анализа POMDP является нахождение политики, удовлетворяющей цели с вероятностью 1. Количественный анализ задается вопросом об удовлетворении цели с вероятностью $\lambda \in (0, 1)$. В [61] доказывается неразрешимость задачи количественного анализа. В [62] приводится конструктивное доказательство того, что задача качественного анализа POMDP разрешима и EXPTIME-полна.

Отметим, что область анализа POMDP является на данный момент мало исследованной. Самой значимой и интересной на данный момент работой на тему применения качественного анализа POMDP в контексте робототехники, на взгляд авторов, является статья [63] 2015 года, где исследовано применение качественного анализа POMDP совместно с LTL-требованиями для классических задач планирования поведения робота.

VII. Другие используемые формализмы

A. Process Algebra for Robot Schemas

Существуют формализмы, используемые для верификации и разработки робототехнических систем, которые не попали ни в одну из категорий выше. Один из таких формализмов — ещё один вариант алгебры процессов, PARS (Process Algebra for Robot Schemas), описанный в работах [64] и [65]. 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 передаёт 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 неопределённость, имеющуюся в реальном мире, переменные рассматриваются как случайные величины с некоторой функцией распределения. Вводится процесс $Ran\langle\Phi\rangle\langle v\rangle$, возвращающий случайную величину v с распределением Φ.

Целью введения такого формализма было подавление комбинаторного взрыва, которому подвержены многие другие методы анализа. Достигается это так: авторы рассматривают 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 серьёзных ограничений на моделируемые системы, поскольку представление в требуемом виде во многих случаях

может быть построено автоматически (причём, полиномиальным относительно количества процессов алгоритмом). Дальше 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.

B. Integrated Behavior-Based Control

Ещё один интересный формализм, используемый для программирования роботов, — Integrated Behavior-Based Control (i2BC), формализм и архитектура, описанные в статье [66]. Архитектура базируется на известной в среде робототехников работе [48] 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 0 до 1, при этом смысл их таков: если ингибиция равна 1, то поведение не должно работать, если 0, то должна работать полностью. Соответственно, если активация равна 1, то наоборот, поведение должно работать, если 0, то не должно (обычно их комбинируют как $a*(1-i)$, где a — это активация, i — ингибиция). Помимо выхода для данных есть ещё выходы активности и целевого рейтинга. Это тоже числа от 0 до 1, активность показывает, насколько поведение активно и готово влиять на состояние системы (с учётом 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 формализм i2BC интересна тем, что сводима к набору конечных автоматов, для которых оказываются применимы методы model checking, как описано в работе [67]. Каждое поведение представляется в виде набора автоматов, связанных друг с другом каналами для сообщений. Автоматное представление системы может быть получено автоматически по модели. Авторы [67] проводили эксперименты по оценке влияния отказов датчиков (тоже моделируемых с помощью поведений) 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).

C. Формальная логика

Помимо верификации уже существующих систем, формальные методы применяются при нахождении корректных «по построению» и оптимальных решений. Ещё один пример такого применения (помимо описанных в разделе II) — решение задачи о нахождении оптимального плана реконфигурации модульных роботов путём сведения её к задаче SAT методами матлогики [68]. Оптимальный (в смысле количества шагов соединения-рассоединения) план для групп размером примерно в 400 роботов находился существующими решателями SAT за время порядка десятков минут.

VIII. 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 ещё убедиться, что модель соответствует реальному миру, в котором будет функционировать робот. Формально это доказать в общем случае невозможно в силу непредсказуемости реального мира, но можно проверить во время выполнения и активировать безопасный (или аварийный) механизм поведения, если данные о реальном мире перестали соответствовать предсказанным моделью.

Пример такой работы — средство ModelPlex, описанное в работе [69]. Верифицированная модель робота, программы управления и окружающей среды здесь уже дана, 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 нет. Авторы формально доказывают корректность и своевременность перехода на аварийный режим работы при условии ограниченности возможных отклонений реальной среды от модели.

Менее формальный метод обеспечения корректности предложен для системы ROS[10] в работе [70]. Система ROS представляет программу в виде модулей (узлов), связанных каналами, по которым узлы обмениваются сообщениями. В работе предлагается вставлять между

[10]Robot Operating System, URL: http://www.ros.org/ (дата обращения: 30.09.2015)

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 статье приводится пример с контролем угла наклона пейнтбольной пушки на роботе, чтобы подавить сигнал о выстреле, если робот может попасть в себя.

