Бугайченко Дмитрий Юрьевич

Разработка и реализация методов формально-логической спецификации самонастраивающихся мультиагентных систем с временными ограничениями

Санкт-Петербург

2007

УДК 510.64; 519.681.

Материалы диссертации на соискание ученой степени кандидата физико-математических наук по специальности “Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей”. Работа выполнена на кафедре информатики математико-механического факультета Санкт-Петербургского государственного университета под руководством доцента Соловьева И.П.

Оглавление

Введение

Одной из основных особенностей многих современных информационных систем является то, что они не предназначены для самостоятельного принятия решений. Предположительно, все возможные варианты поведения таких систем должны быть спроектированы человеком и заложены в них на этапе разработки. Попадание подобной системы в условия, не учтенные ее разработчиками, может приводить к аварийному завершению или более тяжелым последствиям. Одним из подходов, направленных на решение этой проблемы, является применение агентно-ориентированного метода проектирования и разработки. Отличительным свойством концепции *агента* (программного агента) является наличие внешней среды, с которой агент способен взаимодействовать, но не обладает возможностью ее контролировать и поэтому всегда должен быть готов к тому, что предпринятые им действия не приведут к желаемым результатам. Это свойство делает концепцию агента привлекательным инструментом для решения многих задач, в том числе, для создания систем управления сложными устройствами и комплексами.

Для современных информационных систем все чаще встает требование способности к оптимизации поведения в условиях изменяющейся внешней среды, а также способности к накоплению и анализу опыта, что, в определенной степени, входит в конфликт с жесткими ограничениями на время реакции и вычислительную мощность оборудования. Кроме того, для подобных систем особенно остро стоит вопрос корректности их поведения.

Частично проблему корректности помогает решить тщательное и обширное тестирование, но оно обладает тем же недостатком — поведение системы обычно проверяется только в тех условиях, которые запланированы человеком. В случае действительно сложных систем полное тестирование, проверяющее все возможные комбинации условий, как правило, невозможно за разумное время.

Представляется, что решение данной проблемы следует искать на пути применения методов *формальной* спецификации проектируемой системы с последующим *автоматическим* тестированием или *верификацией* (обычно это проверка модели или model checking).

Основу большинства методов формальной спецификации составляет некоторый вариант *темпоральной логики*. Наиболее широкое распространение получила логика ветвящегося времени CTL [56], основным достоинством которой является билинейная относительно размера спецификации и размера модели системы сложность задачи верификации. CTL расширяет классическую логику высказываний темпоральными операторами "в следующий момент ϕ" $\bigcirc\phi$ и "когда-нибудь ϕ, а до тех пор ψ" $\psi\,\mathcal{U}\,\phi$ и кванторами "на любом пути ..." $\mathbf{A}$ и "существует путь на котором ..." $\mathbf{E}$.

Для спецификации систем с временными ограничениями предложены различные расширения логики CTL. Например, логика $RTCTL$ [150] расширяет CTL ограниченным оператором $\psi\,\mathcal{U}_{\leq t}\phi$, интерпретируемым как "не более чем через t шагов ϕ, а до тех пор ψ". Основным достоинством $RTCTL$ является сохранение билинейной сложности задачи верификации. Большей выразительной мощности, за счет увеличения вычислительной сложности, можно достичь введя переменные часов $(x, y, \ldots)$ и квантор фиксации $(\cdot)$ [24]: $x{\cdot}\mathbf{A}\,\psi\,\mathcal{U}\,((x \leq t) \wedge \phi)$.

Основным ограничением CTL является отсутствие явных средств спецификации систем, состоящих из нескольких взаимодействующих сущностей. Это ограничение преодолевается в логике альтернированного времени ATL [25], позволяющей учитывать многокомпонентность системы в явном виде. ATL включает конечное множество игроков-агентов P и заменяет кванторы CTL на квантор "для коалиция игроков $C \subseteq P$ существует такая стратегия, что на любом пути ..." $\langle\langle C\rangle\rangle$. Для ATL также были предложены методы спецификации ограничений на время реакции, например, с помощью подстрочного индекса [108] или с помощью переменных часов [73].

Характерной чертой многих методов формальной верификации является комбинаторный взрыв пространства состояний верифицируемых систем. Частично эту проблему позволяют решить так называемые *символические* (symbolic) алгоритмы, которые оперирует с множествами состояний, а не с отдельными состояниями. Серьезный толчок к развитию символических алгоритмов дало введение ориентированных булевых разрешающих диаграмм ($OBDD$) [41], позволяющих относительно компактно представлять большие множества и эффективно выполнять над ними операции. Было предложено множество различных модификаций концепции $OBDD$, среди которых можно выделить алгебраические разрешающие диаграммы (ADD) [18].

Для уменьшения времени реакции системы широко используется подход *исполняемых планов* [66]. В этом случае до начала непосредственного взаимодействия с внешней средой система составляет план действий для достижения своих целей. Существует множество различных алгоритмов планирования, используемых для мультиагентных систем, часть из которых базируется на классических алгоритмах поиска пути в графе A^* [141]. Однако подобные алгоритмы также подвержены проблеме комбинаторного взрыва пространства состояний, частично решить которую позволяют алгоритмы *символического планирования* [47].

Описанные выше алгоритмы позволяют строить планы для достижения относительно простых целей, например, достижение системой определенного состояния при посещении только допустимых состояний. В работе [145] были предложены алгоритмы для решения более сложной задачи — построения плана, удовлетворяющего спецификации, описанной с помощью логики линейного времени[1]. Подобный подход получил название “планирование с темпорально

[1]Синтаксис логики линейного времени LTL включает те же темпоральные операторы, что и логика ветвящегося времени CTL, но не включает кванторов путей, так как семантика LTL предполагает только один возможный вариант будущего.

расширенными целями".

Другой задачей, возникающей при разработке самонастраивающихся мультиагентных систем, является реализация методов накопления и анализа опыта. Набор методов, предложенных для решения этой задачи очень широк, активно развивается по сей день и включает самые разнообразные методы — от нейронных сетей до сложных баз знаний.

Одной из значимых тенденций в современных информационных системах является параллелизация вычислений и интеграция отдельных информационных систем в более мощные вычислительные комплексы. Влияние этих тенденций на развитие самонастраивающихся систем привело к формированию концепции социальных агентов и мультиагентных систем. Первые работы, предлагающие декомпозицию системы в виде набора автономных взаимодействующих сущностей, появились в 70-х годах 20-го века [69, 75, 111], однако термин *агент* впервые появился лишь спустя десять лет [61].

Исторически первым методом реализации распределенных интеллектуальных систем является метод классной доски, впервые примененный в системе HEARSAY [69], предназначенной для распознавания речи. Метод включает следующие основные архитектурные элементы:

- Глобальную структуру данных, называемую *классной доской (blackboard)*;
- несколько параллельно работающих носителей знаний, *экспертов (knowledge source)*, которые постоянно видят содержимое классной доски;
- центральный *контролирующий механизм (control algorithm)* или протокол, определяющий порядок доступа экспертов к классной доске.

В данном случае эксперты выступают как основные участники процесса решения задачи. Они не общаются между собой напрямую, а взаимодействуют

с контролирующим механизмом и помещают информацию на доску. Эксперты постоянно сканируют содержимое доски или получают уведомления о его изменении. Изначально на доске формулируется задача, которую необходимо решить. Если один из экспертов знает способ решения задачи или хотя бы информацию, которая, по его мнению, может помочь решению задачи, он запрашивает контролирующий механизм получения доступа на запись к классной доске. Когда доступ будет получен, эксперт записывает информацию на доску, и она становится видима всем остальным экспертам.

Существует много вариаций метода классной доски. В частности, можно завести классную доску для записей целей и/или планов агентов, и в этом случае она послужит удобным инструментом для решения проблемы согласованности. Однако наличие централизованного контролирующего механизма и единой централизованной структуры данных является одним из основных ограничений данного подхода и не позволяет эффективно использовать его в полноценно распределенных интеллектуальных системах, таких как мультиагентные системы.

Другой метод был разработан Дугласом Ленатом в его концепции *существ*. Эта концепция была направлена на разработку метода, позволяющего моделировать решение проблемы сообществом экспертов, каждый из которых является специалистом в определенной предметной области. В отличие от метода классной доски, Ленат предложил метод, основанный на взаимодействии экспертов посредством вопросов и ответов. При этом и вопросы, и ответы Ленат предлагал рассылать широковещательно. Существа, в концепции Лената, моделировали таких экспертов.

Для проверки концепции существ Ленат разработал приложение PUP6 [111], где каждое существо моделировалось на LISP в виде фрейма с двадцатью семью слотами (в терминологии Лената *частями (parts)*), представлявшими вопросы, на которое существо могло ответить. При приходе сообщения, существо сравни-

вало его со своими слотами, что могло привести к широковещательной посылке нового сообщения. Опыт, полученный при разработке PUP6, Ленат использовал в своем известном проекте Artificial Mathematician (AM) [112].

Концепция существ лучше подходит для моделирования агентов и мультиагентных систем, чем классная доска. Большим плюсом концепции является отсутствие централизованного механизма управления и полноценная распределенность. Однако существа в концепции Лената не являются самообучающимися и не способны к накоплению и анализу опыта. Набор их знаний фиксирован, и они не обладают возможностью их расширять.

Первые наработки концепции *акторов* [14, 75] появились практически одновременно с работами Лената по существам. Концепция была основана на объектно-ориентированном подходе к моделированию и могла рассматриваться как метод проектирования параллельных объектно-ориентированных систем. В отличие от существ, концепция акторов по сей день активно развивается как в теории, так и на практике в различных распределенных системах.

Система состоит из набора акторов, которые остаются пассивными до тех пор, пока не получат сообщение. Когда сообщение получено, актор пытается выбрать соответствующее сообщению *поведение (behavior)* (в ранних работах поведение называлось *сценарием (script)*). Реакция актора на сообщения могла быть трех видов:

- посылка нового сообщения себе или другим агентам;
- создание дополнительных акторов;
- определение *замещения (replacement)* поведения.

Сообщения в системе посылаются асинхронно, а каждый актор имеет уникальный адрес, таким образом, сообщения рассылаются не широковещательно,

а конкретным получателям. Каждый актор имеет очередь сообщений, в которой хранятся пришедшие сообщения, пока они не буду обработаны. Определение *замещения* поведения подразумевает создание нового актора, который унаследует адрес и очередь сообщений своего создателя. Т.е.В этом случае следующие сообщения в очереди, а так же те сообщения, что отправятся позднее, будут обработаны уже новым актором.

Большим плюсом концепции акторов является её высокий параллелизм, ограниченный исключительно возможностями ЭВМ. Однако актора еще нельзя считать полноценным агентом, так как он представляет достаточно простую вычислительную единицу, все еще не обладающую возможностями к самообучению.

Multi-Agent Computing Environment или MACE исторически является первой полноценно распределенной интеллектуальной системой, описание которой было впервые опубликовано в 1987 году Гессером [61]. Система состоит из следующих пяти компонент:

- набор *прикладных агентов (application agents)*, которые являлись основными вычислительными единицами системы;
- набор предопределенных *системных агентов (system agents)*, предоставлявших сервис пользователю;
- набор средств общего назначения, доступных все агентам (facilities);
- *база данных с описаниями (description databases)*, которая содержала описания агентов и могла произвести исполняемый код на основе описания;
- набор *ядер (kernels)*, по одному на каждую физическую машину, которые обеспечивали коммуникацию и маршрутизацию сообщений.

Гессер выделил три аспекта деятельности агентов: они содержат знания, они ощущают окружающую среду, они выполняют действия. Агенты $MACE$ имели

два типа знания: *специализированные (domain knowledge)*, ориентированные на предметную область, и *ознакомительные знания (acquaintance knowledge)* — знания о других агентах системы. Агент хранил следующую информацию о других агентах.

- *Класс (class)* – агенты организовывались в группы, называемые классами и идентифицируемые по имени класса.
- *Имя (name)* – каждому агенту присваивалось уникальное в классе имя, таким образом, в рамках системы агента можно было идентифицировать парой строк (класс, имя).
- *Роль (roles)* – описание роли, выполняемой агентом в своем классе.
- *Навыки (skills)* – множество возможностей описываемого агента по сведениям описывающего агента.
- *Цели (goals)* – множество целей, которых описываемый агент хочет достичь по сведениям описывающего агента.
- *Планы (plans)* – представление описывающего агента о том, как описываемый агент будет достигать свои целей.

Агенты ощущали внешнюю среду посредством получения сообщений, а единственный видимый результат деятельности агентов заключался в рассылке сообщений, адресованных конкретному агенту, группе агентов или всем агентам.

Универсальность и гибкость системы MACE позволяла моделировать многие другие интеллектуальные системы. В частности, метод классной доски моделировался с помощью MACE достаточно легко.

Система MACE явилась важной вехой в развитии распределенных интеллектуальных систем, и многие новшества, в ней введенные, активно применялись

в последующих разработках. Одним из важнейших следствий её разработки является начальная формализация агентно-ориентированной методологии программирования.

На сегодняшний день мультиагентные системы используются для разработки широкого спектра информационных систем[2], среди которых условно можно выделить три основных класса.

Открытые системы. Системы, структура которых может изменяться в процессе их функционирования. Самая большая и самая открытая на сегодняшний день система — Интернет. Социальность и автономность агента позволяют эффективно применять его в качестве элемента открытой системы.

Сложные распределенные системы. Наилучшими современными методами борьбы со сложностью, являются модульность и абстракция. Благодаря высокой степени автономности, агент позволяет уменьшить количество зависимостей между частями системы и, следовательно, упростить её проектирование и реализацию.

Интерактивные системы. Большинство существующих систем, несмотря на графический интерфейс и мощную систему справки, требуют серьезных усилий со стороны потенциального пользователя для их освоения. С помощью агентов можно построить интерактивную систему, которая будет не просто принимать и выполнять команды пользователя, а активно и интеллектуально взаимодействовать с ним, стремясь к достижению общих целей.

В промышленности мультиагентные системы наиболее распространены в следующих областях.

[2]Более подробное описание областей применения мультиагентных систем можно найти в работах [91] и [137]

Автоматизация управления сложными системами. Область, в которой давно и эффективно применяются интеллектуальные агенты. В качестве примеров можно привести платформу ARCHON [171], систему управления производством YAMS [136] и систему управления воздушным движением OASIS [117].

Сбор и обработка информации. Агенты часто используются для реализации систем, собирающих и обрабатывающих информацию из всемирной сети Интернет. Большинство современных поисковых машин реализовано с использованием агентов.

Игры. Сегодня в компьютерных играх противниками игрока человека часто становятся игроки, реализованные как интеллектуальные агенты.

Хороший обзор современных теоретических и практических подходов к построению мультиагентных систем можно найти в работе [183]. Работа [126] содержит более критический анализ, а также задается вопросом, почему агентно-ориентированные методы не столь популярны, как могли бы быть.

Помимо того, что мультиагентные системы активно применяются на практике, они формируют большое и активно развивающееся направление современной науки. В различных частях света регулярно проводятся конференции и семинары, посвященные практическим и теоретическим аспектам разработки МАС.

- Международная конференция по автономным агентам и мультиагентным системам (AAMAS, `www.aamas-conference.org`) является одной из наиболее популярных и значимых конференций, рассматривающей все аспекты разработки мультиагентных систем.

- Международная конференция по мультиагентным системам центральной и восточной Европы (CEEMAS, `www.ceemas.org`). Эта конференция, в

отличии от ежегодной AAMAS, проводится раз в два года на территории центральной или восточной Европы.

- Международный симпозиум по агентам и мультиагентным системам (KES AMSTA, `amsta-07.kesinternational.org`). Относительно молодое мероприятия, так же охватывающее практически весь спектр возникающих при разработке и проектировании МАС задач.

- Семинар по автономным интеллектуальным системам и применению агентов для извлечения знаний (AIS-ADM, `space.iias.spb.su/ais07`). Проводимый в России семинар, посвященный стыку двух активно развивающихся направлений: применения мултиагентных систем для решения задачи извлечения знаний (data mining).

Кроме того, существует несколько сообществ и организаций, занимающихся разработкой, обобщением и стандартизацией подходов к разработке мультиагентых систем. Одним из наиболее значимых среди них является FIPA (Foundation for Intelligence Physical Agent) [59].

Подходы к разработке мультиагентных систем могут сильно отличаться в зависимости от класса задач, для решения которых эта система предназначена. Тем не менее, существует несколько достаточно общих методологий проектирования МАС.

Gaia. Основанная на концепции *роли* методология предложенная Майклом Вулдриджем и Николасом Дженнингсом [184]. Методология предлагает разработку нескольких типов моделей. На этапе анализа системы разрабатываются модель *ролей* и модель *взаимодействий*, которые на этапе проектирования уточняются в моделях *агентов*, *сервисов* и *осведомленности* (acquaintance).

SODA. Методология, ориентированная на разработку приложений для сети

Интернет [133], ключевым понятием которой является понятие *задачи*. Основное внимание в данной методологии уделяется социальности агентов и связанным с ней особенностям проектирования.

Tropos. Методология [44], основное внимание которой уделяется организации мультиагентной системы. Основными понятиями методологии являются понятия *актора* (агента), *цели* и *зависимости*. Фундаментом методологии является система моделей i^* [187].

В последнее время достаточно активно развиваются методологии и платформы для разработки мультиагентных систем с использованием формально-логических методов. Одним из первых агентно-ориентированных языков, основанных на формально-логических методах, является язык $3APL$ [13], получивший очень широкое распространение. Кроме того, в этой области можно выделить языки *Dribble* [179] и платформу $DESIRE$ [52], а одним из последних достижений в этой области является язык $MABEL$ [128]. Подобные подходы представляют наибольший интерес в контексте данной работы.

Цели работы. Основными целями работы является разработка и экспериментальная реализация методов формально-логической спецификации мультиагентных систем с ограничениями на время реакции, соответствующих алгоритмов верификации и планирования с темпорально расширенными целями, а также методов накопления и анализа опыта для оптимизации поведения системы. Для решения поставленной задачи необходимо было разработать ряд методов и реализовать ряд алгоритмов.

- Метод формально-логической спецификации мультиагентных систем с временными ограничениями. Далее соответствующий формализм именуется $MASL$.
- Алгоритм верификации систем по спецификации $MASL$.

- Алгоритм построения мультиагентных планов с темпорально расширенными целями, выраженными в ограниченном варианте логики $MASL$, обеспечивающем эффективную реализацию.

- Метод накопления и анализа опыта для мультиагентных систем.

Структура работы. Во введении приводится обзор структуры работы, обзор предметной области и основных результатов, полученных в этой области другими исследователями, а также формулируются основные цели поставленной работы и мотивации, которыми эти цели обусловлены. Кроме того, во введении описана эволюция методов реализации интеллектуальных агентов и приведен обзор современных подходов к этой задаче.

Глава 1 начинается с введения основных понятий и рассмотрения основных свойств, присущих *интеллектуальному агенту*. В главе 2 описана модель интеллектуального агента, представленного как набор $ag = (S, A, env, see, I_B, bel, brf, I_D, des, drf, plan, prf)$, где

- S есть непустое конечное множество состояний внешней среды;

- A есть непустое конечное множество действий агента;

- $env : S \times A \to 2^S$ есть функция *поведения* внешней среды, сопоставляющая текущему состоянию внешней среды и выбранному агентом действию *множество* возможных следующих состояний внешней среды. Таким образом, действия агента могут влиять на окружающую среду, но не контролировать ее полностью;

- $see \subseteq S \times S$ есть корректное восприятие агентом состояний внешней среды, задающее множество P классов эквивалентности на S;

- $I_B = I_{bel} \times 2^{P \times A \times P}$ есть множество *представлений* агента;

- $bel \in I_B$ есть множество текущих представлений агента;
- $brf : I_B \times A \times P \to I_B$ есть *функция обновления представлений*;
- $I_D = I_{des} \times 2^{Goal}$ есть множество *желаний* агента, где $Goal$ есть множество всех возможных функций-критериев;
- $des \in I_D$ есть множество текущих желаний агента;
- $drf : I_D \times P \to I_D$ есть *функция обновления желаний*;
- $plan = (P, A, I_{pln}, \sigma_{pln}, i_{pln,0})$ есть текущий план агента, представленный конечным автоматом с входным алфавитом P, выходным алфавитом A, множеством состояний I_{pln}, отношением переходов σ_{pln} и начальным состоянием $i_{pln,0}$;
- $prf : I_B \times I_D \times 2^{Plan} \times P \to 2^{Plan}$ есть *функция обновления плана*, где $Plan$ есть множество всех возможных планов.

В главе 3 модель интеллектуального агента расширяется до модели *мультиагентной системы*, представленной множеством агентов, расширенных способностями получения и отправки сигналов для синхронизации действий, а также восприятием действий других агентов. Ключевым понятием, введенным в главе 3, является понятие *коалиции* как набора совместно действующих агентов. Формально коалиция моделируется как единый агент, возможные действия, представления, желания и намерения которого являются объединение соответствующих характеристик входящих в коалицию агентов. Полученные в главах 2 и 3 результаты были опубликованы в работах [1, 6–8].

В главе 4 вводится формальная логика спецификации свойств интеллектуального агента, объединяющая возможности пропозициональной динамической логики PDL и логики ветвящегося времени $RTCTL$ и расширенная следующими дополнительными операторами.

- Bel ϕ — оператор, используемый для описания представлений агента. Семантика этого оператора задается с помощью возможных миров, определяемых восприятием агента $see \subseteq S \times S$ и представлениями агента $bel \subseteq P \times A \times P$.

- Des ϕ — оператор, используемый для описания желаний агента. Семантика этого оператора задается относительно структуры желаний агента des.

- Intend ϕ — оператор, используемый для описаний намерений агента. Семантика этого оператора задается с помощью возможных миров, определяемых восприятием see, представлениями bel и планом $plan$ агента.

Для данной логики определяется два варианта семантики: с динамическим и со статическим ментальным состоянием. В первом случае предполагается, что представления, желания и намерения агента могут измениться в любой момент времени, что должно учитываться при построении множества возможных путей. Этот вид семантики является наиболее общим, но сопряжен с высокой вычислительной сложностью верификации. Во втором случае предполагается, что представления, желания и намерения агента не изменяются, что верно для ограниченных промежутков времени, совпадающих с реализацией определенного плана. Доказано, что для семантики со статическим ментальным состоянием задача верификации является полиномиальной, кроме того, именно семантика со статическим ментальным состоянием является более удобной для формулирования темпорально расширенных целей.

В главе 5 предлагаются символические алгоритмы верификации для логики спецификации интеллектуального агента, а также способы символического представления модели интеллектуального агента с использованием разрешающих диаграмм. Доказаны корректность и полнота этих алгоритмов, а также то,

что их алгоритмическая сложность не превосходит полинома от длины формулы и размера модели.

В главе 6 работы предлагается формальная логика $MASL$, предназначенная для описания свойств мультиагентной системы с временными ограничениями способной к накоплению и анализу опыта. Логика $MASL$ расширяет логику спецификации интеллектуального агента для случай мультиагентной системы. В этой логике вводится *квантор коалиции* $\langle\langle A\rangle\rangle\phi$, указывающий на то, что интерпретацию формулы ϕ следует проводить относительно формальной модели коалиции A как единого агента. Полученные в главах 4 , 5 и 6 результаты были опубликованы в работах [2, 3, 5, 9, 43]. Рассмотрим несколько примеров описания свойств мультиагентной системы.

- $\langle\langle A\rangle\rangle\mathbf{A}\, G\phi$ — коалиция A ведет себя таким образом, что на любом пути всегда выполнено свойство ϕ. Такие свойства называют свойствами *безопасности.*

- $\psi \Rightarrow \mathbf{A} \bigcirc (\phi \Rightarrow \langle\langle B\rangle\rangle \mathsf{Bel}\ (\psi \Rightarrow \mathbf{E} \bigcirc \phi))$ — если из состояния, удовлетворяющего ψ, система перейдет в состояние, удовлетворяющее ϕ, то коалиция B будет знать, что такой переход возможен.

- $\langle\langle A\rangle\rangle\mathbf{A}\, G(\mathsf{Des}\ \phi \Rightarrow \mathbf{A}\, \mathcal{F}_{\leq 10}\psi)$ — коалиция A ведет себя таким образом, что на любом пути всегда при возникновении у нее желания реализовать ϕ не более чем через 10 тактов ϕ будет реализовано. Такие свойства называют свойствами *живучести.*

- $\langle\langle A\rangle\rangle \mathsf{Des}\ \phi \wedge \mathsf{Bel}\ \mathbf{E}\, \mathcal{F}\, \phi \Rightarrow \mathsf{Intend}\ \phi$ — если коалиция A хочет реализовать свойство ϕ и считает, что существует путь, на котором оно может быть реализовано, то она будет пытаться реализовать это свойство.

- $\mathbf{A}\, G(\mathbf{A}\, \mathcal{F}_{\leq 16}\phi) \wedge (\mathbf{A}\, \mathcal{F}_{\leq 16}\neg\phi)$ — система в целом ведет себя таким образом, что свойства ϕ и $\neg\phi$ чередуются с периодом не более 16 тактов.

- $\langle\langle C \rangle\rangle \mathsf{Intend}\ \phi \Rightarrow \mathbf{A}\,\mathcal{F}\,\phi$ — если коалиция C решила реализовать свойство ϕ, то она этого когда-нибудь добьется.

В главе 7 предлагаются символические алгоритмы мультиагентного планирования в ограничениях $MASL$. Входными данными для алгоритма планирование являются описание модели мультиагентной системы и набор формул $MASL$. Результатом алгоритма является план, представленный моделью взаимодействующих конечных автоматов[3], следуя которому система будет удовлетворять требуемым свойствам. Построение плана производится в четыре этапа: предварительная обработка формулы, построение множества состояний автомата, построение общего плана, выделение планов для отдельных агентов из общего плана.

Наиболее интересным является этап построения общего плана, на котором изначальный план, не накладывающий никаких ограничения на действия агента, постепенно уточняется для каждой из подформул формулы–цели через исключение действий, опровергающих эту подформулу. Для уточнения плана используется техника аналогичная технике символической верификации через поиск неподвижной точки. При этом, в отличии от алгоритмов верификации, после построения плана для формулы в целом производится уточнение плана для подформул с учетом ограничений, накладываемых общим планом. Благодаря тому, что план может уточняться параллельно для нескольких формул, возможна распределенная реализация планирующего алгоритма. Доказаны корректность и завершаемость предложенных алгоритмов, кроме того, доказано, что их алгоритмическая сложность не превосходит полинома от размера модели и экспоненты от длины формулы. Полученные в главе 7 результаты были

[3]Модель взаимодействующих конечных автоматов является одним из возможных расширений модели конечных автоматов. Для взаимодействующего конечного автомата, помимо входного и выходного алфавита, определены множества входящих и исходящих сигналов, а также каналы связи, перенаправляющие исходящие сигналы одного автомата во входящие сигналы другого автомата.

опубликованы в работе [4].

В главе 8 предлагается новый подход к решению задачи накопления опыта системой, использующий символическое представление данных в виде разрешающих диаграмм. Опыт агента представляются в виде функции $res : S \times ACS \times S \to \mathbb{N}$ (где ACS есть декартово произведение множеств действий всех агентов системы, а $\mathbb{N}$ есть множество натуральных чисел), представленной с помощью разрешающих диаграмм. Результат функция res в точке $(s, a, s') \in S \times ACS \times S$ описывает сколько раз результатом выполнения системой в состоянии $s \in S$ действия $a \in ACS$ являлось состояние $s' \in S$.

Для предложенного представления опыта описаны такие операции как добавления факта, интеграции опыта нескольких агентов, а также получение описания представлений агента $bel \subseteq S \times ACS \times S$, построенного с использованием порога отсечения $\theta \in [0, 1]$ следующим образом: $bel = \{(s, a, s') \in S \times ACS \times S \mid res(s, a, s') \geq \theta \cdot \max\limits_{s' \in S} res(s, a, s')\}$. Кроме того, предложен ряд методов анализа опыта, оптимизации представления и построения обобщений: проекция редукции разрешающей диаграммы bel на исходную разрешающую диаграмму res, расширенная редукция схожих относительно некоторого критерия ситуаций, а также объединение малознакомых схожих ситуаций.

В заключении работы приведен список основных результатов и обзор потенциальной области применения предложенных методов. Приложение А к работе содержит доказательства полноты и корректности разработанных алгоритмов, а также анализ их алгоритмической сложности.

Приложение Б содержит описание реализации алгоритмов планирования и верификации, их архитектуре, формате входных файлов, а также тестов, использовавшихся для проверки их работоспособности. Алгоритмы верификации и планирования реализованы в виде консольных приложению. В качестве способа описания моделей системы и их свойств используется диалект XML, заданный с помощью XSD-схемы, используемой для автоматической провер-

ки корректности входных файлов. Для преобразования входных файлов в дерево объектов используется платформа .NET 2.0 и язык программирования C#. На основе полученного дерева строится дерево объектов C++, которые используются для проведения основных вычислений. Использование гибридного C#/C++ решения позволяет использовать богатый набор классов .NET Framework для анализа и разбора XML, не потеряв при этом в производительности основного алгоритма. В качестве пакета разрешающих диаграмм используется пакет университета Колорадо $CUDD$, портированный под операционную систему Windows с использованием набора библиотек и компиляторов MinGW. Все использованные в разработке программы, библиотеки и пакеты свободно распространяются для научных целей и некоммерческого использования.

Основные результаты. В рамках диссертации разработана математическая модель интеллектуального агента, описывающая такие аспекты деятельности агента, как взаимодействие с внешней средой, неполнота информации, способность к анализу и накоплению опыта, ограничения на время реакции, управляемое целями поведение и планирование действий. Кроме того, разработана математическая модель мультиагентной системы, включающая такие аспекты деятельности сообщества интеллектуальных агентов, как коммуникация, координация и кооперация агентов.

Для спецификации свойств математической модели интеллектуального агента разработан логический формализм, позволяющий описывать свойства поведения агента, а также его ментального состояния. Кроме того, разработан логический формализм для спецификации мультиагентной системы с временными ограничениями $MASL$, включающий логику спецификации интеллектуального агента и расширяющий её средствами описания свойств определенных групп агентов (коалиций). Разработаны и реализованы алгоритмы верификации (“проверки модели”, model checking) систем по спецификациям $MASL$, имеющие

полиномиальную сложность и использующие технику символической верификации. Корректность, завершаемость и полнота алгоритмов доказаны, кроме того, доказано, что их алгоритмическая сложность не превосходит полинома от размера модели и длины формулы.

Разработаны и реализованы алгоритмы построения мультиагентных планов, удовлетворяющих спецификации $MASL$ и использующие технику символического планирования. Корректность и завершаемость алгоритмов доказаны, кроме того, доказано, что их алгоритмическая сложность не превосходит полинома от размера модели и экспоненты от длины формулы.

Кроме того, предложены методы накопления и анализа опыта в символической форме с помощью разрешающих диаграмм, позволяющих в относительно компактной форме хранить весь опыт системы и быстро преобразовывать этот опыт в структуры, удобные для символических алгоритмов построения планов и верификации.

Глава 1

Основные понятия

1.1. Понятие *агент*

Понятие “агент” в последнее время было адаптировано ко многим областям как прикладного и системного программирования, так и к исследованиям в областях искусственного интеллекта и распределенных интеллектуальных систем. Причем в каждом конкретном случае понятию придавалось несколько различное значение. В этом разделе мы зафиксируем понятие “агента”, которое будет использоваться далее в работе.

Основным отличительным свойством агентов является то, что агенты совершают *действия*. Далее часто утверждается, что агенты не просто совершают действия, но они действуют *автономно* и *рационально*. Под *автономностью* обычно понимают, что агент действует без прямого вмешательства человека или другой управляющей сущности.

С понятием *рациональности* все не так просто, но часто под рациональностью понимают стремление агента оптимизировать значение некоторой оценочной функции. Мера рациональности неявно указывает на то, что агент имеет *цели*, которых агент “хочет” достичь, и *представления* о внешнем мире, на которые агент опирается при выборе действия.

Еще одним важным свойством агента является то, что он помещен во *внешнюю среду*, с которой он способен *взаимодействовать*. Как правило, среда не контролируется агентом, т.е. он способен лишь *влиять* на неё, но не контролировать полностью.

В итоге, можно сформировать следующее определение агента, адаптированное многими современными исследователями.

Определение 1 Агент — *вычислительная система, помещенная во внешнюю среду, способная взаимодействовать с ней, совершая автономные рациональные действия для достижения целей.*

1.2. Интеллектуальный агент

Очевидно, что перечисленные выше свойства не являются достаточными для определения интеллектуального агента, так как не предполагают явно *гибкости* поведения. Обычно для того, чтобы считаться "интеллектуальным" агент должен обладать следующими свойствами.

- *Реактивность (reactivity)* — агент должен *ощущать* внешнюю среду и *реагировать* на изменения в ней, совершая действия, направленные на достижение целей.
- *Проактивность (pro-activeness)* — агент должен показывать управляемое целями поведение, *проявляя инициативу*, совершая действия направленные на достижение целей.
- *Социальность (social ability)* — агент должен взаимодействовать с другими сущностями внешней среды (другими агентами, людьми и т.д.) для достижения целей.

Любое из первых двух свойств может быть достигнуто достаточно легко. Например, свойством *проактивности*, в широком смысле, обладает любой компилятор, основная цель которого, очевидно, сформировать низкоуровневый объектный код на основе программного кода на языке программирования высокого уровня. По сути, любая функция может быть рассмотрена как система, целью которой является преобразование некоторых входных данных в выходные. Однако, очевидно, что изменение во входных данных функции *во время* её работы

практически наверняка приведет к краху или, как минимум, несоответствию выходных данных входным. Т.е. ни функции, ни компиляторы (которые, безусловно, можно рассматривать как разновидность функций) не обладают свойством *реактивности*.

С другой стороны, только *реактивные* системы (т.е. системы которые только реагируют на изменения во внешней среде) тоже достаточно просты – даже самая примитивная продукционная система демонстрирует реактивность. Совмещение же в системе обоих свойств в нужных пропорциях является непростой задачей. Будет не слишком эффективно, если агент жестко следует сценарию достижения цели, не реагируя на изменения во внешней среде и не обладая способностью заметить необходимость корректировки плана. Но так же не эффективно будет и поведение, ограниченное лишь реакцией на поступающие извне стимулы, без какого-либо планирования целенаправленных действий.

На самом деле описанная проблема настолько сложна, что даже далеко не все люди способны эффективно её решать. Очень часто можно увидеть человека, который кидается на каждую подвернувшуюся возможность, но никогда не концентрируется на этой возможности достаточное время, чтобы полноценно её реализовать. Но так же часто встречаются люди, которые, однажды поставив цель и сформировав план, будут пытаться четко ему следовать, не замечая изменений в ситуации, требующих пересмотра планов или целей.

Что же касается третьего свойства, *социальности*, то оно тоже не так просто, как может показаться на первый взгляд. С одной стороны, каждый день миллионы компьютеров взаимодействуют между собой, обмениваясь пакетами бинарных данных. Но такое поведение еще нельзя считать *социальным*. Помимо коммуникации, социальное поведение должно включать кооперацию с другими сущностями, заключающуюся в разделении целей между отдельными сущностями, совместное планирование и координацию действий, направленных на достижение общих целей. Социальное поведение как минимум предполагает

наличие у агента представлений о целях других сущностей и том, как они планируют этих целей достичь. Обзор общих подходов к организации социального поведения агентов можно найти в работе [82].

Глава 2

Модель интеллектуального агента

Модель исследуемой системы является фундаментом для методов формальной спецификации, верификации и автоматического тестирования. Кроме того, на основе такой модели может быть разработана методология проектирования и реализации систем данного класса.

В данной главе рассматриваются существующие подходы к построению формальных моделей и вводится математическая модель интеллектуального агента, в основу которой положена модель [92], расширенная средствами явного описания таких аспектов деятельности агента, как *целеполагание*, *прогнозирование*, *планирование* и *накопление опыта*.

2.1. Формальные модели распределенных систем

Большинство методов формальной спецификации и верификации включают в себя три основных компонента:

- формальную модель верифицируемой системы, описывающую *состояния* системы и ее *переходы*;
- метод формальной спецификации свойств этой модели, как правило, в некотором логическом языке;
- метод проверки соответствия модели спецификации.

В данном разделе мы проводится обзор существующих формальных моделей распределенных систем, используемых для их верификации.

2.1.1. Конечные автоматы

Конечный автомат (finite-state machine, automaton) является одной из старейших [62] формальных моделей вычислительных систем, активно использующейся в различных вариантах и в настоящее время [180].

Простой конечный автомат

Простой конечный автомат (Finite-State Machine, FSM) определяется конечным множеством состояний, алфавитами входных и выходных символов, функцией перехода и вывода, а также начальным состоянием и множеством конечных состояний. Формально конечный автомат это набор $M = (I, O, S, \sigma, s_0, F)$, где

- I есть конечный непустой алфавит входных символов.
- O есть конечный непустой алфавит выходных символов.
- S есть конечное непустое множество состояний автомата.
- $\sigma : S \times I \rightarrow 2^{S \times O}$ есть функция перехода и вывода, сопоставляющая текущему состоянию автомата и текущему наблюдаемому символу входного алфавита множество возможных изменений внутреннего состояния и символов, записываемых в вывод автомата. В случае если возвращаемое множество всегда состоит из одного элемента ($\sigma : S \times I \rightarrow S \times O$), такой автомат является *детерминированным*, иначе автомат является *недетерминированным*.
- $s_0 \in S$ есть начальное состояние автомата.
- $F \subseteq S$ есть множество конечных состояний автомата.

Конечный автомат начинает работу в состоянии s_0 и, получая на вход слово во входном алфавите $i \in I^*$, последовательно проходит через все его символы, совершая предписанные функцией σ переходы и постепенно формируя выходное слово. Автомат заканчивает работу при достижении одного из конечных состояний из множества F или при достижении конца входного слова. Выходом автомата считается полученное выходное слово и конкретное конечное состояние, в которое пришел автомат. В случае недетерминированного автомата мы получаем *множество* выходов, а в случае детерминированного — всегда один.

Конечные автоматы получили широкое распространение благодаря простоте и наглядности используемых в них концепций. Однако, эта концепция не лишена недостатков, среди которых можно выделить следующие ограничения.

- Взрыв пространства состояний. При моделировании реальных систем с помощью конечных автоматов размер множества состояний очень быстро, часто экспоненциально, растет. Этот недостаток частично удается устранить, используя символическое представление для множества состояний и функции переходов.

- Отсутствие модульности структуры. Декомпозиция системы на несколько модулей является основным средством борьбы со сложностью, однако концепция конечного автомата не предоставляет никаких средств для декомпозиции системы. С точки зрения моделирования распределенных систем этот недостаток является критическим.

Расширенный конечный автомат

Концепция расширенного конечного автомата (Extended Finite-State Machine, EFSM) [164] позволяет более наглядно моделировать системы с большим числом состояний, представимым в виде набора переменных. Формально EFSM это набор $M = (I, O, S, D, \sigma, \langle s_0, x_0 \rangle, F)$, где

- I есть конечный непустой алфавит входных символов.
- O есть конечный непустой алфавит выходных символов.
- S есть конечное непустое множество состояний автомата.
- $D = D_1 \times \ldots \times D_n$ это n-мерное множество, описывающее возможные значения переменных
- $\sigma : S \times D \times I \to 2^{S \times D \times O}$ есть функция переходов, определяющая для текущего внутреннего состояния $s \in S$, текущих значений переменных $d \in D$ и наблюдаемого входного символа $i \in I$ сопоставляет множество возможных изменений внутреннего состояния, значений переменных и выходной символ. Так же как и в случае простого конечного автомата, расширенный конечный автомат может быть детерминированным или недетерминированным.
- $\langle s_0, x_0 \rangle \in S \times D$ есть начальное состояние автомата и начальные значения для всех переменных.
- $F \subseteq S$ есть множество конечных состояний автомата.

Несложно заметить, что концепция расширенного конечного автомата формально эквивалента концепции простого конечного автомата, но она позволяет более наглядным образом моделировать системы, состояния которых определяются набором переменных.

Взаимодействующие конечные автоматы

Концепция взаимодействующих конечных автоматов (Communicating Finite-State Machine, CFSM) [38] позволяет в более удобной форме моделировать многокомпонентные системы. Взаимодействующий конечный автомат может быть

представлен как обычный конечный автомат с несколькими очередями входных и выходных символов. *Сеть* взаимодействующих конечных автоматов включает набор автоматов и *коммуникационных каналов*, где каждый канал объединяет одну выходную очередь одного автомата с одной входной очередью другого автомата. Формально взаимодействующий конечный автомат это набор $M = (I, O, S, \sigma, s_0, F)$, где

- I есть конечный непустой алфавит входных символов.
- O есть конечный непустой алфавит выходных символов.
- S есть конечное непустое множество состояний автомата.
- $\sigma : S \times I^n \to 2^{S \times O^m}$ есть функция перехода и вывода, сопоставляющая текущему состоянию автомата и текущим наблюдаемым символам входного алфавита во всех очередях множество возможных изменений внутреннего состояния и символов, записываемых в выходные очереди автомата. В случае если возвращаемое множество всегда состоит из одного элемента ($\sigma : S \times I^n \to S \times O^m$), такой автомат является *детерминированным*, иначе автомат является *недетерминированным*.
- $s_0 \in S$ есть начальное состояние автомата.
- $F \subseteq S$ есть множество конечных состояний автомата.

Сеть взаимодействующих конечных автоматов является набором $NM = (M_1, \ldots, M_q, C_1, \ldots, C_p)$, где каждый M_i есть взаимодействующий конечный автомат, а каждый C_i есть канал связи. При этом каждый канал связи представляется четверкой (i, k, j, l), гдеi и j есть номера автоматов, а k и l есть номера очередей этих автоматов. Таким образом, канал связи (i, k, j, l) сопоставляет выходную очередь k автомата i входной очереди l автомата j, перенаправляя

тем самым выходные символы одного автомата на вход другому. При этом каждой входной очереди автомата может соответствовать не более одной выходной очереди других автоматов. Не соединенные каналами очереди определяют входные и выходные данные системы в целом.

Очевидным образом концепцию сети взаимодействующих конечных автоматов можно расширить и до сети взаимодействующих *расширенных* конечных автоматов. Возможны и другие модификации этой модели, например, введение асинхронных каналов связи, приоритетов сообщений, понятия времени и т.д.

Именно эту формальную модель используют такие широко известные методы моделирования как SDL [87] и ESTELLE [57].

Конечный автомат со временем

Концепция конечного автомата со временем (Timed Automaton) [19, 20] была предложена Р.Алуром для моделирования систем реального времени, и расширила концепцию конечного автомата переменными-часами, увеличивающими свое значение при переходах автомата, а также ограничениями по времени, при которых те или иные правила переходов автомата являются действующими.

Формально конечный автомат со временем представим как набор $M = (I, O, S, X, \sigma, s_0)$, где

- I есть конечный непустой алфавит входных символов.
- O есть конечный непустой алфавит выходных символов.
- S есть конечное непустое множество состояний автомата.
- $X = \mathbb{N}_1 \times \ldots \times \mathbb{N}_n$ есть n-мерное множество, описывающее значения переменных-часов.
- $\sigma \subseteq S \times I \times 2^X \times S \times O \times X$ есть множество *триггеров*, где каждый триггер для состояния $s \in S$ и входного символа $i \in I$, при условии

что переменные-часы удовлетворяют ограничению $c \subseteq X$, сопоставляет следующее состояние автомата $s' \in S$, выходной символ $o \in O$, а также новые значения переменных-часов $clocks \in X$.

- $s_0 \in S$ есть начальное состояние. Предполагается, что изначально все переменные-часы установлены в 1.

2.1.2. Алгебры процессов

Алгебра процессов (process algebra) или исчисление процессов (process calculus), это подход, в котором система описывается как набор *процессов*, исполняющихся параллельно и взаимодействующих друг с другом через *каналы связи*. Алгебра задает набор элементарных процессов, а также ряд операторов, с помощью которых можно определять новые, более сложные процессы. Большинство алгебр процессов включает следующие операторы в той или иной форме.

- Параллельная композиция процессов. В этом случае новый процесс заключается в параллельном исполнении двух или более исходных процессов, которые могут при этом взаимодействовать друг с другом или с другими процессами. Например, если P и Q являются процессами, то $P \mid Q$ является новым процессом, являющийся параллельной композицией исходных

- Операторы чтения и записи в канал связи. В этом случае процесс определяется как ожидание получения сигнала или отправка сигнала по определенному каналу связи. Обычно отправка сообщения a по каналу x определяется как $x\langle a \rangle$, а прием сообщения как $x(a)$.

- Последовательная композиция процессов. В этом случае новый процесс формируется как последовательное исполнение нескольких процессов. Например, конструкция $x(a) \cdot P$ конструирует процесс, который ожидает прихода сигнала по каналу x, после чего исполняет процесс P.

- Сокрытие. При моделировании распределенных систем для снижения их сложность часто используют сокрытие каналов коммуникации от некоторых процессов. Например, конструкция $P \setminus \{x\}$ обозначает исполнение процесса P, которому недоступен коммуникационный канал x.

- Рекурсия и репликация. Эти две конструкции используются для построения неостанавливающихся (non-terminating) процессов. Рекурсия используется для указания, что в определенный момент процесс запускает сам себя заново, а репликация $!P$ используется для обозначения счетного числа процессов P, исполняющихся параллельно.

Помимо вышеперечисленных операторов, алгебра процессов может содержать операторы недетерминированного выбора, условный оператор, оператор прерывания, таймаута или другие операторы, в зависимости от особенностей моделируемой системы.

Одна из первых алгебр взаимодействующих процессов Communicating Sequential Processes (CSP) [78] была предложена Чарльзом Хоаром еще в 1978 году, другая алгебра Calculus of Communicating Systems (CCS) [124] была предложена Робином Милнером в 1982 году. С тех пор и по сей день это направление активно развивается и его история хорошо изложена в работе [32].

На сегодняшний день существует большое количество расширений классических алгебр процессов, позволяющих учитывать особенности моделирования систем реального времени, например Timed CSP [70], или асинхронную коммуникацию, например, Receptive Process Theory [101]. Также существует много промышленных инструментов, использующих те или иные варианты алгебры процессов для моделирования распределенных систем.

2.1.3. Вероятностные модели

Во многих современных распределенных системах присутствует элемент неопределенности, возникающий из-за взаимодействия с внешними по отношению с системой сущностями, например, с пользователем. Для моделирования подобных систем используются различные вероятностные модели, например, дискретные цепи Маркова (Discrete-Time Markov Chains, DTMC) [186], непрерывные цепи Маркова (Continuous-Time Markov Chains, CTMC) [185], а также Марковские процессы принятия решения (Markov Decision Process, MDP) [149].

В данной работе мы приведем формальное определение Марковского процесса принятия решения (MDP), так как именно эта модель наиболее часто используется для моделирования распределенных, в том числе и мультиагентных систем. MDP представляется тройкой $\langle S, s_0, A, \sigma \rangle$, где

- S есть непустое конечное множество состояний;
- $s_0 \in S$ есть начальное состояние;
- A есть непустое конечное множество действий;
- $\sigma : S \times A \to S \times [0, 1]$ есть функция перехода, сопоставляющая текущему состоянию и действию системы распределение вероятности изменения состояния системы.

Система, моделируемая с помощью MDP, может выбирать те или иные действия и тем самым влиять на изменение состояния. Именно наличие понятия действия в явном виде делает MDP привлекательным инструментом с точки зрения моделирования взаимодействия агента и внешней среды.

2.2. Модель интеллектуального агента

Для начала, дадим определение одной из наиболее абстрактных моделей интеллектуального агента — модели *агента с состоянием*. В этом случае агент рассматривается как набор $AG = (S, A, env, I, refine, action)$, где

- S есть непустое конечное множество состояний внешней среды;
- A есть непустое конечное множество действий агента;
- $env : S \times A \to 2^S$ есть функция *поведения* внешней среды, сопоставляющая текущему состоянию внешней среды и выбранному агентом действию *непустое множество* возможных следующих состояний внешней среды. Таким образом, действия агента могут влиять на окружающую среду, но не контролировать её полностью;
- I есть непустое конечное множество *внутренних* состояний агента;
- $refine : I \times S \to I$ есть *функция обновления состояния*, сопоставляющая предыдущему внутреннему состоянию и новому состоянию внешней среды новое внутреннее состояние агента;
- $action : I \to A$ есть *функция принятия решения*, сопоставляющая текущему внутреннему состоянию агента некоторое действие.

Описание агента в таком виде не дает никакой информации о его внутренней структуре и сводит описание агента практически к модели конечного автомата с входным алфавитом S, множеством состояний I и выходным алфавитом A. Далее в этой главе мы уточним структуру внутреннего состояния агента и процессы его преобразований.

Определение 2 *Цепочка пар состояний* $\lambda \in (S \times I)^+$ *является* возможной историей *взаимодействия агента* $AG = (S, A, env, I, refine, action)$ *с внешней средой, если для нее выполнены следующие свойства.*

- $\forall j \in \mathbb{N} : \lambda[j+1]|_S \in env(\lambda[j]|_S, action(refine(\lambda[j]|_I, \lambda[j]|_S)))$. *Это условие требует того, чтобы реакция внешней среды соответствовала допустимой реакции на действия агента, определяемые его функциями* action *и* $refine$.[1]

- $\forall j \in \mathbb{N} : \lambda[j+1]|_I = refine(\lambda[j]|_I, \lambda[j]|_S)$. *Это условие требует того, чтобы внутреннее состояние агента изменялось в соответствии с его функцией обновления состояния.*

2.3. Знания и представления агента

Интеллектуальный агент обладает некоторой информацией о внешней среде, а также активно использует эту информацию при взаимодействии с внешней средой. Именно эту информацию и называют *знаниями* или *представлениями*[2] агента. В случае *изолированного* интеллектуального агента (действующего отдельно, а не в составе мультиагентной системы) можно выделить два основных блока представлений:

- представления о текущем состоянии внешней среды, часто именуемые *восприятие*;

- представления о закономерностях поведения внешней среды, позволяющие агенту прогнозировать последствия своих действий;

[1] С помощью нотации $\lambda[j]$ мы обозначаем j-ый элемент последовательности, а с помощью нотации $\lambda[j]|_S$ — соответствующий элемент пары (s_j, i_j).

[2] Так как в общем случае эта информация может не соответствовать реальности, для её обозначения чаще используют термин “представления” (beliefs), чем “знания” (knowledge).

2.3.1. Восприятие

Восприятие описывается отношением $see \subseteq S \times S$, определяющим для данного состояния $s \in S$ множество состояний неотличимых от s для данного агента: $see(s) \triangleq \{s' \in S \mid (s, s') \in see\}$.

Семантика отношения see, по сути, совпадает с семантикой отношения допустимости $\Re$ в семантике возможных миров Крипке (см. раздел 4.2.2). Состояния s' допустимо вместе с s (неотличимо от s), если $(s, s') \in S$.

Отношение $see \subseteq S \times S$ назовем *корректным восприятием*, если оно является отношением эквивалентности, т.е. если для него выполнены следующие три свойства.

- Рефлексивность: $\forall s \in S : (s, s) \in see$. Состояние s рассматривается агентом как одна из возможных альтернатив при восприятии состояния s, т.е. одна из альтернатив всегда соответствует реальности (аналогично аксиоме знаний T в разделе 4.2.2).

- Симметрия: $\forall s, s' \in S : (s, s') \in see\ implies\ (s', s) \in see$. Если агент может спутать состояние s с состоянием s', то он может спутать и состояние s' с состоянием s, т.е. можно говорить о том, что состояния s и s' неотличимы друг от друга.

- Транзитивность: $\forall s, s', s'' \in S : (s, s') \in see\ and\ (s', s'') \in see\ implies\ (s, s'') \in see$. Если агент не может отличить состояние s от s', а состояние s' от состояния s'', то он не может отличить s от s''.

Корректное восприятие, являясь отношением эквивалентности, разбивает множество S на множество классов эквивалентности $S\|_{see}$, которое далее будем обозначать как P. Мощность множества классов эквивалентности $|P|$ позволяет судить о сенсорных возможностях агента — чем больше эта мощность, тем четче агент способен воспринимать внешнюю среду. При $|P| = |S|$ агент

обладает совершенными сенсорными способностями и может отличить любые два различные состояния внешней среды[3]. В другом предельном случае, когда $|P| = 1$, сенсорные способности у агента отсутствуют — он не способен отличить ни одно состояние окружающей среды от другого.

2.3.2. Представления о поведении внешней среды

Для моделирования представлений агента о поведении внешней среды введем отношение $bel \subseteq S \times A \times S$. Если некоторая тройка (s, a, s') входит в отношение bel ($(s, a, s') \in bel$), то, по представлениям агента, при выполнении действия a в состоянии внешней среды s внешняя среда может перейти в состояние s'.

Таким образом, представления агента по своей структуре аналогичны описанию поведения внешней среды env и, по сути, являются моделью внешней среды с точки зрения агента. Если для любых $s, s' \in S$ и $a \in A$ выполнено тождество $s' \in env(s, a)$ тогда и только тогда, когда $(s, a, s') \in bel$, можно сказать, что представления агента полностью соответствуют реальности (в этом случае представления можно считать знаниями). Множество всех возможных представлений обозначим $Bel(S, A) \triangleq 2^{S \times A \times S}$.

Представления агента могут обновляться на каждом шаге взаимодействия с внешней средой посредством *функции обновления представлений* (belief revision function):

$$brf : I_{bel} \times Bel(S, A) \times A \times P \to I_{bel} \times Bel(S, A), \tag{2.1}$$

сопоставляющей части текущего внутреннего состояния агента $i_{bel} \in I_{bel}$, текущим представлениям $bel \in Bel(S, A)$, совершенному агентом действию $a \in A$ и восприятию нового состояния внешней среды $p \in P$ новую часть внутреннего состояния $i'_{bel} \in I_{bel}$ и новое отношение представлений $bel' \in Bel(S, A)$.

[3]Заметим, что это утверждение верно только для конечных множеств S и P.

В результате, процесс обновления представлений агента моделируется детерминированным конечным автоматом с входным алфавитом $A \times P$ и множеством внутренних состояний $I_{bel} \times Bel(S, A)$. Множество $I_B = I_{bel} \times Bel(S, A)$ является частью множества внутренних состояний агента I в том смысле, что $I = I_B \times I'$ (где I' есть некоторая дополнительная часть состояния неопределенной пока структуры).

Для иллюстрации процесса обновления представлений агента рассмотрим пример агента с *абсолютной памятью*. В этом случае множество I_{bel} может совпадать с множеством возможных восприятий P и использоваться для хранения восприятия предыдущего состояния внешней среды. В этом случае функция brf может вычисляться по следующему алгоритму:

$$brf(i_{bel}, bel, a, p') = (i'_{bel} = p', bel' = bel \cup p \times \{a\} \times p')\,. \tag{2.2}$$

Таким образом, после каждой итерации взаимодействия с внешней средой в базу представлений добавляется множество фактов вида (s, a, s'), где $s \in p$, а $s' \in p'$ (напомним, что каждое восприятие p представляет собой класс эквивалентности на множестве S). Добавленные в базу знаний факты описывают результаты только что прошедшей итерации взаимодействия с внешней средой.

2.4. Цели и желания агента

Рациональность поведения является основным свойством агента, в тоже время именно это свойство сложнее всего поддается формализации. Многие исследователи отождествляют рациональность с *управляемым целями поведением*, подразумевая, что агент не просто взаимодействует с внешней средой, а пытается при этом достичь некоторых *целей*.

К моделированию понятия цели агента также существует несколько подходов. Самым простым примером цели может служить некоторое подмножество

состояний внешней среды $G \subseteq S$, одного из которых агенту необходимо достичь. *Расширенными* целями часто называют множество конечных цепочек состояний внешней среды $G \subseteq S^+$. Расширенная цель считается достигнутой в том случае, если история взаимодействия агента с внешней средой имеет конечный префикс, состояния внешней среды в котором совпадают с состояниями одной из цепочек множества G.

Описанные выше цели относились к целям с *конечным горизонтом*, т.е. к целям, которые могут быть однажды достигнуты. Другим видом целей являются *инвариантные свойства*, которые должны выполняться на протяжении всего, потенциально бесконечного, взаимодействия агента с внешней средой[4]. Инвариантные свойства можно описать помощью множества состояний внешней среды $V \subseteq S$, подразумевая, что свойство выполнено, пока внешняя среда находится в одном из этих состояний. По аналогии с расширенными целями, можно описать и расширенный вариант инвариантного свойства с помощью множества бесконечных цепочек $V \subseteq S^*$. В этом случае свойство считается выполненным, если состояния внешней среды в истории взаимодействия агента с внешней средой являются префиксом одной из цепочек в множестве V.

Помимо целей, связанных с внешней средой, часто рассматривают цели, связанные с приобретением знаний (эпистемологические цели). Такая цель считается достигнутой в том случае, если знания агента расширяются соответствующим образом.

Одним из наиболее общих описаний цели агента является описание функции-критерия $goal : (P \times Bel(S, A))^* \rightarrow \{completed, continue, failure\}$, сопоставляющей цепочке пар $(p, bel) \in P \times Bel(S, A)$ одно из следующих значений:

- *completed* в том случае, если цель окончательно достигнута;
- *continue* в том случае, если до сих пор история взаимодействия соответ-

[4]Цели такого вида часто называют свойствами безопасности (safety properties).

ствовала цели, но окончательно цель не достигнута;

- $failure$ в том случае, если цель не была достигнута и не может быть достигнута в будущем;

На вход функции-критерию подается часть истории взаимодействия агента со средой, содержащая информацию о восприятии соответствующих состояний внешней среды и информацию о представлениях агента на данный момент (эта часть необходима для моделирования эпистемологических целей). Результатом функции является флаг, показывающий достигнута ли цель и возможно ли её достижение в будущем.