IX. Заключение

В статье приведён обзор использования формальных методов в робототехнике. Обзор проводился по материалам конференций ICRA и IROS последних двух лет, а также данным Scopus и Google Scholar. Наиболее популярный и исследуемый подход на данный момент — постановка задач посредством LTL-спецификаций, также активно ведутся исследования применений 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 решения в некотором смысле обратной задачи — построения программы поведения робота по заданной модели среды и желаемым результатам. Это позволяет задавать поведение робота очень декларативно, и результирующая программа будет корректна по построению, однако такое построение часто требует довольно больших вычислительных ресурсов.

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

Список литературы

[1] Antoniotti Marco, Mishra Bud. Discrete event models+ temporal logic= supervisory controller: Automatic synthesis of locomotion controllers // Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on / IEEE. Т. 2. 1995. С. 1441–1446.

[2] Lamport Leslie. The temporal logic of actions // ACM Transactions on Programming Languages and Systems (TOPLAS). 1994. Т. 16, № 3. С. 872–923.

[3] Clarke Edmund M., Emerson E. Allen, Sistla A. Prasad. Automatic verification of finite-state concurrent systems using temporal logic specifications // ACM Transactions on Programming Languages and Systems (TOPLAS). 1986. Т. 8, № 2. С. 244–263.

[4] Карпов Юрий Глебович. MODEL CHECKING. Верификация параллельных и распределенных программных систем. БХВ-Петербург, 2010.

[5] Emerson E Allen. Temporal and modal logic. // Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B). 1990. Т. 995, № 1072. С. 5.

[6] Piterman Nir, Pnueli Amir, Sa'ar Yaniv. Synthesis of reactive (1) designs // Verification, Model Checking, and Abstract Interpretation / Springer. 2006. С. 364–380.

[7] Pnueli Amir, Rosner Roni. On the synthesis of a reactive module // Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages / ACM. 1989. С. 179–190.

[8] Kripke Saul A. Semantical analysis of modal logic i normal modal propositional calculi // Mathematical Logic Quarterly. 1963. Т. 9, № 5-6. С. 67–96.

[9] Fainekos Georgios E, Kress-Gazit Hadas, Pappas George J. Temporal logic motion planning for mobile robots // Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on / IEEE. 2005. С. 2020–2025.

[10] Nusmv 2: An opensource tool for symbolic model checking / Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia [и др.] // Computer Aided Verification / Springer. 2002. С. 359–364.

[11] Holzmann Gerard J. The SPIN model checker: Primer and reference manual. Addison-Wesley Reading, 2003.

[12] Belta Calin, Habets Luc CGJM. Constructing decidable hybrid systems with velocity bounds // Decision and Control, 2004. CDC. 43rd IEEE Conference on / IEEE. Т. 1. 2004. С. 467–472.

[13] Kress-Gazit Hadas, Fainekos Georgios E, Pappas George J. Where's Waldo? Sensor-based temporal logic motion planning // Robotics and Automation, 2007 IEEE International Conference on / IEEE. 2007. С. 3116–3121.

[14] Kress-Gazit Hadas, Fainekos Georgios E, Pappas George J. Temporal-logic-based reactive mission and motion planning // Robotics, IEEE Transactions on. 2009. Т. 25, № 6. С. 1370–1381.

[15] Composition of local potential functions for global robot control and navigation / David C Conner, Alfred Rizzi, Howie Choset [и др.] // Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on / IEEE. Т. 4. 2003. С. 3546–3551.

[16] Livingston Scott C, Murray Richard M. Just-in-time synthesis for reactive motion planning with temporal logic // Robotics and Automation (ICRA), 2013 IEEE International Conference on / IEEE. 2013. С. 5048–5053.

[17] Towards manipulation planning with temporal logic specifications / Keliang He, Morteza Lahijanian, Lydia E Kavraki [и др.] // Robotics and Automation (ICRA), 2015 IEEE International Conference on / IEEE. 2015. С. 346–352.

[18] Combined task and motion planning through an extensible planner-independent interface layer / Sanjeev Srivastava, Eugene Fang, Lorenzo Riano [и др.] // Robotics and Automation (ICRA), 2014 IEEE International Conference on / IEEE. 2014. С. 639–646.

[19] Raman Vasumathi, Kress-Gazit Hadas. Synthesis for multi-robot controllers with interleaved motion // Robotics and Automation (ICRA), 2014 IEEE International Conference on / IEEE. 2014. С. 4316–4321.

[20] Vasile Cristian Ioan, Belta Calin. Reactive sampling-based temporal logic path planning // Robotics and Automation (ICRA), 2014 IEEE International Conference on / IEEE. 2014. С. 4310–4315.