Существует множество способов описания подобных функций-критериев: с помощью регулярных выражений и конечных автоматов, с помощью набора высказываний в темпоральной или динамической логике, и т.д. На этом этапе мы не будем уточнять конкретный способ описания функции-критерия.

Как правило, агент преследует сразу несколько целей, а также может добавлять новые цели и удалять существующие. Важно отметить, что у ограниченного ресурсами агента не всегда есть возможность достичь всех своих целей, поэтому множество всех целей агента называют *желаниями* (desires) агента, а множество тех целей, которые агент собирается реализовать — *намерениями* (intentions) агента.

Обозначим множество всех возможных функций-критериев для множества восприятий P, множества состояний внешней среды S и множества действий агента A как $Goal(P, S, A)$. В этом случае желания агента являются конечным подмножеством $des \subseteq Goal(P, S, A)$ множества всех возможных функций-критериев. Процесс обновления агентом своих желаний можно, по аналогии с обновлением представлений, смоделировать соответствующей функцией drf (desires revision function):

$$drf : I_{des} \times 2^{Goal(P,S,A)} \times P \to I_{des} \times 2^{Goal(P,S,A)}, \tag{2.3}$$

сопоставляющей части текущего внутреннего состояния $i_{des} \in I_{des}$, текущему набору целей $des \subseteq Goal(P, S, A)$ и восприятию текущего состояния внешней среды $p \in P$ новую часть внутреннего состояния $i'_{des} \in I_{des}$ и новый набор желаний $des' \subseteq Goal(P, S, A)$. Так же как и в случае с представлениями, множество $I_D = I_{des} \times 2^{Goal(P,S,A)}$ является частью множества внутренних состояний агента I в том смысле, что $I = I_D \times I'$.

2.5. Планирование

Одним из отличительных свойств *интеллектуального* агента является *проактивность*, что подразумевает способность агента к построению планов взаимодействия с внешней средой.

План агента можно рассматривать как конечный автомат $plan = (P, A, I_{pln}, \sigma_{pln}, i_{pln,0})$, где

- P есть входной алфавит автомата, совпадающий со множеством возможных восприятий агентом состояний внешней среды;
- A есть выходной алфавит автомата, совпадающий с множеством действий агента;
- I_{pln} есть множество внутренних состояний автомата, являющееся частью множества внутренних состояний агента I в том смысле, что $I = I_{pln} \times I'$;
- $\sigma_{pln} \subseteq P \times I_{pln} \times I_{pln} \times A$ есть отношение переходов, определяющее по восприятию p текущего состояния внешней среды и текущему внутреннему состоянию плана i_{pln} следующее состояние для плана i'_{pln} и действие, которое следует выполнить агенту;
- $i_{pln,0} \in I_{pln}$ есть начальное состояние автомата.[5]

[5]В данном случае используется модель конечного автомата без явно выделенных финальных состояний.

Определение 3 *Возможным результатом плана* $plan$ *в начальном состоянии внешней среды* s_0 *и начальном состоянии плана* $i_{0,pln} = i_0|_{I_{pln}} \in I_{pln}$ *по представлениям агента* $bel = i_0|_{Bel(S,A)} \in Bel(S,A)$ *назовем множество цепочек* $out_{plan,bel}(s_0, i_{0,pln}) = \{\lambda \in (S \times I_{pln} \times A)^+\}$, *где для каждой цепочки* $\lambda \in out_{plan,bel}(s_0, i_{0,pln})$ *выполнено,* $\lambda[0]|_S = s_0$ *и* $\lambda[0]|_{I_{pln}} = i_{0,pln}$ *и для любого* $j < |\lambda|$ *если существуют такие* $s \in S$, $a \in A$ *и* $i'_{pln} \in I_{pln}$, *что* $(\lambda[j]|_S, a, s) \in bel$ *и* $(see(\lambda[j]|_S), \lambda[j]|_{I_{pln}}, i'_{pln}, a) \in \sigma_{pln}$, *то* $j+1 < |\lambda|$ *и существует такое* $a' \in A$, *что*

- $(\lambda[j]|_S, a', \lambda[j+1]|_S) \in bel$.
- $(see(\lambda[j]|_S), \lambda[j]|_{I_{pln}}, \lambda[j+1]|_{I_{pln}}, a') \in \sigma_{pln}$.

Заметим, что результат плана по представлениям агента может отличаться от реальных результатов плана в том случае, если представления агента расходятся с реальностью.

Будем говорить, что план $plan$ реализует цель $goal$ по представлениям агента $bel \in I_B$ в текущем состоянии среды, воспринимаемом как p_0, если для любой цепочки $\lambda \in out(plan, bel, p_0)$ выполнено $goal(\lambda|_{P \times I_B}) \neq failure$.

Множество всех возможных планов обозначим как 2^{Plan}. В тех случаях, когда это не будет вести к двусмысленности, будем отождествлять план с его отношением переходов.

План агента не является статичным — при изменении желаний или представлений агента он может быть перестроен. Этот процесс мы смоделируем *функцией обновления плана* (plan revision function):

$$prf : I_B \times I_D \times 2^{Plan} \times P \to 2^{Plan}, \tag{2.4}$$

сопоставляющей текущим представлениям агента $bel \in I_B$, текущим желаниям агента $des \in I_D$, текущему плану $plan \in 2^{Plan}$ и восприятию текущего состояния

Финальными состояниями считаются состояния, для которых не определено дальнейшее поведение.

внешней среды $p \in P$ новый план $plan' \in 2^{Plan}$. В большинстве случаев новый план будет совпадать со старым, однако, если желания или представления агента радикально изменились или произошедшие во внешней среде изменения не были предусмотрены исходным планом, может быть построен новый план.

Кроме того, с каждым планом можно ассоциировать некоторое множество целей $int \subset Goal(P, S, A)$, которые реализуются планом по представлениям агента и которые содержались во множестве желаний агента des на момент построения плана. Такие цели мы будем называть *намерениями* агента[6].

2.6. BDI-агент

Корни BDI-архитектуры (beliefs-desires-intentions) интеллектуальных агентов лежат в философских подходах к анализу мыслительной деятельности человека, того, как люди на практике принимают решения, о том что им следует делать. Можно выделить следующие отдельные этапы в принятии решения BDI-агентом: сначала агент должен понять, *чего* он хочет, затем определить *какие* цели из желаемых он будет пытаться реализовать, а затем понять, *как* он будет реализовывать выбранные целей. При этом в состоянии агента четко разграничиваются следующие компоненты.

Представления (beliefs). Некоторая информация о закономерностях и текущем состоянии внешней среды, которой располагает агент. При этом предполагается, что эта информация может быть ошибочной и неполной, поэтому её можно рассматривать только как представления, но не как достоверные знания. Способ описания представлений агента описан в разделе 2.3.

Желания (desires). Множество всех целей, которых агент хотел бы добить-

[6]Заметим, что множество намерений не может быть противоречивым, поэтому в данном случае обязательно строгое включение $int \subset Goal(P, S, A)$.

ся. Это множество может быть большим и противоречивым. Маловероятно, что агент, ограниченный ресурсами, сможет реализовать все свои желания. Способ описания желаний агента описан в разделе 2.4.

Намерения (intentions). Множество тех целей, которых агент решил добиться. Сформированное множество целей должно быть *выполнимо* по представлениям агента, т.е. все намерения агента должны быть достижимы в совокупности. Под выполнимостью намерений понимается наличие у агента плана (см. раздел 2.5), ведущего к осуществлению всех намерений.

Ключевым понятием процесса принятия решения в BDI-архитектуре является именно *намерение*. В целом можно выделить следующие свойства намерений.

- Намерения задают направление деятельности — агент пытается найти действия, способные осуществить намерения и выполнить их.
- Намерения ограничивают будущий выбор — агент не может формировать новые намерения, несовместимые с уже принятыми, т.е. ведущие к *невыполнимости* множества намерений.
- Намерения имеют долгое время жизни — если агент сформировал план реализации намерения, но он не привел к успеху, то агент будет формировать новые планы и пытаться реализовать намерение другим способом. Намерение может быть отброшено только при осуществлении определенного ментального усилия в случаях, если агент пришел к выводу, что реализовать намерение невозможно (не удается сформировать план, ведущий к достижению намерения) или оно уже неактуально для агента.
- Намерения влияют на рассуждения о будущем и, соответственно, планы — если агент выработал намерение, то он может строить планы на будущее с предположением, что это намерение реализовано.

Определение 4 *В терминах предложенной нами модели BDI-агент представляется набором* $ag = (S, A, env, see, I_B, bel, brf, I_D, des, drf, plan, prf)$, *где*

- S *есть непустое конечное множество состояний внешней среды;*
- A *есть непустое конечное множество действий агента;*
- $env : S \times A \to 2^S$ *есть функция* поведения *внешней среды, сопоставляющая текущему состоянию внешней среды и выбранному агентом действию* множество *возможных следующих состояний внешней среды;*
- $see \subseteq S \times S$ *есть корректное восприятие агентом состояний внешней среды, задающее множество* P *классов эквивалентности на* S*;*
- $I_B = I_{bel} \times Bel(S, A)$ *есть множество* представлений *агента;*
- $bel \in I_B$ *есть текущие представления агента;*
- $brf : I_B \times A \times P \to I_B$ *есть* функция обновления представлений*;*
- $I_D = I_{des} \times 2^{Goal(P,S,A)}$ *есть множество* желаний *агента;*
- $des \in I_D$ *есть множество текущих желаний агента;*
- $drf : I_D \times P \to I_D$ *есть* функция обновления желаний*;*
- $plan = (P, A, I_{pln}, \sigma_{pln}, i_{pln,0}$ *есть текущий план агента;*
- $prf : I_B \times I_D \times 2^{Plan} \times P \to 2^{Plan}$ *есть* функция обновления плана.

Глава 3

Модель мультиагентной системы

В данной главе предлагается расширение модели интеллектуального агента, рассмотренной в главе 2, для случая мультиагентной системы. Расширенная модель отражает следующие дополнительные свойства мультиагентной системы.

Коммуникация агентов. Можно выделить два основных вида коммуникации: оперативная коммуникация, используемая агентами для координации своих действий в текущий момент, и высокоуровневая коммуникация, используемая агентами для обмена информацией.

Расширенные представления. В случае мультиагентной системы агенты могут обладать представлениями не только о внешней среде и ее реакции на их действия, но и о возможной реакции внешней среды на действия других агентов, а также о возможных действиях других агентов.

Коалиции агентов. Для достижения общих и личных целей агенты объединяются в *коалиции* и действуют сообща, объединяя свои знания и возможности.

Определение 5 *Мультиагентная система представляется тройкой* $MAS = (S, AG, env)$, *где*

- S *есть конечное множество состояний внешней среды;*
- $AG = \{ag_1, \ldots, ag_n\}$ *есть конечное множество агентов, каждый из которых представлен расширенной моделью интеллектуального агента;*
- $env : S \times A_{ag_1} \times \ldots \times A_{ag_n} \rightarrow 2^S$ *есть функция, описывающая возможную реакцию внешней среды на действия* всех *агентов системы. Множество*

всех возможных совместных действий системы обозначим $ACS = A_{ag_1} \times \ldots \times A_{ag_n}$.

3.1. Коммуникация агентов

Можно выделить два основных вида коммуникации, используемой агентами.

Оперативная коммуникация. Этот вид коммуникации используется агентами для координации своих действий в текущий момент. В рамках оперативной коммуникации агентам необходимо обмениваться относительно небольшими объемами информации.

Высокоуровневая коммуникация. В этом случае агенты обмениваются более сложной информацией, которая может влиять на ментальное состояние агентов.

3.1.1. Оперативная коммуникация

Так как в рамках оперативной коммуникации агентам требуется максимально быстро обменяться относительно небольшим количеством информации, для моделирования этого вида коммуникации лучше всего подойдет модель *сигналов*.

Для каждого из агентов ag_i системы определено конечное множество сигналов Sig_{ag_i}, а также функция $Send_{ag_i} : P_{ag_i} \times AG \to Sig_{ag_i}$, описывающая какие сигналы агент ag_i пошлет каждому из других агентов в текущей ситуации. Заметим, что каждый из агентов может послать каждому другому агенту не более одного сигнала. Для моделирования ситуации, когда агент не посылает сигнала другому агенту, введем выделенный фиктивный сигнал $sg^{\varnothing} \in Sig_{ag_i}$.

Поведение функции $Send_{ag_i}$ изменяется в процессе взаимодействия с внешней средой, а посланные и полученные сигналы влияют на принятие другими

агентами решения о действии. Следовательно, функцию $Send_{ag_i}$ удобно моделировать в контексте текущего плана агента. В этом случае план агента представляется как взаимодействующий конечный автомат $plan_{ag_i} = (P_{ag_i}, A_{ag_i}, Sig_{ag_1} \times \ldots \times Sig_{ag_n}, Sig_{ag_i}, I_{pln}, send_{pln}, \sigma_{pln}, i_{pln,0})$, где

- P_{ag_i} есть входной алфавит автомата, совпадающий со множеством возможных восприятий агентом ag_i состояний внешней среды;
- A_{ag_i} есть выходной алфавит автомата, совпадающий с множеством действий агента ag_i;
- $Sig_{ag_1} \times \ldots \times Sig_{ag_n}$ есть множество входных сигналов автомата, совпадающее с декартовым произведением множеств сигналов всех агентов системы;
- Sig_{ag_i} есть множество выходных сигналов автомата, совпадающее с множеством сигналов агента ag_i;
- I_{pln} есть множество внутренних состояний автомата;
- $send_{pln} : P_{ag_i} \times I_{pln} \times AG \rightarrow Sig_{ag_i}$ есть функция посылки сигналов, определяющая по текущим восприятию внешней среды и состоянию автомата сигнал, который будет послан каждому из агентов системы;
- $\sigma_{pln} \subseteq P_{ag_i} \times Sig_{ag_1} \times \ldots \times Sig_{ag_n} \times I_{pln} \times I_{pln} \times A$ есть отношение переходов, определяющее по восприятию p текущего состояния внешней среды, набору полученных сигналов $(sg_1, \ldots, sg_n) \in Sig_{ag_1} \times \ldots \times Sig_{ag_n}$ и текущему внутреннему состоянию плана i_{pln} следующее состояние плана i'_{pln} и действие, которое следует выполнить агенту;
- $i_{pln,0} \in I_{pln}$ есть начальное состояние автомата.

Таким образом, решение о выборе действия и смене состояния автомата происходит в два этапа.

1. Определение и рассылка сигналов — на этом этапе каждый из агентов системы определяет по своему восприятию текущего состояния внешней среды и текущему состоянию своего плана, какие сигналы он должен послать остальным агентам и рассылает их.

2. Выбор действия — на этом этапе, основываясь на восприятии текущего состояния внешней среды, текущем состоянии плана и полученным от других агентов сигналах, агент выбирает свое действие и следующее состояние для своего плана.

Предложенная модель коммуникации достаточно проста в реализации и позволяет обеспечить быстрое принятие каждым из агентов системы решения о текущем действии. Заметим, что в предложенной модели агент может отправить сигнал сам себе, в чем, как правило, не возникает необходимости. Это возможность сохранена с целью упрощения описания.

3.1.2. Высокоуровневая коммуникация

Модель сигналов хорошо подходит для организации оперативной коммуникации, но она плохо применима для организации коммуникации высокоуровневой, так как позволяет обмениваться только относительно простой информацией. В большинстве существующих работ для моделирования коммуникации используется *теория речевого действия* [29, 128, 161], рассматривающая коммуникацию агентов как разновидность их действий. Аналогичный подход можно применить и для моделирования высокоуровневой коммуникации в нашем случае.

Определим для каждого из агентов системы отношение *восприятия действий* $see_{ACS} \subseteq S \times ACS \times ACS$, определяющее доступную агенту информацию

о совершаемых системой действиях. Восприятие действий является *корректным* если для любого $s \in S$ отношение $see_{ACS}(s) = \{(acs, acs') \in ACS \times ACS \mid (s, acs, acs') \in see_{ACS}\}$ является отношением эквивалентности. Далее будем рассматривать только корректные восприятия.

Определение 6 *Отношения восприятия* $see \subseteq S \times S$ *и восприятия действий* $see_{ACS} \subseteq S \times ACS \times ACS$ *совместно задают отношение эквивалентности* see_F *на множестве* $S \times ACS$ *следующим образом:*

$$see_F \triangleq \{(s, acs, s', acs') \in S \times ACS \times S \times ACS \mid s' \in see(s) \text{ and } \exists s'' \in see(s) : (acs, acs') \in see_{ACS}(s'')\}. \quad (3.1)$$

При этом отношение эквивалентности see_F *задает множество классов эквивалентности* P_F *на множестве* $S \times ACS$*, следовательно, его можно рассматривать как функцию* $see_F : S \times ACS \to P_F$*. Функция* see_F *назовем функцией* полного восприятия*, а множество* P_F *— множеством полных восприятий.*

Кроме того, расширим функцию обновления внутреннего состояния агента ag_i, включив в набор параметров общее действие, совершенное системой: $refine_{ag_i} : I \times S \times ACS \cup \{\varnothing\} \to I$. В том случае, когда система только начинает взаимодействовать с внешней средой, вместо общего действия передается $\varnothing$, так как агенты еще не совершали никаких действий.

Дальнейшая детализация и уточнение модели высокоуровневой коммуникации возможна, однако в контексте данной работы в ней нет необходимости.

3.2. Расширенные представления

Для моделирования представлений агентов о возможных последствиях действий других агентов отношение представлений необходимо расширить, включив в него действия *всех* агентов системы: $bel_{ag} \subseteq S \times ACS \times S$. Кроме того, представления агента о возможных действиях других агентов, *социальные*

представления, описываются отношением $sbel_{ag} \subseteq S \times ACS$, определяющим возможные, с точки зрения данного агента, действия системы для каждого из состояний внешней среды. Множество всех возможных социальных представлений обозначим $SBel(S, ACS) \triangleq 2^{S \times ACS}$.

Кроме того, необходимо изменить сигнатуру функции обновления представлений агента таким образом, чтобы она учитывала не только действия, совершенные данным агентом, но и действия других агентов системы, а также обновляла социальные представления агента:

$$\begin{aligned} brf_{ag} \ : \ & I_{ag,bel} \times Bel(S, ACS) \times SBel(S, AG) \times 2^{ACS} \times 2^{S} \\ & \rightarrow I_{ag,bel} \times Bel(S, ACS) \times SBel(S, AG). \end{aligned} \tag{3.2}$$

Таким образом, изменение представлений агента определяется на основании текущего состояния представлений $i_{ag,bel} \in I_{ag,bel}$, текущих представлений $bel_{ag} \in Bel(S, ACS)$ и социальных представлениях $sbel_{ag} \in SBel(S, AG)$, а также доступной агенту информации о совершенных действиях $p_{ACS} \subseteq ACS$ и текущем состоянии внешней среды $p_S \subseteq S$.

Расширение представлений агента требует и расширения его желаний. В случае мультиагентной системы желания агента представляются множеством функций-критериев вида $goal_{ag} \ : \ (2^S \times Bel(S, ACS) \times SBel(S, ACS))^* \rightarrow \{completed, continue, failure\}$. Множество всех таких функций обозначим как $Goal(S, ACS)$. Функция обновления желаний в этом случае примет вид $drf_{ag} : I_{ag,des} \times 2^{Goal(S,ACS)} \times 2^S \rightarrow I_{ag,des} \times 2^{Goal(S,ACS)}$.

3.3. Коалиции

Ключевым понятием при моделировании поведения мультиагентной системы является понятие *коалиции*.

Определение 7 *Коалиция $C \subseteq AG$ есть некоторое подмножество агентов*

системы, действующих совместно для достижения личных и общих целей.

Агенты, входящие в коалицию C, действуют совместно, доверяют друг другу, обмениваются информацией и координируют свои действия. При этом они не доверяют агентам вне коалиции C и не могут никак повлиять на поведение этих агентов. Коалицию агентов, не вошедших в C, назовем коалицией-антагонистом и обозначим $\overline{C} \triangleq AG \setminus C$.

Коалиция агентов $C = \{ag_1, \ldots, ag_{n_C}\} \subseteq AG$ формально может рассматриваться как единый интеллектуальный агент $ag^C = (S^C, A^C, env^C, I^C, refine^C, action^C)$, построенный следующим образом.

- $S^C \triangleq S \times (ACS \cup \{\varnothing\})$ — множество состояний внешней среды для коалиции есть декартово произведение множества состояний внешней среды системы в целом и множества действий системы или пустым множеством (для идентификации начальных состояний).

- $A^C \triangleq A_{ag_1} \times \ldots \times A_{ag_{n_C}}$ — множество действий коалиции есть декартово произведение множеств действий агентов коалиции.

- $env^C = \{(s^C, a^C, s'^C) \in S^C \times A^C \times S^C \mid \exists a^{\overline{C}} \in A^{\overline{C}} : s'^C|_{ACS} = a^C + a^{\overline{C}} \; and \; (s^C|_S, s'^C|_{ACS}, s'^C|_S) \in env\}$ — внешняя среда коалиции, по сравнению с внешней средой системы в целом, обладает большим недетерминизмом за счет действий агентов, не входящих в коалицию. Кроме того, часть состояния внешней среды коалиции, соответствующая действиям системы, должна однозначно определяться действиями.

- $I^C \triangleq I_{ag_1} \times \ldots \times I_{ag_{n_C}}$ — множество внутренних состояний коалиции есть декартово произведение множеств внутренних состояний агентов коалиции.

- $refine^C(i^C, s^C) \triangleq (refine_{ag_1}(i^C|_{I_{ag_1}}, s^C|_S, s^C|_{ACS}), \ldots, refine_{ag_{n_C}}(i^C|_{I_{ag_{n_C}}},$

$s^C|_S, s^C|_{ACS}))$ — функция обновления состояния коалиции есть вектор-функция, составленная из функций обновления состояния отдельных агентов.

- $action^C(i^C) \triangleq (action_{ag_1}(i^C|_{I_{ag_1}}), \ldots, action_{ag_{n_C}}(i^C|_{I_{ag_{n_C}}}))$ — функция принятия решения коалиции есть вектор-функция, составленная из функций обновления состояния отдельных агентов.

Кроме того, для коалиции-агента можно определить восприятие внешней среды и ментальное состояние, комбинируя соответствующим образом восприятия и ментальные состояния входящих в коалицию агентов.

3.3.1. Восприятие коалиции-агента

Обозначим пересечение восприятий состояния внешней среды агентов коалиции как $see^C_S \triangleq \bigcap\limits_{ag \in C} see_{ag} \subseteq S \times S$. В этом случае восприятие коалиции-агента $see^C \subseteq S^C \times S^C$ формируется следующим образом:

$$\begin{aligned} see^C \triangleq\ & \{(s^C, s'^C) \in S^C \times S^C \mid (s^C|_S, s'^C|_S) \in see^C_S \ and \\ & (s^C|_{ACS} = s'^C|_{ACS} \ or \ (s^C|_{ACS}, s'^C|_{ACS}) \in \bigcap_{ag \in C} \bigcup_{s'' \in see^C_S(s)} see_{ag,ACS}(s''))\}. \end{aligned} \tag{3.3}$$

Таким образом, два состояния s^C и s'^C воспринимаются коалицией-агентом одинаково, если части состояния, соответствующие состоянию внешней среды S, воспринимаются одинаково всеми агентами коалиции ($(s^C|_S, s'^C|_S) \in see^C_S$), а части состояния, соответствующие действиям системы, либо совпадают[1], либо воспринимаются одинаково всеми агентами коалиции относительно хотя бы одного состояния внешней среды s'', воспринимаемом одинаково с s всеми агентами коалиции ($(s^C|_{ACS}, s'^C|_{ACS}) \in \bigcap\limits_{ag \in C} \bigcup\limits_{s'' \in see^C_S(s)} see_{ag,ACS}(s'')$).

[1]Это дополнительной условие необходимо для обработки ситуации $s^C|_{ACS} = s'^C|_{ACS} = \varnothing$.

Заметим, что, так как все отношения see_{ag} и $see_{ag,ACS}(s)$ являются отношениями эквивалентности, то и их пересечение, объединение и комбинация являются отношениями эквивалентности. Таким образом, отношение восприятия коалиции see^C является отношением эквивалентности и задает фактор-множество на множестве S^C, которое назовем *множеством восприятий коалиции* $P^C \triangleq S^C|_{see^C}$. При этом для любого агента коалиции $ag \in C$ и для любого класса эквивалентности $p^C \in P^C$ существует единственный класс эквивалентности $p_{ag} \in P_{ag}$ такой, что $p^C|_S \subseteq p_{ag}$. Этот класс обозначим как $p^C|_{ag}$.

3.3.2. Представления и желания коалиции-агента

Текущие представления коалиции-агента формируются на основе представлений и социальных представлений, входящих в коалицию агентов следующим образом:

$$\begin{aligned} bel^C \triangleq \Big\{ & (s^C, a^C, s'^C) \in S^C \times A^C \times S^C \mid (s^C|_S, s'^C|_{ACS}, s'^C|_S) \in \bigcup_{ag \in C} bel_{ag} \\ & and\ \forall ag' \in C : s'^C|_{A_{ag'}} = a^C|_{A_{ag'}} \\ & and\ \forall ag'' \in AG \setminus C : s'^C|_{A_{ag''}} \in \bigcup_{ag \in C} sbel_{ag}(s^C|_S)|_{A_{ag''}} \Big\}. \end{aligned} \tag{3.4}$$

Таким образом, с точки зрения коалиции-агента возможны такие изменения состояний внешней среды, которые возможны по представлениям хотя бы одного из агентов коалиции, с учетом возможных действий других агентов по социальным представлениям агентов коалиции.

Заметим, что хотя коалиция и не обладает явно выделенными социальными представлениями, тем не менее социальные представления входящих в коалицию агентов отражены в представлениях коалиции о поведении внешней среды. Для того, чтобы разделить представления о поведении внешней среды и соци-

альные представления введем следующие обозначения:

$$\begin{aligned} env(bel^C) &\triangleq \{(s,a,s') \in S \times ACS \times S \mid \exists (s^C, a^C, s'^C) \in bel^C : \\ &\quad s = s^C|_S, a = s'^C|_{ACS}, s' = s'^C|_S\}, \qquad (3.5) \\ soc(bel^C) &\triangleq \{(s,a) \in S \times ACS \mid \exists (s^C, a^C, s'^C) \in bel^C : \\ &\quad s = s^C|_S, a = s'^C|_{ACS}\}. \qquad (3.6) \end{aligned}$$

Множество текущих желаний коалиции-агента des^C формируется как объединение всех желаний всех агентов коалиции, при этом каждой цели одного из агентов $goal_{ag} : (P_{ag}, S, ACS)^* \to \{completed, continue, failure\}$ соответствует цель коалиции $goal^C : (P^C, S^C, A^C)^* \to \{completed, continue, failure\}$, определяемая следующим образом:

$$goal^C(\{p_i^C, bel_i^C\}_{i\in\{1,\ldots,m\}}) = goal_{ag}(\{p_i^C|_S, env(bel_i^C), soc(bel_i^C)\}_{i\in\{1,\ldots,m\}}). \quad (3.7)$$

3.3.3. План коалиции-агента

План коалиции $plan^C = (P^C, A^C, I_{pln}^C, \sigma_{pln}^C, i_{pln,0}^C, F_{pln}^C)$ является комбинацией планов агентов, входящих в коалицию. Входной алфавит плана есть множество восприятий коалиции P^C, выходной алфавит является множеством действий коалиции A^C, множество внутренних состояний плана является декартовым произведением внутренних состояний планов агентов $I_{pln}^C = \underset{ag\in C}{\times} I_{ag,pln}$. Отношение переходов плана коалиции имеет вид $\sigma_{pln}^C \subseteq P^C \times I_{pln}^C \times I_{pln}^C \times A^C$ и состоит из таких четверок $(p^C, i_{pln}^C, i_{pln}'^C, a^C)$, что для любого агента коалиции $ag \in C$ для любого набора сигналов всех агентов системы $sig = (sig_{ag_1}, \ldots, sig_{ag_n}) \in Sig_{ag_1} \times \ldots \times Sig_{ag_n}$, если для любого агента коалиции $ag' \in C$ отправленный им сигнал определяется планом $sig|_{Sig_{ag'}} = send_{ag',pln}(p^C|_{ag'}, i_{pln}^C|_{I_{ag',pln}}, ag)$, то переход состояния и выбор действия агента ag определен его планом: $(p^C|_{ag}, sig, i_{pln}^C|_{I_{ag,pln}}, i_{pln}'^C|_{I_{ag,pln}}, a^C|_{A_{ag}}) \in \sigma_{ag,pln}$. Таким образом, в план коалиции попадают только те правила, которые не зависят от сигналов агентов вне коалиции. При этом

начальное состояние плана коалиции $i^C_{0,pln}$ формируется как вектор начальных планов агентов коалиции ($\forall ag \in C : i^C_{0,pln}|_{I_{ag,pln}} = i_{ag,0,pln}$).

3.3.4. Сужение коалиции

В случае, если коалиция D является подмножеством коалиции C ($D \subseteq C$), можно построить *сужение* описания внешней среды, текущего состояния и плана коалиции.

Сужением описания внешней среды для коалиции C ($env^C \subseteq S^C \times A^C \times S^C$) на описание внешней среды для коалиции D ($env^D \subseteq S^D \times A^D \times S^D$) назовем отношение $env^C|_D$, построенное следующим образом (заметим, что $S^C = S^D = S \times ACS \cup \{\varnothing\}$):

$$\begin{aligned} env^C|_D \triangleq\ & \{(s^D, a^D, s'^D) \in S^D \times A^D \times S^D \mid \exists a^{C\setminus D} \in A^{C\setminus D} \cup \{\varnothing\} : \\ & (s^D, a^D + a^{C\setminus D}, s'^D) \in env^C\}. \end{aligned} \tag{3.8}$$

Таким образом, описание $env^C|_D$ отличается от описания env^C суженным множеством действий (A^D вместо A^C) и расширенным за счет исключенных из рассмотрения действий недетерминизмом.

Сужением состояния коалиции C ($i^C \in I^C$) на состояние коалиции D ($i^D \in I^D$) назовем состояние $i^C|_D$, построенное как проекция состояния i^C на множество I^D:

$$i^C|_D \triangleq i^C|_{I^D}. \tag{3.9}$$

Сужением плана коалиции C ($plan^C = (P^C, A^C, I^C_{pln}, \sigma^C_{pln}, i^C_{pln,0}, F^C_{pln})$) на план коалиции D назовем план $plan^C|_D$, построенный, как показано в подразделе 3.3.3, на основе планов агентов из коалиции D, использовавшихся при построении $plan^C$.

Глава 4

Логика спецификации интеллектуального агента

В данной главе приводится обзор современных логик для спецификации свойств систем, изменяющихся со временем и обладающих *ментальным* состоянием, а также предлагается логика спецификации свойств модели интеллектуального агента (см. главу 2).

4.1. Спецификация поведения системы

Для формальной спецификации свойств системы часто используется некоторый логический язык, интерпретация формул которого производится на некоторой формальной *модели* системы (обзор способов построения формальных моделей приведен в разделе 2.1). Плюсом такого подхода является его математическая строгость и отсутствие двусмысленности, основным минусом же является то, что специфицируется и верифицируется именно модель, а не сама система.

Наиболее простым логическим языком является пропозициональная логика, именуемая также *логикой высказываний* [11, 12]. В том или ином виде логика высказываний входит практически в любой формальный логический язык. Формальное описание логики высказываний в начале этого раздела приведено для того, чтобы ознакомить читателя с используемой нотацией на примере уже известного ему логического формализма.

Синтаксис логики высказывания первого порядка задается с помощью алфавита пропозициональных символов $Prop$ и следующей грамматики:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi, \tag{4.1}$$

где $p \in Prop$. Используя базовые операторы отрицания $\neg$ и конъюнкции $\wedge$ можно выразить целый ряд дополнительных операторов.

- Дизъюнкция: $\phi \vee \psi \triangleq \neg(\neg\phi \wedge \neg\psi)$.
- Импликация: $\phi \Rightarrow \psi \triangleq \neg\phi \vee \psi$.
- Равносильность: $\phi \Leftrightarrow \psi \triangleq (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$.
- Константы истинности и лжи: $true \triangleq p \vee \neg p$, $false \triangleq \neg true$.

Семантика логики высказываний задается относительно множества *состояний* S и *оценочной функции* $\pi : S \to 2^{Prop}$, сопоставляющей каждому состоянию $s \in S$ множество пропозициональных символов, которые истинны для данного состояния. Пары $M = \langle S, \pi \rangle$ называют *моделью* (в некоторых работах эту пару называют *интерпретацией*). Формальной описание семантики логики дается с помощью *отношения выполнимости* "$\models$" между парой (M, s) (где M есть некоторая модель, а $s \in S$ есть некоторое состояние) и некоторой формулой логики ϕ. Правила, задающие отношение выполнимости для логики высказываний, выглядят следующим образом.

1. $M, s \models p$ тогда и только тогда, когда $p \in \pi(s)$.
2. $M, s \models \neg\phi$ тогда и только тогда, когда $M, s \not\models \phi$.
3. $M, s \models \phi \wedge \psi$ тогда и только тогда, когда $M, s \models \phi$ и $M, s \models \psi$.

Задача верификации системы, специфицированной с помощью формальной логики, сводится к задаче *проверки модели* (model-checking). Для логики высказываний эта задача формулируется следующим образом: для данных модели $M = \langle S, \pi \rangle$ и формулы логики высказываний ϕ найти такое множество состояний $[\phi] \subseteq S$, что для любого состояния этого множества $s \in [\phi]$ выполнено $M, s \models \phi$.

Для логики высказываний задача проверки модели, при условии конечности множества S, разрешима за время, не превосходящее $O(|\phi| \cdot |S|)$, где $|\phi|$ есть длина формулы, а $|S|$ есть мощность множества S. Однако, несложно заметить, что семантика классической логики высказываний плохо приспособлена для спецификации динамических систем, особенно в том случае, если выполнение этих систем не обязательно останавливается. Для решения этой проблемы было предложено несколько расширений логики высказываний, обзор которых приведен в данном разделе.

4.1.1. Динамическая логика

Одним из подходов к спецификации динамических систем является *динамическая логика высказываний* (Propositional Dynamic Logic, PDL) [68, 105, 147]. Далее мы рассмотрим динамическую логику высказываний для *регулярных программ* (см. ниже), так как она является наиболее распространенным вариантом динамической логики высказываний.

Язык динамической логики высказываний включает два алфавита: алфавит пропозициональных символов $Prop$ и алфавит символов действий $Actions$. Синтаксис этого языка задаётся следующей грамматикой:

$$\alpha \ ::= \ a \mid \alpha;\alpha \mid \alpha + \alpha \mid \phi?\alpha \mid \alpha*; \tag{4.2}$$

$$\phi \ ::= \ p \mid \neg\phi \mid \phi \wedge \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi, \tag{4.3}$$

где $a \in Actions$, а $p \in Prop$. Таким образом, язык динамической логики высказываний включает в себя язык регулярных выражений, позволяющий описать программы, строящиеся из элементарных действий по правилам, заданным регулярным выражением (именно поэтому такие программы и называются *регулярными*).

Семантика динамической логики высказываний задается относительно модели $M = \langle S, A, R, \pi, \theta\rangle$, где

- S есть непустое конечное множество состояний;
- A есть непустое конечное множество действий;
- $R \subseteq S \times A \times S$ есть *отношение переходов*, определяющее возможные изменения состояний при выполнении тех или иных действий;
- $\pi : S \to 2^{Prop}$ есть оценочная функция для пропозициональных символов;
- $\theta : Actions \to A$ есть интерпретация действий.

Семантика динамической логики высказываний описывается с помощью отношения выполнимости "$\models$" между парой (M, s) (где M есть некоторая модель, а $s \in S$ есть некоторое состояние) и некоторой формулой логики ϕ. Правила, задающие отношение выполнимости для динамической логики высказываний, выглядят следующим образом.

1. $M, s \models p$ тогда и только тогда, когда $p \in \pi(s)$.
2. $M, s \models \neg\phi$ тогда и только тогда, когда $M, s \not\models \phi$.
3. $M, s \models \phi \wedge \psi$ тогда и только тогда, когда $M, s \models \phi$ и $M, s \models \psi$.
4. $M, s \models [\alpha]\phi$ тогда и только тогда, когда для любого $s' \in S$, если $(s, s') \in R_\alpha$, то $M, s \models \phi$.
5. $M, s \models \langle\alpha\rangle\phi$ тогда и только тогда, когда существует такое $s' \in S$, что $(s, s') \in R_\alpha$ и $M, s \models \phi$.

где $R_\alpha \subseteq S \times S$ есть отношение, порожденное регулярной программой α на основе отношения R следующим образом.

1. $(s, s') \in R_a$ тогда и только тогда, когда $(s, \theta(a), s') \in R$.
2. $(s, s') \in R_{\alpha;\beta}$ тогда и только тогда, когда существует такое $s'' \in S$, что $(s, s'') \in R_\alpha$ и $(s'', s') \in R_\beta$.

3. $(s, s') \in R_{\alpha+\beta}$ тогда и только тогда, когда существует $(s, s') \in R_\alpha$ или $(s, s') \in R_\beta$.

4. $(s, s') \in R_{\phi?\alpha}$ тогда и только тогда, когда $M, s \vDash \phi$ и $(s, s') \in R_\alpha$.

5. $(s, s') \in R_{\alpha*}$ тогда и только тогда, когда существует такое $n \in \mathbb{N}$ и такая последовательность $(s_0, \ldots, s_n) \in S^{n+1}$, что $s = s_0$, $s' = s_n$ и для любого $0 \leq i < n$ выполнено $(s_i, s_i + 1) \in R_\alpha$.

Несложно заметить схожесть динамической логики высказываний с алгебрами процессов, рассмотренными в разделе 2.1.2. Вопрос рассмотрения логик как алгебраических систем подробно изучен в работе [27].

4.1.2. Логика линейного времени

Логика линейного времени (Linear Temporal Logic, LTL) является представителем большого класса темпоральных логик [56], который широко применяется для формальной спецификации программ и вычислительных микросхем. Среди инструментов, использующих LTL или одну из её модификаций можно выделить: DB-Rover [53, 54], STeP [167], SPIN [80], SMV [122], NuSMV [132].

Язык логики линейного времени формируется из алфавита пропозициональных символов $Prop$ следующей грамматикой:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc\phi \mid \phi \,\mathcal{U}\, \phi, \tag{4.4}$$

где $p \in Prop$. Таким образом, логика линейного времени расширяет логику высказываний следующими дополнительными операторами:

- $\bigcirc\phi$, интуитивно интерпретируемым как “в следующий момент ϕ”;

- $\phi \;\mathcal{U}\, \psi$, интуитивно интерпретируемым как “когда-нибудь ψ, а до этого момента ϕ”;

Кроме того, часто вводятся два дополнительных оператора, выражаемых через основные следующим образом:

- $\mathcal{F}\,\phi \triangleq true\,\mathcal{U}\,\phi$, интуитивно "когда-нибудь ϕ";
- $G\,\phi \triangleq \neg\mathcal{F}\,\neg\phi$, интуитивно "всегда ϕ";

Моделью для интерпретации формул логики линейного времени является набор $M = \langle S, \tau, \pi\rangle$, где

- S есть непустое конечное множество состояний;
- $\tau : \mathbb{N} \to S$ есть упорядочивание состояний, описывающее бесконечную последовательность. Это упорядочивание и задает модель линейного времени;
- $\pi : S \to 2^{Prop}$ есть оценочная функция для пропозициональных символов.

Семантика логики линейного времени описывается с помощью отношения выполнимости "$\vDash$" между парой (M, t) (где M есть некоторая модель, а $t \in \mathbb{N}$ есть некоторый момент времени) и некоторой формулой логики ϕ. Правила, задающие отношение выполнимости для динамической логики высказываний, выглядят следующим образом.

1. $M, t \vDash p$ тогда и только тогда, когда $p \in \pi(\tau(t))$.
2. $M, t \vDash \neg\phi$ тогда и только тогда, когда $M, t \nvDash \phi$.
3. $M, t \vDash \phi \wedge \psi$ тогда и только тогда, когда $M, t \vDash \phi$ и $M, t \vDash \psi$.
4. $M, t \vDash \bigcirc\phi$ тогда и только тогда, когда $M, t + 1 \vDash \phi$.
5. $M, t \vDash \phi\,\mathcal{U}\,\psi$ тогда и только тогда, когда существует такое $t' > t$, что $M, t' \vDash \psi$ и для всех $t \leq t'' < t'$ выполнено $M, t'' \vDash \phi$.

Логика линейного времени хорошо подходит для спецификации программ с детерминированным поведением. Однако, логика линейного времени обладает существенным недостатком — задача верификации для нее принадлежит к классу $EXPTIME$-полных задач.

Стоит отметить, что описанный нами вариант логики линейного времени является наиболее простым в своем классе. Многими исследователями были предложены различные модификации этой логики для разных целей. Одной из наиболее гибких логик этого класса на данный момент является логика $EAGLE$ [158], содержащая два базовых оператора "в следующий момент"($\bigcirc$) и "в предыдущий момент" ($\odot$), а также предоставляющая средства определения новых операторов, семантика которых задается с помощью рекурсивных уравнений, например, $G\phi \equiv \phi \wedge \bigcirc G\phi$. Эта логика активно применяется для автоматизации тестирования [148, 168] и именно она была использована, в частности, для тестирования программного обеспечения планетоходов $NASA$ [50].

4.1.3. Логика ветвящегося времени

Логика линейного времени хорошо подходит для спецификации детерминированных систем, но большинство современных систем либо сами являются недетерминированными, любо находятся в недетерминированной среде. Для спецификации таких систем используется *логика ветвящегося времени* (Computational Tree Logic, CTL). Модель времени для этой логики подразумевает ветвящееся в будущем время, что позволяет специфицировать недетерминированные системы.

Язык логики ветвящегося времени формируется из алфавита пропозициональных символов $Prop$ и задается следующей грамматикой:

$$\rho \ ::= \ \bigcirc\phi \mid \phi \,\mathcal{U}\, \phi; \tag{4.5}$$

$$\phi \ ::= \ p \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{A}\,\rho \mid \mathbf{E}\,\rho, \tag{4.6}$$

где $p \in Prop$. Формулы типа ρ называются *формулами пути* и интерпретируются относительно одного варианта работы системы, представленного последовательностью состояний, а формулы типа ϕ называются *формулами состояния* и интерпретируются относительно некоторого состояния системы. Для приведения формул пути к формулам состояния вводятся *кванторы путей*.

- На всех возможных путях, начинающихся в данном состоянии: $\mathbf{A}$.
- На одном из возможных путей, начинающемся в данном состоянии: $\mathbf{E}$.

Производные операторы G и $\mathcal{F}$ в случае логики ветвящегося времени определяются следующим образом:

- $\mathbf{A}\,\mathcal{F}\,\phi \triangleq \mathbf{A}\,true\,\mathcal{U}\,\phi$.
- $\mathbf{E}\,\mathcal{F}\,\phi \triangleq \mathbf{E}\,true\,\mathcal{U}\,\phi$.
- $\mathbf{A}\,G\,\phi \triangleq \neg\mathbf{E}\,\mathcal{F}\,\neg\phi$.
- $\mathbf{E}\,G\,\phi \triangleq \neg\mathbf{A}\,\mathcal{F}\,\neg\phi$.

Моделью для интерпретации формул логики ветвящегося времени является набор $M = \langle S, R, \pi \rangle$, где

- S есть непустое конечное множество состояний;
- $R \subseteq S \times S$ отношение переходов, описывающее, какие состояния системы могут следовать за какими;
- $\pi : S \to 2^{Prop}$ есть оценочная функция для пропозиций.

Допустимым путем исполнения системы M называется последовательность состояний $\{s_0, s_1, \ldots\} \in S^+$, такая что для любого $i \in \mathbb{N}$ выполнено $(s_i, s_{i+1}) \in R$. Множество всех допустимых путей исполнения системы M обозначим как $Paths_M$. Здесь и далее для операции с последовательностями мы будем использовать следующую нотацию:

- $\lambda[i]$ для идентификации i-го элемента последовательности;
- $\lambda[i, j]$ для идентификации участка последовательности, начинающегося с i-го элемента и имеющую длину j.

Семантика логики ветвящегося времени задается с помощью двух отношений: $\vDash_s$ и $\vDash_p$. Отношение $\vDash_s$ определено между парой (M, s) (где M есть некоторая модель, а $s \in S$ есть некоторое состояние) и формулой состояния, а отношение $\vDash_p$ определено между парой (M, λ) (где M есть некоторая модель, а $\lambda \in Paths_M$ есть некоторый допустимый путь для этой модели) и формулой пути. Формально эти отношения задаются следующими правилами.

1. $M, s \vDash_s p$ тогда и только тогда, когда $p \in \pi(s)$.
2. $M, s \vDash_s \neg\phi$ тогда и только тогда, когда $M, s \nvDash_s \phi$.
3. $M, s \vDash_s \phi \wedge \psi$ тогда и только тогда, когда $M, s \vDash_s \phi$ и $M, s \vDash_s \psi$.
4. $M, s \vDash_s \mathbf{A}\,\rho$ тогда и только тогда, когда для любого допустимого пути $\lambda \in Paths_M$, если $\lambda[0] = s$, то $M, \lambda \vDash_p \rho$.
5. $M, s \vDash_s \mathbf{E}\,\rho$ тогда и только тогда, когда существует допустимый путь $\lambda \in Paths_M$ такой, что $\lambda[0] = s$ и $M, \lambda \vDash_p \rho$.
6. $M, \lambda \vDash_p \bigcirc\phi$ тогда и только тогда, когда $M, \lambda[1] \vDash_s \phi$.
7. $M, \lambda \vDash_p \phi\,\mathcal{U}\,\psi$ тогда и только тогда, когда существует такое $i \in \mathbb{N}$, что $M, \lambda[i] \vDash_s \psi$ и для всех $0 \leq j < i$ выполнено $M, \lambda[j] \vDash_s \phi$.

Важным достоинством логики ветвящегося времени является то, что задача верификации для нее имеет линейную сложность [56, 150] относительно размера спецификации и размера модели. На первый взгляд может показаться странным тот факт, что задача верификации для логики линейного времени имеет

большую сложность, чем задача для более сложной логики ветвящегося времени. Причина упрощения верификации заключается в том, что в случае логики ветвящегося времени допустимы только простые формулы пути, например, формула $\bigcirc\phi \wedge G\psi$ является корректной формулой для логики линейного времени, тогда как формула $\mathbf{A} \bigcirc \phi \wedge G\psi$ является синтаксически некорректной с точки зрения логики ветвящегося времени. В результате, выразительные мощности этих двух логик несравнимы — некоторые свойства системы можно выразить только в логике линейного времени, а некоторые только в логике ветвящегося времени.

Расширенная логика ветвящегося времени CTL^* устраняет это несоответствие, так как её выразительная мощность строго больше обеих рассмотренных ранее логик. Синтаксис расширенной темпоральной логики задается следующей грамматикой:

$$\rho \ ::= \ \bigcirc\phi \mid \phi \,\mathcal{U}\, \phi \mid \neg\rho \mid \rho \wedge \rho; \tag{4.7}$$

$$\phi \ ::= \ p \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{A}\,\rho. \tag{4.8}$$

Таким образом, расширенная логика ветвящегося времени позволяет использовать логические операторы для комбинирования более сложных формул пути из более простых. Заметим, что в расширенном определении синтаксиса отсутствует квантор существования пути, так как в данном случае он может быть выражен с помощью отношения двойственности $\mathbf{E}\,\rho \triangleq \neg\mathbf{A}\,\rho$.

Семантика расширенной логики ветвящегося времени аналогична обычной, но включает два дополнительных семантических правила для формул пути.

8. $M, \lambda \vDash_p \neg\rho$ тогда и только тогда, когда $M, \lambda \nvDash_p \rho$.

9. $M, \lambda \vDash_p \rho \wedge \varrho$ тогда и только тогда, когда $M, \lambda \vDash_p \rho$ и $M, \lambda \vDash_p \varrho$.

Благодаря своей простоте, наглядности, и, что наиболее важно, благодаря линейности задачи верификации логика ветвящегося времени и различные

её варианты и расширения очень широко используются в современной индустрии. Среди инструментов, использующих логику ветвящегося времени можно выделить: SMV [122], KRONOS [169], SGM [81], UPPAALL [170], HyTech [72], PRISM [107, 135].

4.1.4. Логика альтернированного времени

Описанные выше подходы удобны в случае, если специфицируемая система может быть рассмотрена как единое целое, совершающее переходы состояний или действия, ведущие к этим переходам, однако в случае мультиагентных систем такое представление часто невозможно. Агенты такой системы могут как сотрудничать друг с другом для достижения общих целей, так и соперничать в достижении целей индивидуальных.

Серьезно ситуация в этой области изменилась всего несколько лет назад с появлением сразу двух работ: коалиционной логики Паули (Coalition Logic, CL [138, 139]) и логики альтернированного времени (Alternating-time Temporal Logic, ATL [25]). Предложенные авторами логики позволяли явно описывать возможности для агентов и групп агентов (*коалиций*) добиваться определенных целей вне зависимости от действий агентов вне коалиций и возможной реакции внешней среды.

Поскольку, как показано в [64], логика альтернированного времени включает в себя коалиционную логику, в данном разделе мы рассмотрим именно её. Язык логики альтернированного времени формируется из алфавитов пропозициональных символов $Prop$ и алфавитов символов *коалиций* $Coal$ следующей грамматикой:

$$\rho \ ::= \ \bigcirc\phi \mid G\phi \mid \phi \,\mathcal{U}\, \phi; \tag{4.9}$$

$$\phi \ ::= \ p \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle A \rangle\rangle \rho, \tag{4.10}$$

где $p \in Prop$, а $A \in Coal$. Как и в случае логики ветвящегося времени формулы

типа ρ описывают свойства путей (цепочек состояний), а формулы типа ϕ описывают свойства конкретных состояний. По сравнению с логикой ветвящегося времени, в логике альтернированного времени кванторы пути **A** и **E** заменяются квантором коалиции $\langle\langle A\rangle\rangle$. Интуитивная интерпретация формулы $\langle\langle A\rangle\rangle\rho$ звучит как "Коалиция агентов A может добиться того, что будет выполнено свойство ρ не зависимо от действий других агентов и реакции внешней среды".

Для логики альтернированного времени оператор "всегда" G входит в набор основных операторов, а оператор "когда-нибудь" $\mathcal{F}$ можно определить следующим образом: $\langle\langle A\rangle\rangle\mathcal{F}\,\phi \triangleq \langle\langle A\rangle\rangle true\,\mathcal{U}\,\phi$. Кроме того, можно определить производный квантор $[[A]]\rho$, интуитивно интерпретируемый как "Коалиция агентов A не может избежать того, что выполнится свойство ρ", следующим образом:

- $[[A]]\bigcirc\phi \triangleq \neg\langle\langle A\rangle\rangle\bigcirc\neg\phi$.
- $[[A]]\,G\,\phi \triangleq \neg\langle\langle A\rangle\rangle\,G\,\neg\phi$.

Заметим, в большинстве определений темпоральной логики отсутствует оператор двойственный оператору $\phi\,\mathcal{U}\,\psi$, поэтому в нашем определении квантор $[[A]]$ может быть применен только к унарным операторам. Двойственный $\phi\,\mathcal{U}\,\psi$ оператор существует и обычно обозначается как $\phi\,\mathcal{B}\,\psi$. Интуитивная интерпретация этого оператора звучит как: "Если когда-нибудь в будущем выполнится свойство ψ, то в некотором моменте до этого будет выполнено свойство ϕ". Однако, этот оператор вводится и используется достаточно редко, так как в нем практически никогда нет потребности.

Моделью для интерпретации формул логики альтернированного времени является набор $M = \langle S, AG, \{AC_{ag}\}_{ag\in AG}, \{\chi_{ag}\}_{ag\in AG}, R, \pi, \gamma\rangle$, где

- S есть непустое конечное множество состояний.
- $AG = \{ag_1, \ldots ag_n\}$ есть непустое конечное множество агентов.

- $\{AC_{ag}\}_{ag \in AG}$ есть набор конечных непустых множеств действий AC_{ag} для каждого агента $ag \in AG$.

- $\{\chi_{ag}\}_{ag \in AG}$ есть набор функций ограничения выбора действия $\chi_{ag} : S \to 2^{AC_{ag}}$ для каждого агента $ag \in AG$.

- $R \subseteq S \times AC_{ag_1} \times \ldots \times AC_{ag_n} \times S$ есть отношение переходов, описывающее возможные изменения состояния системы в зависимости от общего действия всех агентов.

- $\pi : S \to 2^{Prop}$ есть оценочная функция для пропозиций.

- $\gamma : Coal \to 2^{AG}$ есть интерпретация символов коалиций, сопоставляющая каждому символу коалиции некоторое подмножество множества AG (т.е. некоторый набор агентов).

Ключевым понятием в семантики логики альтернированного времени является понятие *стратегии*. Для некоторого агента $ag \in AG$ стратегия $f_{ag} : S^+ \to AC_{ag}$ есть отображение, сопоставляющее любой непустой конечной последовательности состояний выбор действия агента таким образом, что $f_{ag}(s_1, \ldots, s_k) \in \chi_{ag}(s_k)$ (т.е. выбор действия агента не противоречит текущему состоянию системы). Стратегией коалиции агентов $A \subseteq AG$ является набор стратегий $F_A = \{f_{ag}\}_{ag \in A}$ для каждого из агентов коалиции.

Последовательность $\lambda = (s_0, s_1, \ldots) \in S^+$ назовем *возможным результатом* стратегии F_A в состоянии $s \in S$ если выполнены следующие условия:

- $\lambda[0] = s$;

- для любого натурального $i \in \mathbb{N}$ существует такой вектор действий агентов $ac = (ac_{ag_1}, \ldots, ac_{ag_n})$, что $(\lambda[i], ac, \lambda[i+1]) \in R$ и для любого агента коалиции $ag \in A$ выбор действия определяется его стратегией: $ac_{ag} = f_{ag}(\lambda[0, i])$.

Множество всех возможных результатов стратегии F_A в состоянии $s \in S$ обозначим как $out(F_A, s) \subseteq S^+$.

Теперь мы можем описать семантику логики альтернированного времени с помощью двух отношений: $\vDash_s$ и $\vDash_p$. Отношение $\vDash_s$ определено между парой (M, s) (где M есть некоторая модель, а $s \in S$ есть некоторое состояние) и формулой состояния, а отношение $\vDash_p$ определено между парой (M, λ) (где M есть некоторая модель, а $\lambda \in S^+$ есть некоторая цепочка состояний) и формулой пути. Формально эти отношения задаются следующими правилами.

1. $M, s \vDash_s p$ тогда и только тогда, когда $p \in \pi(s)$.

2. $M, s \vDash_s \neg\phi$ тогда и только тогда, когда $M, s \nvDash_s \phi$.

3. $M, s \vDash_s \phi \wedge \psi$ тогда и только тогда, когда $M, s \vDash \phi$ и $M, s \vDash_s \psi$.

4. $M, s \vDash_s \langle\langle A \rangle\rangle \rho$ тогда и только тогда, когда для коалиции A существует такая стратегия F_A, что для любого её результата $\lambda \in out(F_A, s)$ выполнено $M, \lambda \vDash_p \rho$

5. $M, \lambda \vDash_p \bigcirc\phi$ тогда и только тогда, когда $M, \lambda[1] \vDash_s \phi$.

6. $M, \lambda \vDash_p G\phi$ тогда и только тогда, когда для всех $i \in \mathbb{N}$ выполнено $M, \lambda[i] \vDash_s \phi$.

7. $M, \lambda \vDash_p \phi \,\mathcal{U}\, \psi$ тогда и только тогда, когда существует такое $i \in \mathbb{N}$, что $M, \lambda[i] \vDash_s \psi$ и для всех $0 \leq j < i$ выполнено $M, \lambda[j] \vDash_s \phi$.

Так же как и в случае с логикой ветвящегося времени, логику альтернированного времени можно расширить до ATL^*, позволив конструировать более сложные формулы пути пользуясь грамматикой: $\rho ::= \bigcirc\phi \mid G\phi \mid \phi \,\mathcal{U}\, \phi \mid \rho \wedge \rho \mid \neg\rho$, а также добавив два дополнительных семантических правила.

8. $M, \lambda \vDash_p \neg\rho$ тогда и только тогда, когда $M, \lambda \nvDash_p \rho$.

9. $M, \lambda \vDash_p \rho \wedge \varrho$ тогда и только тогда, когда $M, \lambda \vDash_p \rho$ и $M, \lambda \vDash_p \varrho$.

Сразу же после своего появления, работы по коалиционной логике и логике альтернированного времени привлекли внимание научного сообщества разработчиков мультиагентных систем и получили развитие. Многими авторами ([140, 173–175]) было предложено использовать ATL в качестве эпистемологической логики (логики для представления знаний агента), а также для формализации концепции *роли* [131]. Интересное развитие получила и коалиционная логика в работах [15] и [16]. Интересным обобщением коалиционной логики и логики альтернированного времени является коалиционная логика пропозиционального контроля (Coalition Logic of Propositional Control, $CL - PC$) [177].