[21] Plünnecke Helmut, Reisig Wolfgang. Bibliography of Petri nets 1990 // Advances on I'rtri Net.s 1991, Lecture Notes in Cornputer Science. 1991. Т. 524. С. 312–572.

[22] Yen Hsu Chun. Introduction to Petri net theory // Studies in Computational Intelligence. 2006. Т. 25. С. 343–373.

[23] Murata Tadao. Petri nets: Properties, analysis and applications // Proceedings of the IEEE. 1989. Т. 77, № 4. С. 541–580.

[24] Zurawski Richard, Zhou MengChu. Petri nets and industrial applications: A tutorial // IEEE Transactions on Industrial Electronics. 1994. Т. 41, № 6. С. 567–583.

[25] Мартынов Виктор Геннадьевич, Масягин Василий Борисович. Применение сетей Петри при моделировании управления технологическими процессами сборочного производства // Омский научный вестник. 2014. № 1 (127). С. 134–137.

[26] Costelha Hugo, Lima Pedro. Robotic Tasks Modeling and Analysis Based on Petri Nets. 2003.

[27] Costelha Hugo, Lima Pedro. Petri net robotic task plan representation: Modelling, analysis and execution // Autonomous Agents. 2010. С. 65–91.

[28] Конюх В.Л. Опыт применения сетей Петри для имитации поведения систем // «Имитационное моделирование. Теория и практика» ИММОД-2009, пленарный доклад. 2009. С. 27–37.

[29] Aguiar Adriano Jose Cunha, Villani Emilia. Petri Nets and Graphic Simulation for the Validation of Collaborative Robotic Cells in Aircraft Industry // ABCM Symposium Series in Mechatronics. 2010. Т. 4. С. 364–373.

[30] 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.

[31] Simon Jochen, Moldt Daniel. PyTri, a Visual Agent Programming Language // Algorithmen und Werkzeuge f{\"u}r Petrinetze. 2010. Т. 643. С. 106—-111.

[32] Kashima Hideharu, Masuda Ryosuke. Cooperative Control of Mobile Roots Based on Petri Net // The 10th international conference on advanced robotics (ICAR'2001). 2001. С. 399–404.

[33] A Robotic Soccer Passing Task Using Petri Net Plans (Demo Paper) / F Palamara, V a Ziparo, L Iocchi [и др.] // Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers. 2008. С. 1711–1712.

[34] Kotb Y T, Beauchemin S S, Barron J L. Petri Net-Based Cooperation In Multi-Agent Systems // Computer and Robot Vision, 2007. CRV'07. Fourth Canadian Conference on. 2007. С. 123—-130.

[35] Chang Guoting Jane, Kulić Dana. Robot Task Learning from Demonstration Using Petri Nets // 2013 IEEE RO-MAN: The 22nd IEEE International Symposium on Robot and Human Interactive Communication Gyeongju, Korea. 2013. С. 31–36.

[36] Baeten Jos CM. A brief history of process algebra // Theoretical Computer Science. 2005. Т. 335, № 2. С. 131–146.

[37] Fokkink Wan. Introduction to process algebra. Springer Science & Business Media, 2013.

[38] Hoare Charles Antony Richard. Communicating sequential processes // Communications of the ACM. 1978. Т. 21, № 8. С. 666–677.

[39] Brookes Stephen D, Hoare Charles AR, Roscoe Andrew W. A theory of communicating sequential processes // Journal of the ACM (JACM). 1984. Т. 31, № 3. С. 560–599.

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

[41] Welch Peter H, Barnes Frederick RM. Communicating mobile processes // Communicating Sequential Processes. The First 25 Years. Springer, 2005. С. 175–210.

[42] Lankenau Axel, Meyer Oliver. Formal methods in robotics: Fault tree based verification // In: Proc. of Quality Week / Citeseer. 1999.

[43] Fault tree handbook: Tech. Rep.: / William E Vesely, Francine F Goldberg, Norman H Roberts [и др.]: DTIC Document, 1981.

[44] Hansen Kirsten Mark. Linking safety analysis to safety requirements: exemplified by railway interlocking systems. Institut for Informationsteknologi, DTU, 1996.

[45] Simpson Jonathan, Jacobsen Christian L. Visual Process-Oriented Programming for Robotics. // CPA. 2008. С. 365–380.

[46] Process-Oriented Subsumption Architectures in Swarm Robotic Systems. / Jeremy C Posso, Adam T Sampson, Jonathan Simpson [и др.] // CPA. 2011. С. 303–316.