В работе [25] доказывается, что алгоритмическая сложность задачи верификации для логики альтернированного времени не превосходит полинома от размера спецификации и размера модели. Однако, работа [172] содержит критический анализ и утверждает, что хотя исходные оценки алгоритмической сложности верны, они сделаны для случая, когда модель задана полным перечислением. В реальных же инструментах верификации, таких SMV [122] (для CTL), $SPIN$ [80] (для LTL) и $MOCHA$ [127] (для ATL) используется более краткая нотация для записи моделей (например, $MOCHA$ использует язык Reactive Modules Language, RML [26]), и относительно размера описания модели в этой нотации задача верификация формул ATL является $EXPTIME$-полной (для коалиционной логики этот результат немного “лучше” — относительного краткой формы описания она является $PSPACE$-полной). Кроме того, в работе [88] показано, что в случае если количество агентов рассматривается как параметр проблемы, задача верификации для логики альтернированного времени становится NP-полной.

4.1.5. Логики реального времени

Темпоральные логики являются мощным инструментом для спецификации динамических систем, но некоторые свойства, связанные с описанием ограничений на время реакции, выразить в них достаточно сложно. Рассмотрим, например, такое свойство как "На любом пути всегда если произошло событие p, то не позднее чем через 3 шага произойдет событие q". Спецификация этого свойства средствами логики ветвящегося времени выглядит следующим образом: $\mathbf{A}\, G\, (p \Rightarrow \mathbf{A} \bigcirc (q \vee \mathbf{A} \bigcirc (q \vee \mathbf{A} \bigcirc q)))$. В тоже время свойства такого вида представляют наибольший интерес с точки зрения задачи спецификации систем реального времени.

Для логики ветвящегося времени было предложено множество различных расширений, позволяющих в более удобном виде выражать подобные свойства. Хороший обзор подобных расширений можно найти в работах [71] и [109], здесь мы приведем краткое описание основных результатов.

Подстрочный индекс

Наиболее простым и наглядным решением является введение ограниченного оператора $\phi\, \mathcal{U}_{\leq t} \psi$, предложенное в работах [150] и получившее название $RTCTL$. Интуитивная интерпретация этого оператора звучит как "не позднее чем через t шагов (когда-нибудь, если $t = \infty$) будет выполнено ψ, а до этого момента будет выполнено ϕ". Формальная семантика этого оператора может быть определена без расширения модели логики с помощью очевидной модификации соответствующего семантического правила:

7. $M, \lambda \vDash_p \phi\, \mathcal{U}_{\leq t} \psi$ тогда и только тогда, когда существует такое $0 \leq i < t$,[1] что $M, \lambda[i] \vDash_s \psi$ и для всех $0 \leq j < i$ выполнено $M, \lambda[j] \vDash_s \phi$.

[1] В базовом определении $i \in \mathbb{N}$ не ограничено.

Главным достоинством такого расширения является то, что оно не увеличивает алгоритмическую сложность задачи верификации [150]. В работе [109] приводится обобщение этого подхода $TCTL_s$, позволяющее использовать ограничения более сложного вида $\phi \,\mathcal{U}_{\,t\in I}\psi$ (где $I = \{[i_1, i_2], \ldots [i_{2n-1}, i_{2n}]\}$ есть набор отрезков натуральных чисел и $0 \leq i_1 < \ldots < i_{2n} \leq \infty$), а также расширяющее модель логики понятием *времени перехода* (в $RTCTL$ каждый переход занимает ровно одну единицу времени, тогда как в $TCTL_s$ это время может варьироваться). Показано, что для $TCTL_s$ верификации сохраняет полиномиальную сложность. Кроме того, показано, что выразительная мощность $TCTL_s$ не превосходит выразительной мощности CTL, однако формулы $TCTL_s$ могут быть значительно (экспоненциально) короче.

Для логики альтернированного времени в работе [108] было предложено аналогичное по синтаксису расширение $TATL_s$, задача верификации для которого принадлежит к классу $EXPTIME$-полных задач.

Переменные часов

Другим подходом к описанию ограничений на время является использование *переменных часов*, предложенное в работе [24]. Этот способ расширения был обобщен в работе [109] и получил название $TCTL_c$. Синтаксис логики $TCTL_c$ определяется следующей грамматикой:

$$\rho \;::=\; \bigcirc\phi \mid \phi \,\mathcal{U}\, \phi; \tag{4.11}$$

$$\phi \;::=\; p \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{A}\,\rho \mid \mathbf{E}\,\rho \mid x \;\underline{\mathbf{in}}\; \phi \mid x \sim t, \tag{4.12}$$

где $p \in Prop$ есть пропорциональный символ, $x \in Clocks$ есть переменная часов ($Clocks$ есть алфавит таких переменных), $t \in \mathbb{N}$, а $\sim$ есть одни из операторов сравнения: $<$, $\leq$, $=$, $\geq$ или $>$. Квантор $\underline{\mathbf{in}}$ используется для того, чтобы зафиксировать значение переменной часов равным 0 в текущий момент времени. В процессе интерпретации формул пути, значения всех переменных часов уве-

личиваются на каждом шаге, а ограничения $x \sim t$ используется для проверки того, что значения переменной находится в нужных пределах.

Для определения формальной семантики $TCTL_c$ введем оценочную функцию для переменных часов $v : Clocks \to \mathbb{N}$, а также опишем следующие операции с этой функцией:

$$(v + i)(x) \triangleq v(x) + i \tag{4.13}$$

$$v[y \leftarrow 0](x) \triangleq \begin{cases} v(x), & if\ x \neq y \\ 0, & if\ x = y \end{cases} \tag{4.14}$$

Для интерпретации формул $TCTL_c$ оба отношения выполнимости $\vDash_s$ и $\vDash_p$ необходимо расширить, включив в левую часть вместе с моделью и состоянием (путем) оценочную функцию для переменных часов $v : Clocks \to \mathbb{N}$. Семантические правила для логических связок и классических кванторов пути **A** и **E** останется неизменной, а правила для формул пути и новых типов формул состояния выглядят следующим образом.

6. $M, \lambda, v \vDash_p \bigcirc\phi$ тогда и только тогда, когда $M, \lambda[1], v + 1 \vDash_s \phi$.

7. $M, \lambda \vDash_p \phi\ \mathcal{U}\ \psi$ тогда и только тогда, когда существует такое $i \in \mathbb{N}$, что $M, \lambda[i], v + i \vDash_s \psi$ и для всех $0 \leq j < i$ выполнено $M, \lambda[j], v + j \vDash_s \phi$.

8. $M, s, v \vDash_s x \in \phi$ тогда и только тогда, когда $M, s, v[x \leftarrow 0] \vDash_s \phi$.

9. $M, s, v \vDash_s x \sim t$ тогда и только тогда, когда $v(x) \sim t$.

Основной проблемой такого подхода к описанию ограничений на время является то, что задача верификации принадлежит к классу $PSPACE$-полных задач. Аналогичное по синтаксису расширение логики альтернированного времени $TATL_c$ было предложено в работе [73]. Задачи верификации для $TATL_c$ принадлежит к классу $EXPTIME$-полных задач.

Непрерывное время

В изложенных ранее подходах рассматривался случай дискретного времени, однако в реальных системах время может являться величиной непрерывной. Предложенные выше модели с дискретным временем могут быть обобщены для случая непрерывного времени, однако не для всех этих обобщений задача верификации является разрешимой. Семантика логик с непрерывным временем может сильно отличаться от семантики логик со временем дискретным. Обзор подходов к моделированию и спецификации систем с непрерывным временем можно найти в работах [22, 23].

Использование ограниченного оператора $\phi \, \mathcal{U}_{\, t \in [i,j)} \psi$, где $[i, j) \subseteq \mathbb{Q}$ есть интервал рациональных чисел, было предложено в работе [21]. В случае, если используемые интервалы не сингулярны (не состоят из единственной точки), задача верификации для этой логики принадлежит к классу $EXPSPACE$-полных задач. Для случая, позволяющего сингулярные интервалы, задача верификации алгоритмически неразрешима.

Еще хуже дела обстоят с адаптацией подхода переменных часов для случая непрерывного времени, так как для большинства предложенных вариантов задача верификации является алгоритмически неразрешимой [24]. Именно поэтому логики с непрерывным временем не получили такого широкого распространения на практике.

4.1.6. Комбинированная логика $PDL + CL$

Можно выделить два основных класса логических подходов к спецификации программ.

Эндогенные. Такие подходы подразумевают явное описание специфицируемой программы в формуле логики, как это происходит в динамической логике высказываний [68], или, наоборот, включение логических формул

в текст программы, как это происходит в логике Хоара [77].

Экзогенные. В этом случае описание программы включается не в формулы спецификации, а в модели для интерпретации формул. К этому классу относятся темпоральные логики.

Каждый из этих подходов обладает своими плюсами и минусами. К плюсу эндогенных подходов можно отнести большую наглядность и тесную связь программы и спецификации, а плюсом экзогенных логик является более высокая степень абстракции. Некоторым авторам, например [156], удается эффективно сочетать эти два подхода.

Предложенная в [156] логика объединяет возможности коалиционной логики CL по описанию способностей агентов и коалиций с возможностями динамической логики высказываний PDL по описанию действий агентов и их последствий. Язык этой логики формируется из трех алфавитов: пропозициональных символов $Prop$, символов действий $Actions$ и символов коалиций $Coal$ с помощью следующей грамматики:

$$\alpha ::= a \mid \neg\alpha \mid \alpha \wedge \alpha; \tag{4.15}$$

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid [\alpha]\phi \mid \langle\langle A\rangle\rangle\alpha \mid \langle\langle A\rangle\rangle\phi, \tag{4.16}$$

где $a \in Actions$, $p \in Prop$, а $A \in Coal$.

Моделью для комбинированной логики является набор $M = \langle S, AG, AC, R, \mu, \pi, \theta, \gamma\rangle$, где

- S есть непустое конечное множество состояний;
- AG есть непустое конечное множество агентов;
- AC есть непустое конечное множество действий;

- $R \subseteq S \times 2^{AC} \times S$ есть отношение переходов, определяющее возможные изменения состояния при совершении системой некоторого набора действий единовременно;

- $\mu : AG \times S \rightarrow 2^{AC}$ есть функция *возможностей* агента, сопоставляющая каждому агенту множество действий, которое он может выполнить в текущем состоянии системы;

- $\pi : S \rightarrow 2^{Prop}$ есть оценочная функция для пропозиций;

- $\theta : Actions \rightarrow AC$ есть интерпретация символов действий, сопоставляющая каждому символу действия определенное действие из множества AC;

- $\gamma : Coal \rightarrow 2^{AG}$ есть интерпретация символов коалиций, сопоставляющая каждому символу коалиции некоторое подмножество множества AG (т.е. некоторый набор агентов).

Семантика логики описывается с помощью двух отношений выполнимости: $\models_a$, между парой вида (M, ac) (где M есть некоторая модель, а $ac \subseteq AC$ есть некоторое подмножество действий системы) и формулой вида α; и $\models_s$, между парой вида (M, s) (где M есть некоторая модель, а $s \in S$ есть некоторое состояние системы) и формулой вида ϕ. Семантические правила для этих отношений определяются следующим образом.

1. $M, ac \models_a a$, тогда и только тогда, когда $\theta(a) \in AC$.

2. $M, ac \models_a \neg\alpha$, тогда и только тогда, когда выполнено $M, ac \not\models_a \alpha$.

3. $M, ac \models_a \alpha \wedge \beta$, тогда и только тогда, когда выполнено $M, ac \models_a \alpha$ и $M, ac \models_a \beta$.

4. $M, s \models_s p$ тогда и только тогда, когда $p \in \pi(s)$.

5. $M, s \models_s \neg\phi$ тогда и только тогда, когда $M, s \not\models_s \phi$.

6. $M, s \models_s \phi \wedge \psi$ тогда и только тогда, когда $M, s \models \phi$ и $M, s \models_s \psi$.

7. $M, s \models_s \langle\langle A\rangle\rangle\alpha$ тогда и только тогда, когда существует такое действие $ac \in \bigcup\limits_{ag \in A} \mu(ag, s)$, что для него выполнено $M, ac \models_a \alpha$.

8. $M, s \models_s \langle\langle A\rangle\rangle\phi$ тогда и только тогда, когда существует такое действие $ac \in \bigcup\limits_{ag \in A} \mu(ag, s)$, что для любого действия $ac' \in \bigcup\limits_{ag \in AG \setminus A} \mu(ag, s)$ и для любого состояния системы s', если $(s, ac \cup ac', s') \in R$, то выполнено $M, s' \models_s \phi$.

Таким образом, комбинированная логика позволяет в рамках единой спецификации описать возможные изменения состояний системы в зависимости от действий агентов (используя PDL-подобную конструкцию $[\alpha]\phi$), а также описать возможности коалиции по совершению некоторого действия в данный момент ($\langle\langle A\rangle\rangle\alpha$) и по переводу системы в некоторое состояние в следующий момент ($\langle\langle A\rangle\rangle\phi$). Подобная комбинация, с одной стороны, позволяет описывать более наглядные спецификации, но, с другой стороны, ведет к смешению синтаксиса двух логик и, следовательно, затрудняет адаптацию существующих для них инструментов. Более подробно вопрос интеграции логики действий и коалиций изучен в работе [37].

4.2. Спецификация ментальных состояний

При формировании спецификации интеллектуального агента часто возникает необходимость описание таких свойств, как: “Агент ag знает, что ϕ”, “Целью агента ag является ϕ” и “Агент ag планирует осуществить ϕ”. Подобные утверждения относятся к *ментальному* состоянию агента и их невозможно формализовать в рамках описанных ранее логик. В этом разделе рассматриваются несколько подходов к формализации утверждений о ментальных состояниях

агента, а также роль подобных свойств в проектировании и реализации интеллектуального агента. Более подробный обзор современных подходов можно найти в работах [176] и [182].

4.2.1. Логика предикатов первого порядка

Наиболее широко известным формальным методом описания знаний и рассуждений является логика предикатов первого порядка [11, 12]. Поскольку, данный инструмент является классическим и детально изученным во множестве работ, здесь мы не будем приводить формальные определения синтаксиса и семантики логики предикатов первого порядка, предполагая, что читатель с ним уже хорошо знаком.

К сожалению, формальная логика предикатов первого порядка не предоставляет удобных и наглядных инструментов для рассуждений о представлениях агента о мире, даже если сам агент использует для формализации представлений именно логику предикатов. Таким образом, логика предикатов первого порядка не может быть использована для представлений знаний агента о других агентах. Для наглядности, приведем простой пример – рассмотрим выражение "агент ag знает, что $2 * 2 = 4$". Наиболее близкой формулой логики предикатов для этого утверждения является $Know(ag, Equal(Mult(2, 2), 4))$. Но данная формула не является формулой логики предикатов первого порядка, так как в качестве аргумента для предикатного символа $Know$ используется терм с предикатным символом $Equals$, а не с функциональным символом..

Помимо очевидной синтаксической некорректности, приведенное выше высказывание приводит к менее заметной, но не менее значимой семантической проблеме. Рассмотрим выражение $23 * 27 = 621$. Это выражение является истинным, следовательно, с точки зрения логики предикатов первого порядка, эквивалентно выражению $2 * 2 = 4$, и можно произвести замену: $Know(ag, Equal(Mult(2, 2), 4)) \sim Know(i, Equal(Mult(23, 27), 621))$. Таким образом, рав-

нозначны утверждения “агент ag знает, что $2 * 2 = 4$” и “агент ag знает, что $23 * 27 = 621$”, что, очевидно, некорректно.

Семантически значение выражения в логике предикатов первого порядка зависит исключительно от значений его подвыражений. Например, истинность или ложность формулы “$\phi \wedge \psi$” зависит только от истинности или ложности выражений ϕ и ψ. В то же время истинность или ложность утверждения “агент ag знает, что ϕ” зависит не от истинности или ложности ϕ (ведь агент может и ошибаться), а от самого высказывания ϕ и его внутренней структуры.

Одним из методов, позволяющих устранить синтаксическую некорректность формулы, является использование *модальных операторов (modal operators)*. Синтаксически использование модальных операторов аналогично использованию логических связок, но значение их зависит не от значений формул, к которым они применяются, а от самих формул. Альтернативой модальным операторам является введение *метаязыка (metalanguage)*, который является языком предикатов первого порядка, термами для него являются выражения *объектного языка (object language)*, который тоже может являться языком предикатов первого порядка. Каждый из подходов обладает своими плюсами, и своими минусами, которые мы рассмотрим далее.

Для решения семантической проблемы тоже выработано несколько методов решения. Одним из наиболее распространенных методов, является семантика *возможных миров (possible worlds)*. В этом случае представления описываются как множество возможных миров, связанных *отношением допустимости (accessibility relation)*. Использование отношений делает данный инструмент привлекательным для формализации, так как позволяет пользоваться богатым инструментарием теории отношений. Основной проблемой, связанной с семантикой возможных миров, является проблема логического всезнания, подразумевающая, что все агенты являются идеальными логиками, что является слишком сложным условием для ограниченного в ресурсах агента.

Альтернативой семантики возможных миров является метод *интерпретируемых символических структур (interpreted symbolic structures)*. Этот метод предполагает, что знания агентов описываются символическими формулами, явно записанными в структуре данных, ассоциированной с агентом. Таким образом "агент ag знает ϕ", означает, что формула ϕ хранится в структуре представлений агента. Основное преимущество такой схемы заключается в невысокой вычислительной сложности. К недостаткам можно отнести меньшую элегантность и наглядность.

Часто при описании агентов различные аспекты их деятельности описываются различными формальными логиками, для каждой из которых вводятся свои модальные операторы.

Эпистемологическая логика (epistemic logic). Логика для описания знаний агента, содержащая модальный оператор $Know$. Для описания знаний часто используют семантику возможных миров.

Конативная логика (conative logic). Логика для описания желаний и целей агента, содержащая модальный оператор $Goal$. Для описания целей часто используют метод интерпретируемых символических структур.

Хороший пример применения формального логического аппарата для описания мультиагентных систем и ментальных состояний агентов можно найти в работе [181].

4.2.2. Семантика возможных миров

Семантика возможных миров имеет свои корни в философских суждениях об истине [76], и была впервые применена в формальном логическом аппарате Соулом Крипке (Saul Kripke) [106].

Идею возможных миров можно достаточно наглядно описать на примере агента, играющего в преферанс. В преферансе, чем больше вы сможете узнать

о картах оппонентов, тем лучше сможете спланировать стратегию своего поведения. Однако, полное знание в большинстве случаев невозможно. Предположим, в картах на руках агента есть семерка бубен. Следовательно, он наверняка знает, что семерки бубен нет на руках ни у одного из оппонентов. Теперь будем считать, что каждый возможный расклад карт это *мир*. При этом любой мир, в котором агент не держит семерку бубен, является невозможным. После откидывания всех невозможных миров, которые не совместимы с информацией, доступной агенту, остается набор *возможных миров*, или *эпистемологических альтернатив (epistemic alternatives)*. То, что верно во всех возможных мирах, может рассматриваться как знания агента. Например, агент знает, что он держит семерку бубен.

Такой подход обладает рядом ценных свойств. Во-первых, такой метод не зависит от внутренних структур, используемых агентом для хранения знаний. Естественно, не предполагается, что агент содержит у себя в памяти полный список всех возможных миров. В выше описанном примере агенту достаточно запоминать, какие карты есть у него, и какие карты уже вышли из игры. Во-вторых, мощная математическая теория, связанная с семантикой возможных миров, делает её очень привлекательной.

Нормальная модальная логика

Изначально модальная логика разрабатывалась философами, как попытка формализовать различие между *неизбежными (necessary)* истинами и *возможными (contingent)* истинами. Интуитивно понятно, что неизбежные истины это то, что не может быть иначе в принципе, тогда как возможные истины вполне могут в каком-то случае перестать быть истинами. Например, утверждение “за окном идет снег”, верное на момент написания его, вполне может перестать быть истинным.

С другой стороны, выражение “квадратный корень из 2 число иррациональ-

ное" явно относится к истинам необходимым. Необходимые истины обычно представляются как нечто, истинное во всех возможных мирах. К сожалению, помимо законов математики мало что можно считать необходимой истиной.

Синтаксис и семантика Синтаксис нормальной модальной логики определяется на основе алфавита пропозициональных символов $Prop$ следующей грамматикой:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \Box\phi \mid \Diamond\phi, \tag{4.17}$$

где $p \in Prop$. Нормальная модальная логика расширяет стандартную пропозициональную логику двумя модальными операторами *необходимости (necessarily)* "$\Box$" и *возможности (possibly)* "$\Diamond$".

Нормальная модальная логика рассматривает вопрос истинности в мирах. Модель для такой логики состоит из множества всех миров Wи бинарного отношения $\Re \subseteq W \times W$, указывающего, какие миры считаются допустимыми относительно других миров. Формально, модель нормальной модальной логики это тройка $M = \langle W, \Re, \pi \rangle$ где

- W есть непустое конечное множество миров;
- $\Re \subseteq W \times W$ есть бинарное *отношение допустимости* миров;
- $\pi : W \to 2^{Prop}$ есть оценочная функция пропозициональных символов, показывающая какие из них истинны в заданном мире.

Семантика нормальной модальной логики описывается с помощью отношения *выполнимости* $\vDash$ между парой вида M, w (где M есть некоторая модель, а $w \in W$ есть некоторый мир) и формулой языка ϕ. Это отношение определяется следующими семантическими правилами.

1. $M, w \vDash p$ тогда и только тогда, когда $p \in \pi(w)$.

2. $M, w \vDash \neg\phi$ тогда и только тогда, когда $M, w \nvDash \phi$.

3. $M, w \vDash \phi \wedge \psi$ тогда и только тогда, когда $M, w \vDash \phi$ и $M, w \vDash \psi$.

4. $M, w \vDash \Box\phi$ тогда и только тогда, когда для любого мира $w' \in W$, если $(w, w') \in \Re$, то выполнено $M, w' \vDash \phi$.

5. $M, w \vDash \Diamond\phi$ тогда и только тогда, когда существует такой мир $w' \in W$, что $(w, w') \in \Re$ и выполнено $M, w' \vDash \phi$.

Таким образом, формула $\Box\phi$ интерпретируется как истинная для мира w, если для *любого* допустимого относительно w мира w' выполнено свойство ϕ, а формула $\Diamond\phi$ — если *существует* допустимый относительно w мир w', в котором выполнено свойство ϕ. Одним из фундаментальных свойств нормальной модальной логики является двойственность модальных операторов: $\Box\phi \Leftrightarrow \neg\Diamond\neg\phi$.

Введем понятия *общезначимости* и *выполнимости* для нормальной модальной логики.

- Формула ϕ *выполнима*, если существует такая модель M и такой мир в этой модели w, что выполнено $M, w \vDash \phi$.

- Формула ϕ *общезначима на модели* M, если для любого мира в этой модели w выполнено $M, w \vDash \phi$.

- Формула ϕ *общезначима на классе моделей* C, если для любой модели M в этом классе и для любого мира в этой модели w выполнено $M, w \vDash \phi$. Для обозначения общезначимости на классе моделей мы используем нотацию $\vdash_C \phi$.

- Формула ϕ *общезначима* если для любой модели M и для любого мира в этой модели w выполнено $M, w \vDash \phi$. Для обозначения общезначимости мы используем нотацию $\vdash \phi$.

Аксиоматизация Если для формулы нормальной модальной логики ϕ для любой модели M и любого мира w в этой модели выполнено $M, w \vDash \phi$, такая формула называется *общезначимой*. Для обозначения формулы ϕ общезначимости используется нотация $\vdash \phi$. Крипке описал схему общезначимых формул для нормальной модальной логики, названную в его честь аксиомой K:

$$\vdash \Box(\phi \Rightarrow \psi) \Rightarrow (\Box\phi \Rightarrow \Box\psi)$$

Помимо аксиомы K, любая нормальная модальная логика обладает следующим свойством: "если $\vdash \phi$, то $\vdash \Box\phi$", получившим название *правило неизбежности* (necessitation rule).

Существуют классические аксиомы, общезначимые на классе моделей, определенном свойствами отношения $\Re$.

- Если отношение $\Re$ *рефлексивно* ($\forall w \in W : (w, w) \in \Re$), то на классе моделей общезначимы формулы вида $\Box\phi \Rightarrow \phi$. Аксиомы такого вида называют T.

- Если отношение $\Re$ *сериально* ($\forall w \in W : \exists w' \in W : (w, w') \in \Re$), то на классе моделей общезначимы формулы вида $\Box\phi \Rightarrow \Diamond\phi$. Аксиомы такого вида называют D.

- Если отношение $\Re$ *транзитивно* ($\forall w, w', w'' \in W : ((w, w') \in \Re \wedge (w', w'') \in R) \Rightarrow (w, w'') \in \Re$), то на классе моделей общезначимы формулы вида $\Box\phi \Rightarrow \Box\Box\phi$. Аксиомы такого вида называют 4.

- Если отношение $\Re$ *эвклидово* ($\forall w, w', w'' \in W : ((w, w') \in \Re \wedge (w, w'') \in \Re) \Rightarrow (w', w'') \in \Re$), то на классе моделей общезначимы формулы вида $\Diamond\phi \Rightarrow \Box\Diamond\phi$. Аксиомы такого вида называют 5.

Обозначения вида $KA_1A_2...A_n$ (где A_i есть некоторая аксиома) используют для обозначения класса моделей, содержащей аксиомы $A_1, A_2, ..., A_n$ (аксиома Крипке K общезначима для всех моделей [63]).

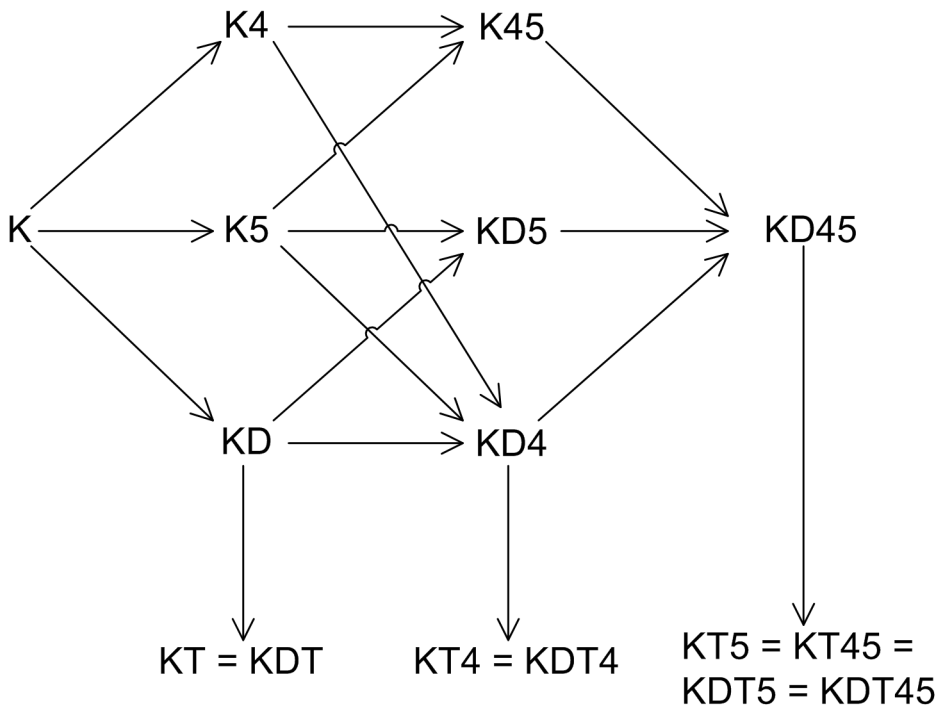

Рис. 4.1. Иерархия классов моделей для нормальной модальной логики.

На первый взгляд, с помощью аксиом T, D, 4 и 5 можно описать 16 различных классов моделей, однако некоторые из этих классов являются эквивалентными [45], так как сериальность очевидно следует из рефлексивности, а транзитивность очевидно следует из эвклидовости и рефлексивности (т.е. $T \Rightarrow D$ и $T \wedge 5 \Rightarrow 4$). Всего эти четыре аксиомы описывают одиннадцать классов, вложенность которых отражена на рисунке 4.1.

Некоторые классы моделей широко используются и получили специальные названия:

- класс KT получил название T;
- класс $KT4$ получил название $S4$;
- класс $KD45$ получил название $weak - S5$;
- класс $KT5$ получил название $S5$;

Нормальная модальная логика как эпистемологическая логика

В случае использования нормальной модальной логики для формализации знаний и представления агента о мире, оператор $\Box\phi$ интерпретируется "агент считает, что ϕ". Миры модели интерпретируются как альтернативы, а отношение допустимости определяет, какие миры могут являться альтернативой данному миру.

При формализации рассуждений о знаниях нескольких агентов, модель нормальной модальной логики расширяется дополнительными отношениями допустимости $\Re_i$ для каждого агента до структуры: $M = \langle W, \Re_1, \Re_2, \ldots, \Re_n, \pi \rangle$, а синтаксис логики расширяется индексированными операторами $\Box_i$ и $\Diamond_i$

Далее рассмотрим значение аксиом, с точки зрения формализации знаний и представлений агента. Начнем с правила неизбежности и аксиомы Крипке, так как они являются универсальными для всех моделей.

Правила неизбежности утверждает "если $\vdash \phi$, то $\vdash \Box\phi$", что означает, что агент знает все общезначимые формулы. Т.е. как минимум, агент знает все общезначимые формулы пропозициональной логики. Поскольку число общезначимых формул пропозициональной логики бесконечно, знания агента так же бесконечны.

Аксиома Крипке $\vdash (\phi \Rightarrow \psi) \Rightarrow (\Box\phi \Rightarrow \Box\psi))$ показывает, что знания и представления агента замкнуты относительно импликации. Если агент считает, что верна каждая формула конъюнкта $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$ $(\Box(\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n))$, а также считает что ϕ является логическим следствием этого конъюнкта $(\Box(\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \Rightarrow \phi))$, то агент считает, что верно ϕ $(\Box\phi)$. Таким образом, агент знает все логические следствия своих знаний. Т.е. агент является идеальным логиком, каковым ограниченный в ресурсах агент быть не может, что и приводит к проблеме *логического всезнания* [103, 113].

Рассмотрим уместность аксиом D_i, T_i, 4_i и 5_i (индекс при имени аксиомы

показывает, что она общезначима для модели соответствующего индексу агента).

Аксиома D_i ($\Box_i\phi \Rightarrow \Diamond_i\phi$) с использованием двойственности модальных операторов может быть записана в виде "$\Box_i\phi \Rightarrow \neg\Box_i\neg\phi$", что читается как "если агент i считает, что ϕ, то агент i не считает что $\neg\phi$". Таким образом, аксиома D_i требует непротиворечивости представлений агента.

Аксиома T_i ($\Box_i\phi \Rightarrow \phi$) часто называется *аксиомой знаний*, так как она требует, чтобы все представления агента были верными. Именно аксиому T_i используют для различения *знаний* и *представлений*. Т.е. с этой точки зрения верные преставления являются знаниями.

Аксиомы 4_i ($\Box_i\phi \Rightarrow \Box_i\Box_i\phi$) называют аксиомой позитивного самоанализа, так как она требует, чтобы агент знал, что он знает. По аналогии аксиома 5_i ($\Diamond_i\phi \Rightarrow \Box_i\Diamond_i\phi$ или, по правилам двойственности, $\neg\Box_i\neg\phi \Rightarrow \Box_i\neg\Box_i\neg\phi$) называется аксиомой негативного самоанализа, так как утверждает, что агент знает, чего он не знает.

Вопросы алгоритмической сложности для нормальной модальной логики исследованы в работе [67]. Пусть запись вида A^n означает, что аксиома A верна для каждого агента системы из n агентов. Задача проверки общезначимости и выполнимости формул для логик класса K^n, T^n, $S4^n$, (при $n \geq 1$) и $weak-S5^n$, $S5^n$ (при $n \geq 2$) являются алгоритмически разрешимыми и принадлежат к классу $PSPACE$-полных задач. Однако, задача верификации для нормальной модальной логики может быть разрешена более эффективно (за линейное время).

Общее и распределенное знание

В дополнение к рассуждениям о знаниях отдельного агента, полезно иметь инструментарий для работы со знаниями системы в целом. Здесь можно выделить два типа таких знаний.

Общее знание. Знания "культурной среды", известные каждому агенту. Более того, каждый агент знает, что все это знают, знает, что все знают, что все знают и т.д.

Распределенное знание. Знания, не содержащиеся в отдельном агенте, но содержащиеся в системе в целом. Например, один из агентов знает ϕ, а другой $\phi \Rightarrow \psi$, следовательно, системе известно ψ, хотя не один из агентов напрямую ψ не знает.

Формальное определение общего знания дается с помощью оператора "все знают: $EK\phi \triangleq \bigwedge_{i=1}^{n} \Box_i \phi$. Однако сам оператор еще не удовлетворяет условиям общего знания, так как не требует знания о знаниях, поэтому продолжим его уточнение рекурсивно:

$$\begin{aligned} EK^1\phi &\triangleq EK\phi \\ EK^{k+1}\phi &\triangleq EK(EK^k\phi) \end{aligned}$$

Таким образом, оператор общего знания может быть определен как объединение всех EK_i:

$$CK\phi \triangleq \bigwedge_{k=1}^{\infty} EK^k\phi \tag{4.18}$$

Распределенное знание это удобная концепция для формализации рассуждений о совместном решении проблем агентами и для его обозначения используется оператор DK. Оператор распределенного знания, к сожалению, не может быть сведен к уже введенным операторам, поэтому его формальное определение дается с помощью введения собственного семантического правила.

6. $M, w \vDash DK\phi$ тогда и только тогда, когда для всех $w' \in W$ если $(w, w') \in \bigcup_{i=1}^{n} R_i$, то выполнено $M, w' \vDash \phi$.

4.2.3. Метаязыки

Альтернативой модальным операторам является введение *метаязыка*, позволяющего использовать в качестве термов выражения *объектного* языка. Для обеспечения читаемости формул часто используется синтаксическая конструкция $\lceil\ \rceil$. Допустим, что ϕ является выражением объектного языка, тогда $\lceil\phi\rceil$ это обозначение терма метаязыка, обозначающего ϕ. Часто обозначение $\lceil\phi\rceil$ называют Геделевым номером ϕ, так как именно Гедель первым использовал эту синтаксическую конструкцию при доказательстве теоремы о неполноте формальной арифметики.

Для описания представлений о мире в метаязыке вводится предикат $Consider(\lceil\phi\rceil)$, означающий "агент считает, что утверждение, представляемое термом $\lceil\phi\rceil$ верно", что эквивалентно "агент считает, что утверждение ϕ верно". При рассуждении о нескольких агентах используют индексированный предикат, причем для большей наглядности индекс принято писать в качестве первого аргумента предиката: $Consider(i, \lceil\phi\rceil)$ — "i считает, что утверждение ϕ верно".

Считается, что метаязыки обладают рядом преимуществ, по сравнению с модальными операторами.

Выразительная мощность. Некоторые утверждения, описанные с использование метаязыка, не могут быть выражены с помощью модальных операторов. Например, "i имеет какое-то представление о мире" $(\exists x : Consider(i, x))$, или "$i$ считает, что все представления j верны" $(\exists x : Consider(j, x) \Rightarrow Consider(i, x))$.

Удобство вычислений. Метаязык является языком предикатов первого порядка, что позволяет использовать для автоматических доказательств богатый набор средств и наработок в этой области.

Однако метаязыки обладают целым рядом серьезным недостатков [142–144]. Если использовать метаязык для описания ментальных систем второго порядка, т.е. систем обладающих знаниями о знаниях, то возникает необходимость использовать в качестве термов метаязыка выражения самого метаязыка, что приводит к *самоссылочности* и, в большинстве случаев, к противоречивости такой системы. Для адекватного описания знаний в таком метаязыке необходимо вводить *предикат истинности*:

$$True(\ulcorner\phi\urcorner) \Leftrightarrow \phi. \tag{4.19}$$

В этом случае предикат для знания может быть описан следующим образом (знания, это истинные представления):

$$Know(\ulcorner\phi\urcorner) \Leftrightarrow Consider(\ulcorner\phi\urcorner) \wedge True(\ulcorner\phi\urcorner). \tag{4.20}$$

На первый взгляд, такое описание выглядит естественным и интуитивно понятным, однако было доказано, что любой самоссылочный метаязык, содержащий предикат истинности в таком виде является противоречивым. И несмотря на то, что было предложено достаточно много схем, способных разрешить этот парадокс, они тоже оказывались противоречивыми.

Альтернативным вариантом самоссылочным метаязыкам является введение иерархии языков:

$$L_0 \rightarrow L_1 \rightarrow \ldots \rightarrow L_k \rightarrow \ldots, \tag{4.21}$$

где язык L_0 является объектным языком (например, классическим языком логики предикатов первого порядка), а язык L_k, где $k > 0$, является метаязыком, использующим в качестве термов выражения языков, находящихся ниже по иерархии ($L_i : i < k$). Однако высказывания в иерархии языков сложнее для понимания и не могут выразить некоторые аспекты описания знаний о знаниях. Например, выражение "i считает, что все представления j верны" не может

быть представлено в иерархии языков, так как невозможно выбрать конкретный язык в иерархии для описания утверждения.

4.2.4. BDI-логики

Корни *ментальноого* (Beliefs-Desires-Intentions) подхода [39, 40] к моделированию интеллектуальных агентов лежат в философских подходах к анализу мыслительной деятельности человека и того, как люди на практике принимают решения, о том, что им следует делать. Можно выделить следующие отдельные этапы в принятии решения *ментальным* агентом: сначала агент должен понять, *чего* он хочет, затем определить *какие* цели из желаемых он будет пытаться реализовать, а затем понять, *как* он будет реализовывать выбранные цели. При этом в состоянии агента четко разграничиваются следующие компоненты.

Представления (Beliefs). Некоторая информация о закономерностях и текущем состоянии внешней среды, которой располагает агент. При этом предполагается, что эта информация может быть ошибочной и неполной, поэтому её можно рассматривать только как представления, но не как достоверные знания.

Желания (Desires). Множество всех целей, которых агент хотел бы добиться. При этом множество может быть большим и противоречивым. Маловероятно, что агент, ограниченный ресурсами, сможет реализовать все свои желания.

Намерения (Intentions). Множество тех целей, которых агент решил добиться. Сформированное множество целей должно быть *выполнимо* по представлениям агента, т.е. все намерения агента должны быть достижимы в совокупности.

Ключевым понятием процесса принятия решения в ментальной архитектуре является именно *намерение*. В целом можно выделить следующие свойства намерений.

- Намерения задают направление деятельности — агент пытается найти действия, способные осуществить намерения и выполнить их.
- Намерения ограничивают будущий выбор — агент не может формировать новые намерения, несовместимые с уже принятыми, т.е. ведущие к *невыполнимости* множества намерений.
- Намерения имеют долгое время жизни — если агент сформировал план реализации намерения, но он провалился, то агент будет формировать новые планы и пытаться реализовать намерение другим способом. Намерение может быть отброшено только при осуществлении определенного ментального усилия в случаях, если агент пришел к выводу, что реализовать намерение невозможно (не удается сформировать план, ведущий к достижению намерения) или оно уже потеряло актуальность для агента.
- Намерения влияют на рассуждения о будущем и, соответственно, планы — если агент выработал намерение, то он может строить планы на будущее с предположением, что это намерение реализовано.

Этот подход к моделированию был адаптирован целым семейством логик, получившем название *BDI-логики*. Обзор логик этого семейства приведен в данном разделе.

Логика намерений Левескйе

Именно в этой логике, предложенной в [49], было впервые формализовано понятие *намерения*. Эта логика расширяет логику предикатов первого порядка четырьмя индексированными модальными операторами:

- $\mathsf{Bel}_{ag}\ \phi$ — агент ag считает, что верно ϕ;
- $\mathsf{Goal}_{ag}\ \phi$ — агент ag имеет целью, чтобы стало верно ϕ;
- $\mathsf{Happens}\ \alpha$ — в следующий момент произойдет действие α;
- $\mathsf{Done}\ \alpha$ — действие α завершено к текущему моменту.

Семантика операторов Bel_{ag} и Goal_{ag} описывается с использованием семантики возможных миров, при этом отношение допустимости миров для оператора Bel_{ag} должно быть эвклидовым, транзитивным и сериальным (что ведет к логике класса $KD45$), а отношение допустимости должно быть сериальным (логика класса KD). Миром в семантике Левескйе является последовательность *событий* бесконечная в прошлом и будущем. Для описания таких последовательностей предложен логический язык, во многом аналогичный языку, используемому в динамической логике высказываний регулярных программ для описания программ (см. грамматику 4.2), а модальные операторы $\mathsf{Happens}\ \alpha$ и $\mathsf{Done}\ \alpha$ используются для проверки, удовлетворяет ли хвост цепочки, начинающийся от текущего момента ($\mathsf{Happens}$), или голова цепочки до текущего момента (Done) выражению α. Интересно, что используя оператор $\mathsf{Happens}$ можно выразить основные темпоральные операторы "в следующий момент ϕ" ($\bigcirc\phi$), "когда-нибудь ϕ, а до этого ψ" ($\phi\,\mathcal{U}\,\psi$) и "всегда ϕ" ($G\,\phi$):

$$\bigcirc\,\phi \triangleq \mathsf{Happens}\ \varepsilon;\phi?. \tag{4.22}$$

$$\phi\,\mathcal{U}\,\psi \triangleq \mathsf{Happens}\ ((\phi?;\varepsilon)+\psi?)*, \tag{4.23}$$

где ε есть недетерминированный выбор среди всех возможных элементарных событий $Actions$ ($\varepsilon \triangleq \underset{a\in Actions}{+} a$). Используя основные темпоральные операторы стандартным образом можно вывести операторы дополнительные:

- $\mathcal{F}\,\phi \triangleq true\,\mathcal{U}\,\phi$, интуитивно "когда-нибудь ϕ";

- $\mathcal{F}_s\,\phi \triangleq true\,\mathcal{U}\,\phi$, интуитивно “когда-нибудь, не включая текущий момент, ϕ”;

- $G\,\phi \triangleq \neg\mathcal{F}\,\neg\phi$, интуитивно “всегда ϕ”;

- $\phi\,\mathcal{B}\,\psi \triangleq \neg(\neg\phi\,\mathcal{U}\,\psi)$, интуитивно “если когда-нибудь ψ, то хотя бы в один момент до этого ϕ”;

Важным понятием логики Левескйе является понятие *постоянной цели*. ϕ является постоянной целью для агента ag, если

- Агент считает, что ϕ не выполнено в данный момент и имеет цель добиться того, что ϕ когда-нибудь будет выполнено.

- Агент может отбросить цель ϕ только если он считает, что цель уже достигнута или считает, что её невозможно достичь.

Формально оператор постоянной цели $\mathsf{PGoal}_{ag}\ \phi$ определяется следующим образом:

$$\mathsf{PGoal}_{ag}\ \phi \triangleq \begin{array}{c}(\mathsf{Bel}_{ag}\ \neg\phi) \wedge (\mathsf{Goal}_{ag}\ \mathcal{F}_s\,\phi) \\ \wedge\,((\mathsf{Bel}_{ag}\ \phi) \vee (\mathsf{Bel}_{ag}\ G\neg\phi)\ \ \mathcal{B}\,\neg\mathsf{Goal}_{ag}\ \mathcal{F}_s\,\phi)\end{array} \tag{4.24}$$

В определенном смысле, постоянная цель уже является намерением агента, но в работе [49] это понятие уточняется далее и вводится два дополнительных оператора “агент ag намеревается совершить действие α” и “агент ag намеревается добиться того, что ϕ будет истинно”.

Логика намерений Левескйе является мощным формальным аппаратом для рассуждений о ментальных состояниях агента и была с успехом применена в нескольких работах, посвященных кооперации агентов и кооперативному решению проблем [93, 114]. Однако не лишена она и недостатков, обзор которых можно найти в работе [165].

BDI-логики Рао и Джорджеффа

Это семейство логик, по мнению [176], является наиболее представительным в классе BDI-логик. Первая логика из этого семейства была предложена в 1991 году в работе [151, 152] и затем активно развивалась в работах [154, 155], а также была с успехом применена при разработки системы управления воздушным движением для аэропорта Сиднея OASIS [117, 153].

BDI-логика расширяет ветвящуюся темпоральную логику CTL (см. раздел 4.1.3) тремя индексированным модальными операторами Bel_{ag} , Des_{ag} и Intend_{ag} для описания представлений, желаний и намерений агента соответственно. Семантика этих операторов, опять же, задается с помощью возможных миров, причем каждый мир описывает множество состояний и отношения переходов для интерпретации формул темпоральной логики.

Обозначим множество всех состояний системы как S, а множество всех возможных её переходов как $R \subseteq S \times S$. В этом случае один мир это пара вида $w = (S_w, R_w)$, где $S_w \subseteq S$ есть некоторое подмножество множества состояний, а $R_w \subseteq S_w \times S_w$ есть отношение переходов на множестве S_w, удовлетворяющее условию $R_w \subseteq R$. Множество всех миров для S и R обозначим как $W_{S,R}$. *Ситуацией* называется пара вида (w, s), где $w \in W_{S,R}$ есть мир, а $s \in T_w$ есть некоторое состояние из возможных в этом мире.

Отношения допустимости в BDI-логике имеют вид $W \times T \times W$ и для каждого агента ag системы определено три таких отношения: отношение представлений $B_{ag} \subseteq W \times T \times W$, отношение желаний $D_{ag} \subseteq W \times T \times W$ и отношение намерений $I_{ag} \subseteq W \times T \times W$. Семантика модальных операторов определяется относительно текущей ситуации (w, t) следующим образом.

1. $(w, t) \vDash \mathsf{Bel}_{ag}\ \phi$ тогда и только тогда, когда для любого $w' \in W$, если $(w, t, w') \in B_{ag}$, то выполнено $(w', t) \vDash \phi$.

2. $(w, t) \vDash \mathsf{Des}_{ag}\ \phi$ тогда и только тогда, когда для любого $w' \in W$, если

$(w, t, w') \in D_{ag}$, то выполнено $(w', t) \models \phi$.

3. $(w, t) \models \mathsf{Intend}_{ag}\ \phi$ тогда и только тогда, когда для любого $w' \in W$, если $(w, t, w') \in I_{ag}$, то выполнено $(w', t) \models \phi$.

Таким образом, введенные операторы имеют семантику, аналогичную оператору неизбежности $\Box\phi$ в нормальной модальной логике. Как и для нормальной модальной логики, накладывая ограничения на структуру отношений B, D и I можно добиться того, что формулы некоторого вида станут общезначимыми. Например, если для отношений выполнено $D_{ag} \subseteq I_{ag}$, то общезначимы формулы вида $\mathsf{Intend}_{ag}\ \phi \Rightarrow \mathsf{Des}_{ag}\ \phi$. Более подробный анализ подобных зависимостей можно найти в работе [154].

KARO-логика

Логика знаний, действий, результатов и возможностей (Knowledge, Actions, Results and Opportunities, *KARO*) [157, 178] является обобщением идей Мура [130] по интеграции динамической логики и логики описания знаний.

Если рассматривать динамическую логику высказываний *PDL* (см. раздел 4.1.1) как основу, то логика *KARO* добавляет к ней следующие модальные операторы:

- $\mathsf{Know}_{ag}\ \phi$ — модальный оператор знаний агента ag, семантика которого задается с помощью возможных миров класса $S5$;
- $\mathsf{do}_{ag}\ \alpha$ — модальный оператор выполнения действия α агентом ag. Именно формулы вида $\mathsf{do}_{ag}\ \alpha$ являются параметрами операторов динамической логики $[\mathsf{do}_{ag}\ \alpha]$ и $\langle \mathsf{do}_{ag}\ \alpha \rangle$;
- $\mathsf{A}_{ag}\ \alpha$ — оператор способности агента ag выполнить действие α.

Понятие *знаний* логики $KARO$ очевидно соответствует оператору $\mathsf{Know}_{ag}\ \phi$. Действия, по аналогии с PDL, комбинируются из атомарных действий $AC = \{a_1, a_2, \ldots, a_n\}$ с помощью операторов последовательного выполнения $(\alpha_1; \alpha_2)$, недетерминированного выбора $(\alpha_1 + \alpha_2)$, проверки условия $(\phi?)$ и итерации $(\alpha*)$. Результаты описываются оператором $[\mathsf{do}_{ag}\ \alpha]\phi$ (результатом выполнения агентом ag действия α *будет* ϕ), а возможности (opportunity) — оператором $\langle \mathsf{do}_{ag}\ \alpha\rangle\phi$ (результатом выполнения агентом ag действия α *может* стать ϕ). Оператор $\mathsf{A}_A\ \alpha$ описывает *способность* (ability) агента ag выполнить действие α.

Комбинация возможности и способности позволяет описать понятие *практической реализуемости*: $\mathsf{PracPoss}_{ag}\ (\alpha, \phi) \triangleq \langle ag\alpha\rangle\phi \wedge \mathsf{A}_{ag}\ \alpha$. Свойство ϕ практически реализуемо агентом ag, если агент способен совершить действие α и результатом этого может стать ϕ. Используя оператор практической реализуемости, можно ввести оператор осуществимости $\Diamond_{ag}\phi$ — свойство ϕ осуществимо агентом ag, если существует такая последовательность действий $(a_1, \ldots, a_k) \in S^k$, что $\mathsf{PracPoss}_{ag}\ (a_1; \ldots; a_k, \phi)$.

В работе [123] предлагается расширение логики $KARO$ операторами желания $(\mathsf{W}_{ag}\phi)$, выбранных желаний $(\mathsf{W}_{ag}\phi)$ и обязательств $\mathsf{Committed}_{ag}\alpha$, а также выделенными атомарными действиями для выбора новых желаний $\mathsf{select}_{ag}\phi$, принятия новых обязательств $\mathsf{commit\_to}_{ag}\alpha$ и отказа от существующих обязательств $\mathsf{uncommit}_{ag}\alpha$. Семантика оператора $\mathsf{W}_{ag}\phi$ описывалась с помощью возможных миров, а операторов $\mathsf{W}_{ag}\phi$ и $\mathsf{Committed}_{ag}\alpha$ — с помощью интерпретируемых символических структур (с каждым агентом ассоциировано множество формул состояний C, описывающего выбранные им желания, а также множество формул действий $Agenda$, описывающих его обязательства). Используя эти операторы можно формализовать понятие цели агента следующим образом:

$$\mathsf{Goal}_{ag}\ \phi \triangleq \mathsf{W}_{ag}\phi \wedge \neg\phi \wedge \Diamond_{ag}\phi \wedge \mathsf{C}_{ag}\phi. \tag{4.25}$$

$\mathcal{LORA}$, $\mathcal{MORA}$ и $MABEL$

Одним из последних достижений в сфере разработки логик спецификации мультиагентных систем можно считать логику рациональных агентов (Logics of Rational Agency, $\mathcal{LORA}$) [182], а также ее упрощенный вариант $\mathcal{MORA}$ [129], положенный в основу языка $MABEL$ [128].

$\mathcal{LORA}$ представляет собой предикатный вариант расширенной логики ветвящегося времени $CTL*$ 4.1.3, дополненный следующими операторами:

- модальными операторами $\mathsf{Bel}_{ag}\ \phi$, $\mathsf{Des}_{ag}\ \phi$ и $\mathsf{Intend}_{ag}\ \phi$ для описания представлений, желаний и намерений агента;
- средствами динамической логики 4.1.1 для описания действий ($\alpha;\alpha$,$\alpha + \alpha$,$\phi?$ и $\alpha*$) и их последствий ($[\alpha]\phi$ и $\langle\alpha]\rangle\phi$);
- модальным оператором $\mathsf{Happens}\ \alpha$ для обозначения того, что в следующий момент будет выполнено действие α;
- *действиями* $\mathsf{send}_{ag}\ \mathsf{pr}\ ag'\ \mathsf{of}\ \phi$ и $\mathsf{receive}_{ag}\ \mathsf{pr}\ ag'\ \mathsf{of}\ \phi$ позволяющими описывать коммуникацию агентов.

Таким образом, логика $\mathcal{LORA}$ является комбинацией и обобщением большинства рассмотренных нами ранее логик, а также предоставляет дополнительные возможности по описанию коммуникации [2], которые мы рассмотрим более подробно.

Рассмотрение коммуникации агентов как разновидности действия имеет свои корни в философской теории речевого действия (speech act theory) [29, 161], которая активно используется при разработки коммуникационных протоколов

[2] Хотя формально введенные конструкции $\mathsf{send}_{ag}\ \mathsf{pr}\ ag'\ \mathsf{of}$ и $\mathsf{receive}_{ag}\ \mathsf{pr}\ ag'\ \mathsf{of}$ не добавляют выразительной мощности (они могут быть выражены с помощью комбинаций элементарных действий определенного вида), использование явных конструкций значительно облегчает написание формул и увеличивает их читаемость.

и языков, в том числе и для мультиагентных систем [58, 119]. Ключевым понятием этой теории является понятие *перформатива* — глагола, придающего сообщению смысл определенного действия. К типичным глаголам-перформативам можно отнести такие глаголы как “сообщать”, “просить”, “приказывать” и т.д.

Чтобы лучше понять смысл и роль перформативов рассмотрим две фразы “Я сообщаю Вам, что за окном солнечная погода” и “Я гуляю по улице в солнечный день”. Первая фраза, будучи произнесенной и услышанной, приведет к тому, что будет выполнено действие “сообщать”, тогда как вторая фраза является просто декларацией факта и никак не связана с действием “гулять”. Таким образом, именно перформативы позволяют придать сообщениям суть действий.

Конструкция send_{ag} pr ag' of ϕ используется для обозначения действия посылки агентом ag агенту ag' сообщения с перформативом pr и информационной частью ϕ, а конструкция $\mathsf{receive}_{ag}$ pr ag' of ϕ — для обозначения действия получения агентом ag от агента ag' сообщения с перформативом pr и информационной частью ϕ. Приведем несколько формул, иллюстрирующих использование этих действий:

- send_{ag} inform ag' of $\phi \Rightarrow \mathbf{A}\mathcal{F}\,\mathsf{Happens}\ \mathsf{receive}_{ag'}$ inform ag of — посланное агентом ag сообщение будет когда-нибудь получено агентом ag';

- $\mathsf{Happens}\ \mathsf{receive}_{ag'}$ inform ag of $\phi \Rightarrow \mathbf{A} \bigcirc \mathsf{Bel}_{ag'}\ \phi$ — получив информационное сообщение о ϕ от агента ag, агент ag' соответствующим образом меняет свои представления;

- $(\mathsf{Happens}\ \mathsf{receive}_{ag'}$ request ag of $\phi) \Rightarrow \mathbf{A} \bigcirc (\mathsf{Des}_{ag}\ \phi \wedge \mathsf{Intend}_{ag'}\ \phi \wedge \mathsf{Happens}$ $\mathsf{send}_{ag'}$ confirm ag of $\phi)$ — получив запрос о ϕ от агента ag, агент ag' добавляет ϕ к своим желаниями и намерениям, а также посылает подтверждение агенту ag.

Логика $\mathcal{MORA}$ представляет собой упрощенный вариант $\mathcal{LORA}$, где, например, представлениями, желаниями и намерениями агента могут являться только формулы состояния. Именно эта логика используется для формального описания семантики языка $MABEL$, применяемого для спецификации и верификации мультиагентных систем и протоколов. Агенты $MABEL$ обладают представлениями, желаниями и намерениями, а также могут обмениваться сообщениями, используя язык Agent Communication Language ACL [58], предложенный сообществом Foundation for Intelligent Physical Agents $FIPA$ [59]. Язык позволяет описать последствия определенных действий агентов, специфицировать желаемые свойства системы, а затем проверить, удовлетворяет ли система спецификации. Для верификации системы спецификация транслируется в логику линейного времени LTL, а описание системы в язык $PROMELA$, после чего полученное описание и спецификация передаются верификатору $SPIN$ [80].

4.3. Логика спецификации интеллектуального агента

В данном разделе предлагается логика спецификации свойств модели интеллектуального агента (см. главу 2), объединяющая возможности пропозициональной динамической логики PDL и логик ветвящегося времени $RTCTL$, а также расширяющая их следующими дополнительными операторами:

- $\mathsf{Bel}\ \phi$ — оператор, используемый для описания представлений агента. Семантика этого оператора задается с помощью возможных миров, определяемых восприятием агента $see \subseteq S \times S$ и представлениями агента $bel \subseteq P \times A \times P$;

- $\mathsf{Des}\ \phi$ — оператор, используемый для описания желаний агента. Семантика этого оператора задается относительно структуры желаний агента des;

- Intend ϕ — оператор, используемый для описаний намерений агента. Семантика этого оператора задается относительно представлений агента bel и его плана $plan$.