[47] Şahin Erol. Swarm robotics: From sources of inspiration to domains of application // Swarm robotics. Springer, 2005. С. 10–20.

[48] Brooks R. A robust layered control system for a mobile robot // IEEE Journal on Robotics and Automation. 1986. Т. 2, № 1. С. 14–23.

[49] Varricchio Valerio, Chaudhari Pratik, Frazzoli Emilio. Sampling-based algorithms for optimal motion planning using process algebra specifications // Robotics and Automation (ICRA), 2014 IEEE International Conference on / IEEE. 2014. С. 5326–5332.

[50] Vaandrager Frits W, van Schuppen Jan H. Hybrid Systems: Computation and Control: Second International Workshop, HSCC'99, Berg en Dal, The Netherlands, March 29-31, 1999 Proceedings. № 1569. Springer Science & Business Media, 1999.

[51] Lynch Nancy, Krogh Bruce. Hybrid Systems: Computation and Control: Third International Workshop, HSCC 2000 Pittsburgh, PA, USA, March 23-25, 2000 Proceedings. Springer Science & Business Media, 2007.

[52] Henzinger Thomas A, Ho Pei-Hsin, Wong-Toi Howard. HyTech: A model checker for hybrid systems // Computer aided verification / Springer. 1997. С. 460–463.

[53] Frehse Goran. PHAVer: Algorithmic verification of hybrid systems past HyTech // Hybrid Systems: Computation and Control. Springer, 2005. С. 258–273.

[54] Platzer André, Quesel Jan-David. KeYmaera: A hybrid theorem prover for hybrid systems (system description) // Automated Reasoning. Springer, 2008. С. 171–178.

[55] The algorithmic analysis of hybrid systems / Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs [и др.] // Theoretical computer science. 1995. Т. 138, № 1. С. 3–34.

[56] Beranek William. RONALD A. HOWARD "Dynamic Programming and Markov Processes," // Technometrics. 1961. Т. 3, № 1. С. 120–121.

[57] Pentland Alex, Liu Andrew. Modeling and prediction of human behavior // Neural computation. 1999. Т. 11, № 1. С. 229–242.

[58] Rothkopf Constantin A, Ballard Dana H. Modular inverse reinforcement learning for visuomotor behavior // Biological cybernetics. 2013. Т. 107, № 4. С. 477–490.

[59] McGhan Catharine LR, Nasir Ali, Atkins Ella M. Human intent prediction using markov decision processes // Journal of Aerospace Information Systems. 2015. С. 1–5.

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

[61] Baier Christel, Größer Marcus, Bertrand Nathalie. Probabilistic ω-automata // Journal of the ACM (JACM). 2012. Т. 59, № 1. С. 1.

[62] Chatterjee Krishnendu, Chmelik Martin, Tracol Mathieu. What is Decidable about Partially Observable Markov Decision Processes with {\ omega}-Regular Objectives // arXiv preprint arXiv:1309.2802. 2013.

[63] Qualitative analysis of pomdps with temporal logic specifications for robotics applications / Krishnendu Chatterjee, Martin Chmelík, Raghav Gupta [и др.] // arXiv preprint arXiv:1409.3360. 2014.

[64] Performance Verification for Behavior-based Robot Missions / D Lyons, R Arkin, S Jiang [и др.] // IEEE Transactions on Robotics. 2015. Т. 31, № 3. С. 619 – 636.

[65] A Software Tool for the Design of Critical Robot Missions with Performance Guarantees / Damian M Lyons, Ronald C Arkin, Paramesh Nirmal [и др.] // Procedia Computer Science. 2013. Т. 16. С. 888–897.

[66] Proetzsch Martin, Luksch Tobias, Berns Karsten. The Behaviour-Based Control Architecture iB2C for Complex Robotic Systems. 2007.

[67] Kiekbusch Lisa, Armbrust Christopher, Berns Karsten. Formal Verification of Behaviour Networks Including Hardware Failures // Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13). 2014.

[68] Gorbenko A. A., Popov V. Yu. Programming for modular reconfigurable robots // Programming and Computer Software. 2012. Т. 38, № 1. С. 13–23.

[69] Mitsch Stefan, Platzer Andre. ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models // Runtime Verification. 2014. С. 199–214.

[70] ROSRV: Runtime Verification for Robots / Jeff Huang, Cansu Erdogan, Yi Zhang [и др.] // Runtime Verification. 2014. С. 247–254.