4.3.1. Синтаксис

Определение 8 *Язык логики описания свойств интеллектуального агента формируется из следующих алфавитов.*

- *Алфавитов предметных переменных состояния* Var_s *и действия* Var_a.
- *семейства алфавитов функциональных символов* $Func_i$, *где* $i \in \{0, \ldots, n_f\}$ *есть размерность символа, а* n_f *есть максимальная размерность функциональных символов. Функциональные символы размерности* 0 *соответствуют предметным константам.*
- *Семейства алфавитов предикатных символов* $Pred_i$, *где* $i \in \{0, \ldots, n_p\}$ *есть размерность символа, а* n_p *есть максимальная размерность предикатных символов. Предикатные символы размерности* 0 *соответствуют пропозициональным константам.*

Синтаксис логики определяется следующей грамматикой:

$$\tau^a ::= v_a \mid f_i(\tau_1^a, \ldots, \tau_i^a), \quad (v_a \in Var_a, f_i \in Func_i); \tag{4.26}$$

$$\tau^s ::= v_s \mid f_i(\tau_1^s, \ldots, \tau_i^s), \quad (v_s \in Var_s, f_i \in Func_i); \tag{4.27}$$

$$\alpha ::= P_i(\tau_1^a, \ldots, \tau_i^a) \mid \alpha \wedge \alpha \mid \neg\alpha, \quad (P_i \in Pred_i); \tag{4.28}$$

$$\varepsilon ::= P_i(\tau_1^s, \ldots, \tau_i^s) \mid \varepsilon \wedge \varepsilon \mid \neg\varepsilon \mid [\alpha]\varepsilon, \quad (P_i \in Pred_i); \tag{4.29}$$

$$\phi ::= \varepsilon \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{A}\,\rho \mid \mathbf{E}\,\rho \mid \mathsf{Bel}\ \phi \mid \mathsf{Des}\ \phi \mid \mathsf{Intend}\ \phi; \tag{4.30}$$

$$\rho ::= \bigcirc\phi \mid \phi\,\mathcal{U}_{\leq t}\phi, \quad (t \in \mathbb{N} \cup \{\infty\}). \tag{4.31}$$

Формулы вида τ^a являются *термами действий системы*, а формулы вида α являются *формулами действий системы* и задают некоторые множества

действий системы. Формулы вида τ^s являются *термами состояний среды*, а формулы вида ε являются *формулами состояний среды* и задают некоторые множества состояний среды. Формулы α и ε задают синтаксис бескванторной логики предикатов первого порядка для описания действий системы и состояний внешней среды. Кроме того, формулы вида ε расширяют логику предикатов оператором динамической логики $[\alpha]\varepsilon$, интуитивно интерпретируемым как "при выполнении системой действия, удовлетворяющего условию α внешняя среда перейдет в одно из состояний множества ε".

Формулы вида ϕ расширяют формулы вида ε следующими синтаксическими конструкциями.

- Кванторы пути $\mathbf{A}\,\rho$ и $\mathbf{E}\,\rho$. Аналогичны кванторам пути логики ветвящегося времени и интерпретируются как "любой возможный путь взаимодействия агента со средой удовлетворяет свойству ρ" и "существует такой возможный путь взаимодействия агента со средой, который удовлетворяет свойству ρ".

- Оператор представлений агента $\mathsf{Bel}\ \phi$. Интуитивная интерпретация этого оператора звучит как "агент считает что верно ϕ".

- Оператор желаний агента $\mathsf{Des}\ \phi$. Интуитивная интерпретация этого оператора звучит как "агент хочет, чтобы было верно ϕ".

- Оператор намерений агента $\mathsf{Intend}\ \phi$. Интуитивная интерпретация этого оператора звучит как "агент планирует вести себя так, что, по его представлениям, будет верно ϕ".

Формулы вида ρ являются формулами пути и позволяют использовать следующие темпоральные операторы: "в следующий момент ϕ" ($\bigcirc\phi$) и "не более чем через t шагов (когда-нибудь, если $t = \infty$) ϕ_2, а до этого ϕ_1" ($\phi_1\, \mathcal{U}_{\leq t} \phi_2$).

4.3.2. Семантика

Для логики описания интеллектуального агента можно предложить два варианта описания семантики.

С динамическим ментальным состоянием. В этом случае предполагается, что агент обновляет свое ментальное состояние (представления, желания и намерения) на каждом шаге взаимодействия с внешней средой. Такая семантика является наиболее общей, однако, вычислительная сложность верификации формул в ней значительно выше, так как количество возможных ментальных состояний экспоненциально больше количества состояний внешней среды и возможных действий агента.

Со статическим ментальным состоянием. В этом случае предполагается, что ментальное состояние агента не изменяется со временем. Подобное предположение вполне соответствует поведению систем реального времени на ограниченных участках их жизненного пути, так как пересмотр агентом своего ментального состояния происходит относительно редко (при завершении или провале исполнения плана, а также при постановке новых целей). При изменении ментального состояния агент должен пройти повторную верификацию.

Семантика с динамическим ментальным состоянием

Описание семантики начнем с определения *модели.*

Определение 9 *Моделью для интерпретации формул логики спецификации интеллектуального агента (формул вида ϕ из определения 8) является набор $M = \langle ag, Domain, \pi_s, \pi_a, \{\Phi_i\}_{i\in\{0,\ldots,n_f\}}, \{\Pi_i\}_{i\in\{0,\ldots,n_p\}}\rangle$, где*

- *$ag = (S, A, env, see, I_B, bel, brf, I_D, des, drf, plan, prf)$ есть модель интеллектуального агента из определения 4, где $I_D = 2^{2^\phi}$ (таким образом, в*

качестве возможных способов описания функций-критериев мы ограничиваемся описанием с помощью набора логических формул);

- *$Domain$ есть множество, определяющее возможные значения переменных (доменное множество);*

- *$pi_s : S \times Var_s \to Domain$ и $pi_a : A \times Var_a \to Domain$ есть интерпретации символов переменных состояний внешней среды и действий агента соответственно, сопоставляющая состоянию внешней среды (или действию агента) и символу переменной некоторое значение предметной области;*

- *$\Phi_i : Func_i \to \{f \mid f : Domain^i \to Domain\}$ есть интерпретация функциональных символов размерности i, сопоставляющая каждому символу некоторую функцию из множества $Domain^i$ в множество $Domain$;*

- *$\Pi_i : Pred_i \to \{P \mid P : Domain^i \to \{0,1\}\}$ есть интерпретация предикатных символов размерности i, сопоставляющая каждому символу некоторую функцию из множества $Domain^i$ в множество $\{0,1\}$.*[3]

Заметим, что при таком определении модели только интерпретации символов переменных зависят от текущего состояния среды или действия агента, а доменное множество и интерпретации функциональных и предикатных символов являются фиксированными. Такое ограничение часто называют *предположением о постоянстве домена* [181].

Определение 10 *Семантику логики спецификации интеллектуального агента с динамическим ментальным состоянием зададим с помощью трех отношений:*

[3]Здесь и далее для обозначения пропозициональной истины и пропозициональной лжи будем взаимозаменяемо использовать как символы 1 и 0, так и символы $true$ и $false$.

- *для формул действий* $M, a \models_a \alpha$;
- *для формул состояний* $M, (env_c, act_c), (s, i, i_{act}) \models_s \phi$;
- *для формул пути* $M, (env_c, act_c), \lambda \models_p \rho$;

где

- $M = \langle ag, Domain, \pi_s, \pi_a, \{\Phi_i\}_{i\in\{0,\ldots,n_f\}}, \{\Pi_i\}_{i\in\{0,\ldots,n_p\}}\rangle$ *есть модель;*
- $a \in A$ *есть некоторое действие агента;*
- $env_c \subseteq S \times A \times S$ *есть описание поведения внешней среды в текущем контексте;*
- $act_c \subseteq S \times I \times I_{act} \times A \times I_{act}$ *есть отношение, описывающее способ выбора действия в текущем контексте. При этом множество* I_{act} *описывает потенциальные дополнительные параметры выбора;*
- $(s, i, i_{act}) \in S \times I \times I_{act}$ *есть текущее состояние системы, включая состояние внешней среды* s*, внутреннее состояние агента* i *и текущее значение дополнительного параметра выбора действия* i_{act}*;*
- $\lambda \in (S \times I \times I_{act})^+$ *есть некоторый путь, представленный цепочкой троек вида* $(s, i, i_{act}) \in S \times I \times I_{act}$.

Определение 11 *Для агента* ag основной контекст (env_{ag}, act_{ag}) *определяется следующим образом:*

$$env_{ag} \triangleq env,$$
$$act_{ag} \triangleq \{(s, i, \varnothing, action(refine(i, s)), \varnothing) \in S \times I \times \{\varnothing\} \times A \times \{\varnothing\}\}.$$

Таким образом, в основном контексте описание поведения внешней среды совпадает с ее моделью env*, а выбор действия определяется функцией выбора действия агента* $action$*. При этом дополнительный параметр выбора действия не используется.*

Базовые формулы состояния и формулы действия. Для сопоставления некоторого значения из множества $Domain$ некоторому терму вида τ^s в состоянии среды s введем функцию $\lceil\tau^s\rceil : S \to Domain$, определяемую рекурсивно:

- Для символов переменных состояния $v_s \in Var_s$: $\lceil v_s\rceil(s) = \pi_s(s, v_s)$.
- Для функциональных символов $f \in Func_i$: $\lceil f(\tau_1^s, \ldots, \tau_i^s)\rceil(s) = \Phi_i(f)(\lceil\tau_1^s\rceil(s), \ldots, \lceil\tau_i^s\rceil(s))$.

Аналогично для определения значения термов вида τ^a на действии a определим функцию $\lceil\tau^a\rceil : A \to Domain$ следующим образом:

- Для символов переменных действия $v_a \in Var_a$: $\lceil v_a\rceil(a) = \pi_a(a, v_a)$.
- Для функциональных символов $f \in Func_i$: $\lceil f(\tau_1^a, \ldots, \tau_i^a)\rceil(a) = \Phi_i(f)(\lceil\tau_1^a\rceil(a), \ldots, \lceil\tau_i^a\rceil(a))$.

Семантические правила для интерпретации базовых формул состояния определим стандартным образом.

1. $M, (env_c, act_c), (s, i, i_{act}) \models_s P(\tau_1^s, \ldots, \tau_i^s)$ тогда и только тогда, когда $\Pi_i(P)(\lceil\tau_1^s\rceil(s), \ldots, \lceil\tau_i^s\rceil(s)) = 1$.
2. $M, (env_c, act_c), (s, i, i_{act}) \models_s \neg\phi$ тогда и только тогда, когда $M, (env_c, act_c), (s,\ i, i_{act}) \not\models_s \phi$.
3. $M, (env_c, act_c), (s, i, i_{act}) \models_s \phi_1 \wedge \phi_2$ тогда и только тогда, когда $M, env_c, act_c, (s, i, i_{act}) \models_s \phi_1$ и $M, (env_c, act_c), (s, i, i_{act}) \models_s \phi_2$.
4. $M, (env_c, act_c), (s, i, i_{act}) \models_s [\alpha]\varepsilon$ тогда и только тогда, когда для любого $a \in A$, если $M, a \models \alpha$, то для любого $s' \in S$, такого, что $(s, a, s') \in env_c$ выполнено $M, (env_c, act_c), (s', i, i_{act}) \models_s \varepsilon$. [4]

[4]Заметим, что формула ε, согласно грамматике 4.29 не может содержать темпоральных и ментальных операторов, поэтому при её интерпретации допустимо зафиксировать состояние агента.

Аналогичным образом определяются семантические правила и для отношения $\models_a$.

1. $M, a \models_a P(\tau_1^a, \ldots, \tau_i^a)$ тогда и только тогда, когда $\Pi_i(P)(\lceil\tau_1^a\rceil(a), \ \ldots \ , \lceil\tau_i^a\rceil(a)) = 1$.

2. $M, a \models_a \neg\alpha$ тогда и только тогда, когда $M, a \not\models_a \alpha$.

3. $M, a \models_a \alpha_1 \wedge \alpha_2$ тогда и только тогда, когда $M, a \models_a \alpha_1$ и $M, a \models_a \alpha_2$.

Операторы темпоральной логики. Для задания семантики темпоральных операторов определим множество возможных путей в контексте (env_c, act_c) из начальной тройки состояний $(s_0, i_0, i_{0,act}) \in S \times I \times I_{act}$ (определение 12). Возможный путь удовлетворяет трем основным условиям: он начинается в точке $(s_0, i_0, i_{0,act})$, не завершается, если может продолжиться, и каждая следующая тройка определяется предыдущей тройкой, контекстом (env_c, act_c) и функцией обновления состояния агента $refine$.

Определение 12 *Множество возможных путей из начальной тройки состояний* $(s_0, i_0, i_{0,act}) \in S \times I \times I_{act}$ *в контексте* (env_c, act_c) *есть множество цепочек* $out_{(env_c, act_c)}(s_0, i_0, i_{0,act}) \in (S \times I \times I_{act})^+$, *где для каждой цепочки* λ *выполнено* $\lambda[0] = (s_0, i_0, i_{0,act})$ *и для любого* $j < |\lambda|$ *если существуют такие* $a \in A$, $i'_{act} \in I_{act}$ *и* $s \in S$, *что* $(\lambda[j], a, i'_{act}) \in act_c$ *и* $(\lambda[j]|_S, a, s) \in env_c$, *то* $j + 1 < |\lambda|$ *и существует такое* $a' \in A$, *что*

- $(\lambda[j]|_S, a', \lambda[j+1]|_S) \in env_c$.
- $\lambda[j+1]|_I = refine(\lambda[j]|_S, \lambda[j]|_I)$.
- $(\lambda[j], a', \lambda[j+1]|_{I_{act}}) \in act_c$.

Зададим семантические правила для интерпретации операторов динамической и темпоральной логики следующим образом.

5. $M, (env_c, act_c), (s, i, i_{act}) \models_s \mathbf{A}\,\rho$ тогда и только тогда, когда для любого пути $\lambda \in out_{(env_c, act_c)}(s, i, i_{act})$ выполнено $M, (env_c, act_c), \lambda \models_p \rho$.

6. $M, (env_c, act_c), (s, i, i_{act}) \models_s \mathbf{E}\,\rho$ тогда и только тогда, когда существует такой путь $\lambda \in out_{(env_c, act_c)}(s, i, i_{act})$, для которого $M, (env_c, act_c), \lambda \models_p \rho$.

7. $M, (env_c, act_c), \lambda \models_p \bigcirc\phi$ тогда и только тогда, когда выполнено $|\lambda| > 1$ и $M, (env_c, act_c),\ \lambda[1] \models_s \phi$.

8. $M, (env_c, act_c), \lambda \models_p \phi_1\, \mathcal{U}_{\leq t} \phi_2$ тогда и только тогда, когда существует такое $0 \leq i \leq \min(t, |\lambda| - 1)$, что $M, (env_c, act_c), \lambda[i] \models_s \phi_2$ и для любого $0 \leq j < i$ выполнено $M, (env_c, act_c), \lambda[j] \models_s \phi_1$.

Операторы Bel , Des **и** Intend . Семантику оператора Bel ϕ определим через подмену в текущем контексте описания внешней среды env_c на представления агента в текущем его состоянии $i|_{Bel(S,A)}$. Кроме того, учтем возможные вариации текущего состояния внешней среды относительно восприятия этого состояния агентом $see(s)$. В итоге, семантическое правило для интерпретации оператора Bel ϕ выглядит следующим образом.

10. $M, (env_c, act_c), (s, i, i_{act}) \models_s$ Bel ϕ тогда и только тогда, когда для любого $s' \in see(s)$ выполнено $M, (i|_{Bel(S,A)}, act_c), (s', i, i_{act}) \models_s \phi$.

Семантику оператора Des ϕ, в отличие от двух других ментальных операторов, определим не с помощью семантики возможных миров, а методом интерпретируемых символических структур. Оператор "агент хочет, чтобы было выполнено ϕ" интерпретируется как истинный тогда и только тогда, когда формула ϕ принадлежит множеству желаний агента.

11. $M, (env_c, act_c), (s, i, i_{act}) \models_s$ Des ϕ тогда и только тогда, когда $\phi \in i|_{I_{des}}$.

Семантику оператора Intend ϕ определим относительно текущего плана агента $plan = i|_{Plan}$, заместив в контексте описание способа выбора действия на соответствующий способ, предписанный планом следующим образом:

$$act_{plan} = \{(s, i, i_{pln}, a, i'_{pln}) \in S \times I \times I_{pln} \times A \times I_{pln} \mid (see(s), i_{pln}, i'_{pln}, a) \in \sigma_{pln}\}, \quad (4.32)$$

где I_{pln} есть множество внутренних состояний плана $plan$, а $\sigma_{pln} \subseteq P \times I_{pln} \times I_{pln} \times A$ есть отношение переходов плана $plan$. Кроме того, описание поведения внешней среды в текущем контексте заменяется на представления агента, учитываются возможные вариации текущего состояния внешней среды относительно его восприятия агентом, а также в текущей тройке (s, i, i_{act}) третий параметр заменяется на начальное состояние плана $plan$ $i_{pln,0}$. В итоге, семантическое правило для оператора Intend ϕ выглядит следующим образом.

12. $M, (env_c, act_c), (s, i, i_{act}) \models_s$ Intend ϕ тогда и только тогда, когда для любого $s' \in see(s)$ выполнено $M, (i|_{Bel(S,A)}, act_{plan}), (s', i, i_{pln,0}) \models_s \phi$, где $plan = i|_{Plan}$ есть текущий план агента, а $i_{pln,0}$ текущее начальное состояние этого плана.

Семантика со статическим ментальным состоянием

Определение 13 *Семантику логики спецификации интеллектуального агента со статическим ментальным состоянием зададим с помощью трех отношений:*

- *для формул действий* $M, a \models_a \alpha$*;*
- *для формул состояний* $(M, i), env_c, (s, i_{pln}) \models_s \phi$*;*
- *для формул пути* $(M, i), env_c, \lambda \models_p \rho$*;*

где

- $M = \langle ag, Domain, \pi_s, \pi_a, \{\Phi_i\}_{i\in\{0,\ldots,n_f\}}, \{\Pi_i\}_{i\in\{0,\ldots,n_p\}}\rangle$ *есть модель из определения 9;*

- $a \in A$ *есть некоторое действие агента;*

- $i \in I$ *есть некоторое состояние агента;*

- $env_c \subseteq S \times A \times S$ *есть описание поведения внешней среды в текущем контексте;*

- $(s, i_{pln}) \in S \times I_{pln}$ *есть пара состояний, где* $s \in S$ *есть состояние внешней среды, а* $i_{pln} \in I_{pln}$ *есть состояние плана агента;*

- $\lambda \in (S \times I_{pln})^+$ *есть некоторый путь, представленный цепочкой пар вида* $(s, i_{pln}) \in S \times I_{pln}$.

Таким образом, отличием семантики со статическим ментальным состоянием от семантики с динамическим ментальным состояние является (см. раздел 4.3.2) то, что состояние агента включено в постоянную часть отношения вместе с моделью M, а выбор действия агента не зависит от контекста и определяется его планом.

Интерпретация базовых формул состояния и формул действия для семантики со статическим ментальным состоянием аналогична случаю с динамическим ментальным состоянием, описанному в разделе 4.3.2.

Для задания семантики темпоральных операторов необходимо переопределить множество возможных путей, как показано в определении 14. С учетом измененного определения семантические правила для темпоральных операторов задаются аналогичным образом.

Определение 14 *Определим множество возможных путей из начальной пары состояний* $(s_0, i_{0,pln}) \in S \times I_{pln}$ *в контексте* env_c *как множество цепочек* $out_{env_c}(s_0, i_{0,pln}) \in (S \times I_{pln})^+$, *где для каждой цепочки* $\lambda \in out_{env_c}(s, i_{pln})$

выполнено $\lambda[0] = (s_0, i_{0,pln})$ *и для любого* $j < |\lambda|$ *если существуют такие* $s \in S$, $a \in A$ *и* $i'_{pln} \in I_{pln}$, *что* $(\lambda[j]|_S, a, s) \in env_c$ *и* $(see(\lambda[j]|_S), \lambda[j]|_{I_{pln}}, i'_{pln}, a) \in \sigma_{pln}$, *то* $j + 1 < |\lambda|$ *и существует такое* $a' \in A$, *что*

- $(\lambda[j]|_S, a', \lambda[j+1]|_S) \in env_c$.
- $(see(\lambda[j]|_S), \lambda[j]|_{I_{pln}}, \lambda[j+1]|_{I_{pln}}, a') \in \sigma_{pln}$.

Семантические правила для интерпретации операторов Bel ϕ и Des ϕ аналогичны правилам 10 и 11 из раздела 4.3.2, за исключением того, что представления и желания агента извлекаются из статической части, а семантическое правило для оператора Intend ϕ выглядит следующим образом.

12. $(M, i)env_c, (s, i_{pln}) \models_s$ Intend ϕ тогда и только тогда, когда $\phi \in i|_{des}$ и для любого $s' \in see(s)$ выполнено $(M, i), i|_{Bel(S,A)}, (s', i_{pln}) \models_s \phi$.

Таким образом, намерения агента есть подмножество желаний агента, которые, по представлениям агента, будут реализованы текущим планом.

4.3.3. Дополнительные операторы и некоторые тождества

По аналогии с классической логикой высказываний введем дополнительные логические связки следующим образом:

$$\phi_1 \vee \phi_2 \triangleq \neg(\neg\phi_1 \wedge \neg\phi_2), \tag{4.33}$$

$$\phi_1 \Rightarrow \phi_2 \triangleq \neg\phi_1 \vee \phi_2, \tag{4.34}$$

$$\phi_1 \Leftrightarrow \phi_2 \triangleq (\phi_1 \wedge \phi_2) \vee (\neg\phi_2 \wedge \neg\phi_1). \tag{4.35}$$

Важным производным оператором темпоральной логики является *слабый* вариант оператора $\psi_1 \mathcal{U}_{\leq t} \psi_2$, обозначаемый $\psi_1 \mathcal{W}_{\leq t} \psi_2$ (weak until) и имеющий

следующую интуитивную интерпретацию: “не более чем через t шагов (когда-нибудь, если $t = \infty$) будет выполнено свойство ψ_2, а до этого момента *включительно* будет выполнено свойство ψ_1, или свойство ψ_1 будет выполнено в течении t шагов (всегда, если $t = \infty$)”. Таким образом, слабый оператор $\mathcal{W}_{\leq t}$ допускает то, что свойство ψ_2 не будет выполнено, если на соответствующем временном интервале будет выполнено свойство ψ_1. Оператор $\mathcal{W}_{\leq t}$ может быть выражен следующим образом:

$$\mathbf{A}\,\phi_1\,\mathcal{W}_{\leq t}\phi_2 \triangleq \neg\mathbf{E}\,\neg\phi_2\,\mathcal{U}_{\leq t}\neg\phi_1, \tag{4.36}$$

$$\mathbf{E}\,\phi_1\,\mathcal{W}_{\leq t}\phi_2 \triangleq \neg\mathbf{A}\,\neg\phi_2\,\mathcal{U}_{\leq t}\neg\phi_1. \tag{4.37}$$

Кроме того, можно определить стандартные дополнительные темпоральные операторы “не позднее чем через t шагов (когда-нибудь, если $t = \infty$) ϕ” ($\mathcal{F}\,\phi$) и “в течение t шагов (всегда, если $t = \infty$) ϕ” ($G\,\phi$):

$$\mathbf{A}\,\mathcal{F}_{\leq t}\phi \triangleq \mathbf{A}\,true\,\mathcal{U}_{\leq t}\phi, \tag{4.38}$$

$$\mathbf{E}\,\mathcal{F}_{\leq t}\phi \triangleq \mathbf{E}\,true\,\mathcal{U}_{\leq t}\phi, \tag{4.39}$$

$$\mathbf{A}\,G_{\leq t}\phi \triangleq \mathbf{A}\,\phi\,\mathcal{W}_{\leq t}false, \tag{4.40}$$

$$\mathbf{E}\,G_{\leq t}\phi \triangleq \mathbf{E}\,\phi\,\mathcal{W}_{\leq t}false. \tag{4.41}$$

Для всех операторов и кванторов логики определим следующий приоритет (в порядке убывания): $\neg$, $\bigcirc$, Bel , Des , Intend $\wedge$, $\vee$, $\mathbf{A}$, $\mathbf{E}$, G, $\mathcal{F}$, $\mathcal{U}$, $\mathcal{W}$, $\Rightarrow$, $\Leftrightarrow$.

Формулу ϕ будем называть *общезначимой* тогда и только тогда, когда для любых модели M, контекста (env, act) и тройки состояний (s, i, i_{act}) выполнено $M, (env, act), (s, i, i_{act}) \models \phi$. Факт общезначимости формулы ϕ будем обозначать как $\vdash \phi$.

Одним из основных общезначимых формул логики спецификации интеллектуального агента являются следующие формулы, выражающие правила двой-

ственности:

$$\vdash \neg\mathbf{A} \bigcirc \phi \Leftrightarrow (\mathbf{E} \bigcirc \neg\phi \vee \neg\mathbf{E} \bigcirc true), \tag{4.42}$$

$$\vdash \neg\mathbf{E} \bigcirc \phi \Leftrightarrow (\mathbf{A} \bigcirc \neg\phi \vee \neg\mathbf{E} \bigcirc true), \tag{4.43}$$

$$\vdash \neg\mathbf{A}\, \phi_1\, \mathcal{U}_{\leq t} \phi_2 \Leftrightarrow \mathbf{E}\, \neg\phi_2\, \mathcal{W}_{\leq t} \neg\phi_1, \tag{4.44}$$

$$\vdash \neg\mathbf{E}\, \phi_1\, \mathcal{U}_{\leq t} \phi_2 \Leftrightarrow \mathbf{A}\, \neg\phi_2\, \mathcal{W}_{\leq t} \neg\phi_1, \tag{4.45}$$

$$\vdash \neg\mathbf{A}\, \phi_1\, \mathcal{W}_{\leq t} \phi_2 \Leftrightarrow \mathbf{E}\, \neg\phi_2\, \mathcal{U}_{\leq t} \neg\phi_1, \tag{4.46}$$

$$\vdash \neg\mathbf{E}\, \phi_1\, \mathcal{W}_{\leq t} \phi_2 \Leftrightarrow \mathbf{A}\, \neg\phi_2\, \mathcal{U}_{\leq t} \neg\phi_1, \tag{4.47}$$

$$\vdash \neg\mathbf{A}\, \mathcal{F}_{\leq t} \phi \Leftrightarrow \mathbf{E}\, G_{\leq t} \neg\phi, \tag{4.48}$$

$$\vdash \neg\mathbf{E}\, \mathcal{F}_{\leq t} \phi \Leftrightarrow \mathbf{A}\, G_{\leq t} \neg\phi, \tag{4.49}$$

$$\vdash \neg\mathbf{A}\, G_{\leq t} \phi \Leftrightarrow \mathbf{E}\, \mathcal{F}_{\leq t} \neg\phi, \tag{4.50}$$

$$\vdash \neg\mathbf{E}\, G_{\leq t} \phi \Leftrightarrow \mathbf{A}\, \mathcal{F}_{\leq t} \neg\phi. \tag{4.51}$$

Общезначимость формул 4.42 и 4.43 легко доказывается методом “от противного” через анализ семантических правил для кванторов пути и оператора “в следующий момент”, а общезначимость формул 4.44–4.51 очевидным образом следует из определения дополнительных операторов 4.36–4.41.

Другой важный класс общезначимых формул составляют правила *рекурсивного вычисления* 4.52–4.55. Эти правила являются стандартными для CTL-подобных логик и их доказательства, применимые и к предложенной логике, можно найти в [56].

$$\vdash \mathbf{A}\, \phi_1\, \mathcal{U}_{\leq t} \phi_2 \Leftrightarrow \phi_2 \vee (\phi_1 \wedge \mathbf{A} \bigcirc \mathbf{A}\, \phi_1\, \mathcal{U}_{\leq t-1} \phi_2), \tag{4.52}$$

$$\vdash \mathbf{E}\, \phi_1\, \mathcal{U}_{\leq t} \phi_2 \Leftrightarrow \phi_2 \vee (\phi_1 \wedge \mathbf{E} \bigcirc \mathbf{E}\, \phi_1\, \mathcal{U}_{\leq t-1} \phi_2), \tag{4.53}$$

$$\vdash \mathbf{A}\, \phi_1\, \mathcal{W}_{\leq t} \phi_2 \Leftrightarrow \phi_1 \wedge (\phi_2 \vee \mathbf{A} \bigcirc \mathbf{A}\, \phi_1\, \mathcal{W}_{\leq t-1} \phi_2), \tag{4.54}$$

$$\vdash \mathbf{E}\, \phi_1\, \mathcal{W}_{\leq t} \phi_2 \Leftrightarrow \phi_1 \wedge (\phi_2 \vee \mathbf{E} \bigcirc \mathbf{E}\, \phi_1\, \mathcal{W}_{\leq t-1} \phi_2). \tag{4.55}$$

Глава 5

Алгоритмы верификации для логики спецификации интеллектуального агента

5.1. Символические алгоритмы

Одной из наиболее серьезных проблем, затрудняющих внедрение формальной верификации, а также многих других методов, является проблема *комбинаторного взрыва пространства состояний*. Например, для системы, состояние которой включает всего 40 двоичных флагов, полное количество состояний будет превышать тысячу миллиардов, кроме того, этот размер будет расти экспоненциально с добавлением каждого нового флага. В тоже время, для верификации такой системы нужно некоторым образом обработать все её состояния.

Частично устранить эту проблему позволяют *символические алгоритмы* [42], оперирующие не отдельными состояниями, а их множествами. Реализация таких алгоритмов стала возможна после появления в 1986 году упорядоченных бинарных разрешающих диаграмм (Ordered Binary Decision Diagrams, $OBDD$) [41]. Сразу после своего появления $OBDD$ привлекли внимание научного сообщества и были применены, в том числе, для реализации символических алгоритмов верификации [48, 122].

$OBDD$ показали отличные результаты для систем, состояния которых легко представимы в виде набора двоичных флагов, однако для систем, содержащих целочисленные параметры, они не всегда оказывались удобны. Для устранения этого недостатка были предложены алгебраические разрешающие диаграммы (Algebraic Decision Diagrams, ADD) [18], в некоторых работах именуемые многозначными разрешающими диаграммами (Multi-Terminal Binary Decision Diagrams, $MTBDD$) [60].

Несмотря на то, что ADD предоставили удобный способ для представления систем с целочисленными параметрами в состоянии, в случае, если система умеет выполнять сложение и/или умножение, размер представления такой системы будет экспоненциально расти. Для устранения этой проблемы так же было предложено много разнообразных подходов, начиная от модификации концепции разрешающих диаграмм, например, аффинные алгебраические разрешающие диаграммы (Affine Algebraic Decision Diagrams, $AADD$) [159], вплоть до принципиально новых подходов, например, аффинное представление данных [10].

5.1.1. $OBDD$

Концепция бинарных разрешающих диаграмм (Binary Decision Diagrams, BDD) появилась еще в середине прошлого века [17, 110], однако не нашла широкого применения на практике. Только в 1986 году после публикации работы Рэндала Брайанта (Randal E. Bryant) [41], предложившего относительно простое усовершенствование концепции (фиксированный порядок переменных), а также описавшего возможные операции с этими структурами, $OBDD$ начали внедряться на практике.

Основные понятия

Изначально и BDD, и $OBDD$ предназначались для представления булевых функций и операций с ними. Пусть $\mathbb{B} = \{0, 1\}$, тогда функция вида $f : \mathbb{B}^n \times \mathbb{B}$ есть булева функция n переменных. Для таких функций можно построить расширение Шэнона (Shannon expansion) [162]:

$$f(x_1, \ldots, x_i, \ldots, x_n) = x_i \wedge f(x_1, \ldots, 1, \ldots, x_n) \vee \neg x_i \wedge f(x_1, \ldots, 0, \ldots, x_n), \quad (5.1)$$

где $1 \leq i \leq n$.

Формально, упорядоченная бинарная разрешающая диаграмма есть ориентированный корневой ациклический граф с множеством вершин $V = N \cup T$, где

$N \cap T = \emptyset$. Вершины множества N являются *нетерминалами*, и для каждой такой вершины $v \in N$ определено *значение порядка* $index(v) \in \{1, \ldots, n\}$ и ровно две дочерних вершины $low(v), high(v) \in V$. Вершины множества T являются *терминалами*. Для таких вершин $v \in T$ определено значение $value(v) \in \mathbb{B}$, а дочерних вершин у нее нет. Кроме того, выполнено *условие порядка*: для любого нетерминала $v \in N$ и любой его дочерней вершины v' выполнено либо $v' \in T$, либо $index(v) < index(v')$.

Граф V (здесь и далее в этой главе будем отождествлять граф с множеством его вершин) с корневой вершиной $v \in V$ рекурсивно порождает булеву функцию n-переменных $f_v : \mathbb{B}^n \times \mathbb{B}$ следующим образом:

- если $v \in T$, то $f_v(x_1, \ldots, x_n) \equiv value(v)$;
- если $v \in N$, и $index(v) = i$ то $f_v(x_1, \ldots, x_n) = x_i \wedge f_{high(v)}(x_1, \ldots, x_n) \vee \neg x_i \wedge f_{low(v)}(x_1, \ldots, x_n)$.

Граф V *изоморфен* графу V' тогда и только тогда, когда существует такое отображение $\sigma : V \in V'$, что выполнены следующие свойства:

- если $v \in T$, то $\sigma(v) \in T'$ и $value(v) = value(\sigma(v))$;
- если $v \in V$, то $\sigma(v) \in V'$ и выполнено $index(v) = index(\sigma(v))$, $\sigma(low(v)) = low(\sigma(v))$ и $\sigma(high(v)) = high(\sigma(v))$.

Граф V является *приведенным*, если в нем не существует ни одной вершины $v \in V$, для которой выполнено *хотя бы одно* из следующих условий:

- $v \in T$ и существует такая $v' \in T$, что $value(v) = value(v')$;
- $v \in V$ и $low(v) = high(v)$;
- $v \in V$ и существует такая $v' \in V$, что под-графы с корнем в v и с корнем в v' изоморфны.

В [41] доказывается, что приведенные упорядоченные бинарные разрешающие диаграммы являются каноническим представлением для булевых функций, т.е. каждой функции соответствует ровно один приведённый граф V с точностью до изоморфизма. Далее под $OBDD$ будем понимать именно приведенные графы, если явно не указано обратное.

Операции с $OBDD$

Пусть V и V' есть $OBDD$ с корнями $v \in V$ и $v' \in V'$, представляющие функции f_v и $f_{v'}$ соответственно. Граф, представляющий функцию $f_v \ op \ f_{v'}$, где op есть некоторая логическая операция, можно построить рекурсивно следующим образом.

- Если $v \in T$ и $v' \in T'$, то граф для $f_v \ op \ f_{v'}$ состоит из одной терминальной вершины u, такой, что $value(u) = value(v) \ op \ value(v')$.
- Если $v \in N$, $v' \in N'$ и $index(v) = index(v') = i$, то корневой вершиной результирующего графа будет вершина u, такая, что $index(u) = i$, $low(u)$ является корнем графа, представляющего $f_{low(v)} \ op \ f_{low(v')}$, а $high(u)$ — корнем графа для $f_{high(v)} \ op \ f_{high(v')}$.
- Если $v \in N$, а $v' \in T'$ или $v' \in V'$ и $i = index(v) < index(v')$, то корневой вершиной результирующего графа будет вершина u, такая, что $index(u) = i$, $low(u)$ является корнем графа, представляющего $f_{low(v)} \ op \ f_{v'}$, а $high(u)$ — корнем графа для $f_{high(v)} \ op \ f_{v'}$. Аналогично для случая $v' \in V'$, а $v \in T$ или $v \in V$ и $index(v) < index(v')$.

Сложность описанного выше алгоритма не превышает $O(|V| \cdot |V'|)$, однако полученный граф может не быть приведенным. В [41] предлагается алгоритм построения приведенного графа класса $O(|V| \cdot \log(|V|)$, однако позднее были

предложены линейные алгоритмы решения этой задачи. В результате, построение приведенного графа, представляющего функцию f_v *op* $f_{v'}$ занимает времени не более, чем $O(|V| \cdot |V'|)$.

Естественным образом можно обобщить предложенный алгоритм и на случай более сложных операций с булевыми функциями, например, часто используется операция if f_v then $f_{v'}$ else $f_{v''}$. Сложность построения соответствующего графа не превышает $O(|V| \cdot |V'| \cdot |V''|)$.

Часто используется и операция подстановки значения $f(x_1, \ldots, x_n)|_{x_i=c}$ (где $c \in \mathbb{B}$). Реализовать её можно простым обходом графа и замещением каждой нетерминальной вершины $v \in N$, такой, что $index(v) = i$, на $low(v)$ или $high(v)$ с последующим редуцированием графа, в результате чего в графе не останется нетерминальных вершин с индексом i. Примером более сложной операцией является операция *композиции функций*: $f(x_1, \ldots, x_n)|_{x_i=g(y)}$, вычисляемая как if $g(y)$ then $f(x_1, \ldots, x_n)|_{x_i=1}$ else $f(x_1, \ldots, x_n)|_{x_i=0}$.

Представление векторных функций и множеств

Естественным образом концепцию $OBDD$ можно расширить для представления функций вида $f : \mathbb{B}^n \to \mathbb{B}^m$. Такую функцию можно рассматривать как вектор из m обычных булевых функций $(f_1, \ldots, f_m)$, где каждая $f_i : \mathbb{B}^n \to \mathbb{B}$ может быть представлена своим графом V.

Однако, как показывает практика, такое представление часто бывает избыточным, так как разные функции могут иметь схожее строение графов или их частей. Для устранения этого недостатка была предложена концепция *разделяемых* (shared) $OBDD$ [83]. Основная идея этого метода заключается в том, что вектор функций представляется не набором независимых графов, а одним графом с несколькими корневыми вершинами (по одной для каждой функции). Такое представление позволяет описывать схожие фрагменты единожды и затем ссылаться на них из других частей графа. В работе [86] этот же прием

использовался для оптимизации представления простых булевых функций — функция разбивалась на комбинацию более простых функций, которые в совокупности представлялись разделяемым $OBDD$.

Технику $OBDD$ с успехом можно применять для представления не только булевых, но и других дискретных функций. Рассмотрим, например, функцию вида $f : \{0, \ldots, 2^{16} - 1\}^2 \to \{0, \ldots, 2^{16} - 1\}$. Это функция сопоставляет двум натуральным числам, не превышающим $2^{16} - 1$, одно такое же число. Такая функция может быть рассмотрена как функция вида $f' : \mathbb{B}^{32} \to \mathbb{B}^{16}$, оперирующая двоичными 16-битными представлениями чисел, а уже для представления этой функции можно построить разделяемый $OBDD$. Заметим, что такое представление функций является естественным для большинства современных ЭВМ, так как они работают именно в двоичной системе исчисления. Кроме того, заметим, что для любой дискретной конечной функции можно построить двоичное представление и, следовательно, $OBDD$.

Вполне естественным является и использование булевых функций для представление множеств. Для любого конечного множества S можно построить сюръективное отображение $S_{\mathbb{B}} : \mathbb{B}^n \to S$ для некоторого n. При этом некоторому подмножеству $S' \subseteq S$ будет соответствовать некоторое подмножество векторов в $S'_{\mathbb{B}} \subseteq \mathbb{B}^n$. А уже это множество векторов может быть представлено в виде булевой функции $f_{S'_{\mathbb{B}}} : \mathbb{B}^n \to \mathbb{B}$, такой что выполнены следующие свойства:

- $f_{S'_{\mathbb{B}}}(x_1, \ldots, x_n) = 1$ тогда и только тогда, когда $(x_1, \ldots, x_n) \in S'_{\mathbb{B}}$, а, следовательно и $S^{-1}_{\mathbb{B}}(x_1, \ldots, x_n) \in S'$;
- $f_{S'_{\mathbb{B}}}(x_1, \ldots, x_n) = 0$ тогда и только тогда, когда $(x_1, \ldots, x_n) \notin S'_{\mathbb{B}}$, а, следовательно и $S^{-1}_{\mathbb{B}}(x_1, \ldots, x_n) \notin S'$.

При этом операциям со множествами будут соответствовать операции с булевыми функциями, например: $P \cup Q$ эквивалентно $f_{P_{\mathbb{B}}} \vee f_{Q_{\mathbb{B}}}$, $P \cap Q$ эквивалентно

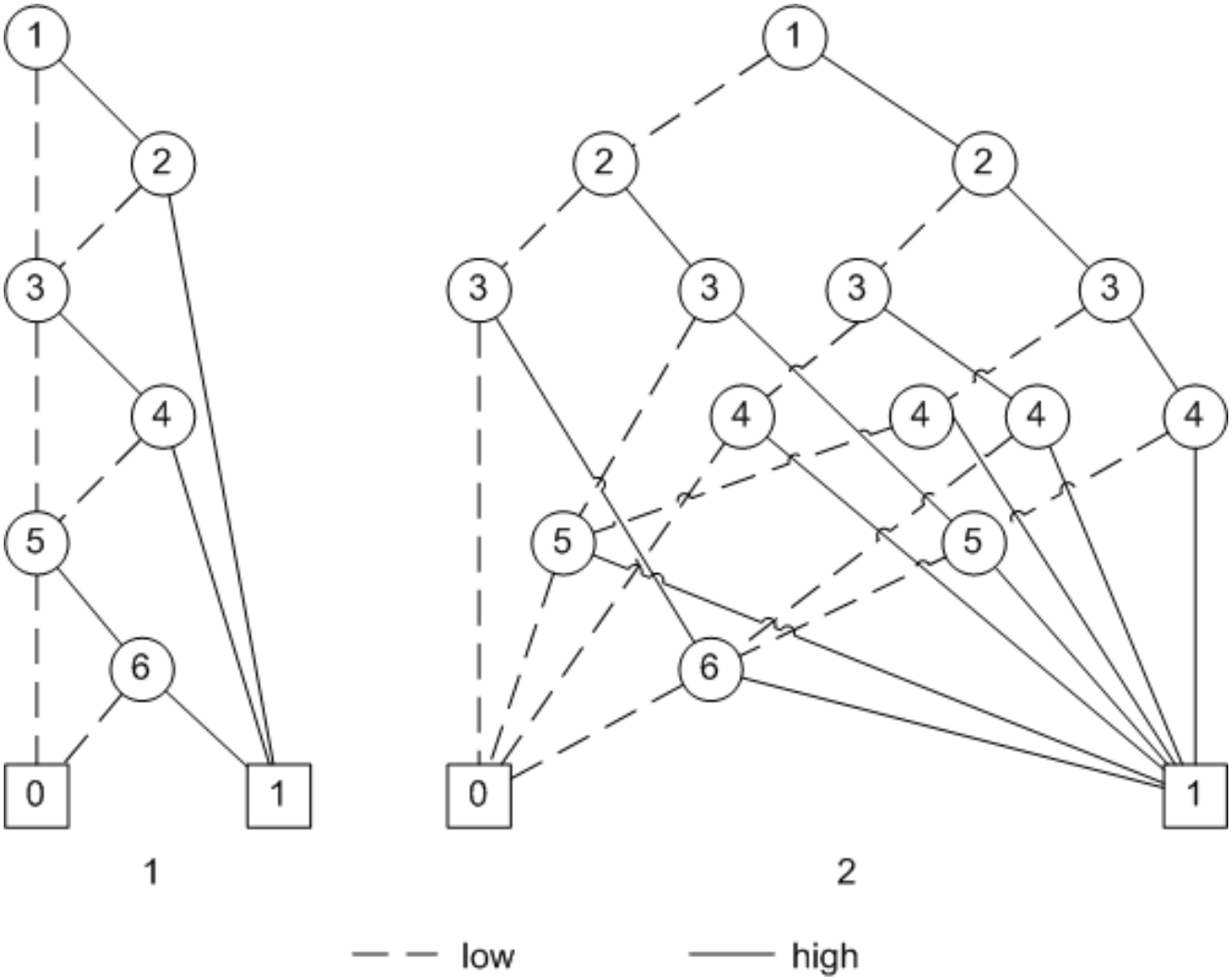

Рис. 5.1. *OBDD* для $x_1 \wedge x_2 \vee x_3 \wedge x_4 \vee x_5 \wedge x_6$ (1) и $x_1 \wedge x_4 \vee x_2 \wedge x_5 \vee x_3 \wedge x_6$ (2).

$f_{P_{\mathbb{B}}} \wedge f_{Q_{\mathbb{B}}}$, $P \setminus Q$ эквивалентно $f_{P_{\mathbb{B}}} \wedge \neg f_{Q_{\mathbb{B}}}$ и т.д. Следовательно, *OBDD* можно эффективно использовать для представления множеств и операций над ними.

О значении порядка переменных

Рассмотрим две булевы функции: $x_1 \wedge x_2 \vee x_3 \wedge x_4 \vee x_5 \wedge x_6$ и $x_1 \wedge x_4 \vee x_2 \wedge x_5 \vee x_3 \wedge x_6$. *OBDD* для этих функций представлены на рисунке 5.1. При этом наглядно видно, что в первом случае размер графа растет линейно при добавлении новых переменных, тогда как во втором — экспоненциально.

Это проблема наглядно иллюстрирует сильные и слабые стороны использования *OBDD* — с помощью правильного выбора порядка переменных можно

добиться того, что размер представления функции будет полиномиально зависеть от количества параметров, но в общем случае эта зависимость экспоненциальная. Задача поиска оптимального порядка является NP-полной, к тому же, полиномиальный порядок не всегда возможен. Тем не менее, для большинства *упорядоченных* структур, зная особенности их строения, можно относительно легко угадать оптимальное порядок.

Например, при представлении в виде $OBDD$ отношения переходов конечного автомата (см. раздел 2.1.1) $\sigma \subseteq S \times I \times S$ (где S есть множество состояний, а I — входной алфавит) в двоичном виде первой на ум приходит кодировка вида $(s_1, \ldots, s_n, i_1, \ldots, i_m, s'_1, \ldots, s'_n) \in \mathbb{B}^{2n+m}$. Однако, как правило, близким к оптимальному оказывается порядок переменных вида $(i_1, \ldots, i_m, s_1, s'_1, \ldots, s_n, s'_n)$, а наиболее очевидный порядок ведет к экспоненциальному росту размера графа [30, 84].

5.1.2. ADD

Одним из широко используемых обобщений концепции упорядоченных бинарных разрешающих диаграмм $OBDD$ является концепция алгебраических разрешающих диаграмм ADD [18]. ADD используются для представления булевых функций в многозначной алгебре Буля, когда $\mathbb{B} = \{0, 1, \ldots, n\}$ (а не $\mathbb{B} = \{0, 1\}$). Для таких функций, так же как и для обычных булевых функций, выполнена теорема о расширении Шэнона (см. уравнение 5.1). В результате, для представления таких функций могут быть использованы графы, аналогичные $OBDD$, с единственной модификацией: для любой терминальной вершины $v \in T$ определено значение $value(v) \in \{0, 1, \ldots, n\}$ (а не $value(v) \in \{0, 1\}$).

Таким образом, единственное видимое отличие ADD от $OBDD$ является то, что ADD допускает наличие терминальных вершин, отличных от 0 и 1. Именно поэтому такие графы часто называют многозначными разрешающими диаграммами (Multi-Terminal Binary Decision Diagrams, $MTBDD$) [60].

Изначально было предложено использовать ADD для представления матриц и операций с ними [18, 60], а также для решения задачи поиска кратчайшего пути в графе [18]. Позднее ADD стали активно применять для представления стохастических систем [134], в том числе марковских цепей и марковских процессов [33], а также для реализации верификации таких систем [135].

5.1.3. Символическая верификация CTL

Одной из наиболее обширных областей применения $OBDD$ являются *символические* алгоритмы верификации [48, 122, 132]. В отличие от классических алгоритмов, оперирующих отдельными состояниями верифицируемой системы, символические алгоритмы оперируют *множествами* состояний, используя для их представления $OBDD$.

Рассмотрим, например, задачу верификации для логики CTL (см. раздел 4.1.3). Модель логики CTL содержит следующие элементы.

- Конечное множество состояний S, для которого можно построить двоичную кодировку $S_{\mathbb{B}} : S \to \mathbb{B}^n$.
- Отношение переходов $R \subseteq S \times S$, которое может быть представлено как булева функция $2n$ переменных и, следовательно, для него можно построить $OBDD$
- Оценочная функция для пропозициональных символов $\pi : S \times 2^{Prop}$, которую можно рассматривать как отношение $\pi : S \times Prop$ и естественным образом представить в виде булевой функции $f_\pi : \mathbb{B}^n \times \mathbb{B}^{|Prop|} \to \mathbb{B}^1$.

При решении задачи верификации для формулы ϕ нужно найти такое подмножество состояний $[\phi] \subseteq S$, для которых эта формула выполнена. Для представления таких множеств и операций с ними естественно использовать $OBDD$.

[1] Такая кодировка, естественно не является эффективной, но является наиболее наглядной.

Для простоты изложения можно считать, что все используемые $OBDD$ представляют функции $2n + |Prop|$, но не все функции зависят от всех переменных (например, множество $[\phi]$ представляется функцией, зависимой только от первых n переменных). В результате полученные $OBDD$ не будут содержать нетерминальных вершин, соответствующих неиспользуемым в данный момент переменным.

Алгоритм 15 (Символическая верификация CTL)
switch ϕ
  case $\phi = p$: $[\phi] := \pi^{-1}(p)$;
  case $\phi = \neg\psi$: $[\phi] := S \setminus [\psi]$;
  case $\phi = \psi_1 \wedge \psi_2$: $[\phi] := [\psi_1] \cap [\psi_2]$;
  case $\phi = \mathbf{E} \bigcirc \psi$: $[\phi] := Pre_{\mathbf{E}}([\psi])$;
  case $\phi = \mathbf{A} \bigcirc \psi$: $[\phi] := Pre_{\mathbf{A}}([\psi])$;
  case $\phi = \mathbf{E}\,\psi_1\,\mathcal{U}\,\psi_2$:
    $x := \emptyset$; $y := [\psi_2]$***;***
    while $y \not\subseteq x$ ***do*** $x := x \cup y$; $y := Pre_{\mathbf{E}}(x) \cap [\psi_1]$; ***od***;
    $[\phi] := x$;
  case $\phi = \mathbf{A}\,\psi_1\,\mathcal{U}\,\psi_2$:
    $x := \emptyset$; $y := [\psi_2]$***;***
    while $y \not\subseteq x$ ***do*** $x := x \cup y$; $y := Pre_{\mathbf{A}}(x) \cap [\psi_1]$; ***od***;
    $[\phi] := x$;
end switch

Основная идея алгоритма верификации (см. Алгоритм 15) заключается в рекурсивном спуске по структуре формулы, когда для составной формулы сначала решается задача для формул-частей, а затем из полученных результатов формируется результат для всей формулы.

Для случая, если формула состоит из одного пропозиционального символа ($\phi = p$), множество состояний, на которых она выполнена, определяется как $\pi^{-1}(p)$. С точки зрения операций с $OBDD$ это вычисление можно смоделировать как подстановку в функцию f_π фиксированного вектора значений, что в результате даст функцию, зависящую только от состояния системы (в его двоичном представлении) и представляющую множество $\pi^{-1}(p)$.

Для обработки логических операторов $\neg\psi$ и $\psi_1 \wedge \psi_2$ можно использовать простые операции со множествами: дополнение $S \setminus [\psi]$ (где множество S представлено графом с одной терминальной вершиной 1) и пересечение $[\psi_1] \cap [\psi_2]$.

При построении множества для $\mathbf{E} \bigcirc \psi$ выбираются только те состояния, для которых существует возможность перейти в одно из состояний множества $[\psi]$. Для вычисления этого множества используется функция $Pre_{\mathbf{E}} : 2^S \to 2^S$, которую можно вычислить следующим образом:

$$Pre_{\mathbf{E}}(x) \triangleq \{s \in S \mid \exists s' \in x : (s, s') \in R\}. \tag{5.2}$$

С точки зрения операций с $OBDD$ реализация функции $Pre(x)$ включает следующие шаги.

1. В представлении x все переменные, представляющие текущее состояние $s_1, \ldots, s_n$, переименовать в переменные, представляющее следующее состояние $s'_1, \ldots, s'_n$. Т.е. переопределить все значения $index$ для всех нетерминальных вершин соответствующим образом. Полученное множество x' можно рассматривать как множество $S \times x'$.

2. Вычислить представления множества $R \cap (S \times x')$.

3. В представлении множества $R \cap (S \times x')$, каждую нетерминальную вершину $v \in V$, соответствующую кодировке следующего состояния, заменить на корневую вершину графа $low(v) \vee high(v)$.

Для вычисления $\mathbf{A} \bigcirc \psi$ используется функция $Pre_{\mathbf{A}}(x) : 2^S \to 2^S$, которую можно выразить следующим образом:

$$Pre_{\mathbf{A}}(x) \triangleq Pre_{\mathbf{E}}(x) \cap (S \setminus Pre_{\mathbf{E}}(S \setminus x)). \tag{5.3}$$

Результатом функции $Pre_{\mathbf{A}}(x)$ является множество таких состояний y, из которых можно попасть в одно из состояний множества x и нельзя выйти за переделы множества x.

При обработке формулы $\mathbf{E}\,\psi_1\,\mathcal{U}\,\psi_2$ ищется неподвижная точка оператора $x = x \cup (Pre_{\mathbf{E}}(x) \cap [\psi_1]$, начиная с $x = [\psi_2]$. Так как этот оператор является монотонным (множество x только растет), эта неподвижная точка будет найдена не более чем за $|S|$ итераций. Аналогичным образом обрабатывается и формула вида $\mathbf{A}\,\psi_1\,\mathcal{U}\,\psi_2$, используя функцию $Pre_{\mathbf{A}}(x)$.

5.1.4. Пакеты $OBDD$

Концепция разрешающих диаграмм достаточно проста и понятна, однако реализация алгоритмов на основе $OBDD$ сопряжена со множеством технических трудностей. К счастью, существует большое количество библиотек и пакетов для различных платформ и языков, реализующих наиболее распространенные операции с $OBDD$ и ADD. Работа [90] содержит сравнительный анализ 13 из наиболее распространенных пакетов. В данном разделе мы кратко рассмотрим основные из них.

Пакет Colorado University Decision Diagrams, $CUDD$ [166], является, пожалуй, наиболее распространенным из свободно распространяемых пакетов. Как и большинство подобных решений, $CUDD$ написан на языке C для компилятора GCC и поддерживает такие полезные возможности, как динамическое изменение порядка переменных, кэширование узлов и сборку мусора. Основным достоинством этого пакета является то, что, помимо классических $OBDD$, он поддерживает операции с ADD.

Другим широко распространенным пакетом является пакет $BuDDy$ [116]. Хорошее сравнение $BuDDy$ с $CUDD$ можно найти в работе [94]. Так же как и $CUDD$, $BuDDy$ написан на C и поддерживает весь набор операций с $OBDD$, однако $BuDDy$ не поддерживает работу с ADD.

Интересная реализация операций с $OBDD$ для платформы $Java$ предложена в работе [115]. В отличие от большинства других работ, предлагающих набор функций или классов для работы с $OBDD$, в [115] предложено расшире-

ние языка, получившее название $Jedd$.

Так как $OBDD$ по сути является графом, основным элементом его представления в пакетах $OBDD$ является вершина графа. При этом большинство пакетов описывают дуги графа как два *указателя* на $low(v)$ и $high(v)$. Такой подход является наиболее простым в реализации, однако он обладает существенным недостатком — при переходе к работе с более большими объемами памяти (более 4-х гигабайт) необходимо увеличение размера указателя с обычных 32-х бит до 64-х, что ведет к увеличению размера представления $OBDD$ практически в два раза. Альтернативным подходом, примененным в пакетах $ABCD$ [34] и Pointerless BDD [89], является использование индексов. Пакет Pointerless BDD по скорости работы и экономичности расхода памяти превосходит $CUDD$ почти в два раза, однако объем реализованной в Pointerless BDD функциональности пока значительно уступает $CUDD$.

Достаточно активно развивается и направление *параллельных* $OBDD$, представленного пакетами $BDDNOW$ [125] и $MORE$ [74]. Основная отличительная черта этого направления заключается в том, что операции над $OBDD$ выполняются параллельно сетью вычислительных машин, что позволяет значительно увеличить размер задач, поддающихся обработке.

5.2. Алгоритмы верификации

Задачу верификации для логики спецификации интеллектуального агента определим следующим образом: по заданной модели M, внутреннему состоянию агента $i \in I$ и формуле ϕ определить такое множество состояний внешней среды $[\phi] \subseteq S$, что для любого состояния $s \in [\phi]$ выполнено $(M, i), env, (s, i|_{0,pln}) \models \phi$, где $i|_{0,pln}$ есть начальное состояние плана агента $i|_{plan}$, а env есть описание поведения внешней среды.

В данном разделе предлагаются символические алгоритмы для решения за-

дачи в этой формулировке. В приложении А.1 приведены доказательства полноты и корректности предложенных алгоритмов, а также анализ их алгоритмической сложности.

5.2.1. Представление входных данных

Первая проблема, которую нужно решить при построении символических алгоритмов верификации является проблема символического представления входных данных. В нашем случае этими данными являются части модели $M = \langle ag, Domain, \pi_s, \pi_a, \{\Phi_i\}_{i\in\{0,\ldots,n_f\}}, \{\Pi_i\}_{i\in\{0,\ldots,n_p\}}\rangle$.

Символическое представление состояния агента

Основной составляющей модели M является модель интеллектуального агента $ag = (S, A, env, see, I_B, bel, brf, I_D, des, drf, plan, prf)$. Для *конечных* множеств состояний S и действий A существуют минимальные n_S и n_A такие, что $|S| \leq 2^{n_S}$ и $|A| \leq 2^{n_A}$. Следовательно, для этих множеств можно построить двоичные кодировки $S_{\mathbb{B}} : S \to \mathbb{B}^{n_S}$ и $A_{\mathbb{B}} : A \to \mathbb{B}^{n_A}$.

Функцию описания внешней среды $env : S \times A \to 2^S$ можно представить в виде отношения $env \subseteq S \times A \times S$. Само это отношение можно представить в виде функции $env_{\mathbb{B}} : \mathbb{B}^{n_S+n_A+n_S} \to \mathbb{B}$, которая, в свою очередь представляется в виде одного $OBDD$ от $n_S + n_A + n_S$ переменных.

Отношение восприятия $see \in S \times S$ моделируется функцией $see_{\mathbb{B}} : \mathbb{B}^{n_S+n_S} \to \mathbb{B}$, которую можно представить в виде $OBDD$ от $n_S + n_S$ переменных. Однако, если ограничиться только корректными восприятиями (т.е. предположить, что отношение see является отношением эквивалентности), то это отношение можно рассматривать как функцию $see : S \to P$ (где P есть множество классов эквивалентности). Поскольку множество S является конечным, то и множество классов эквивалентности P тоже будет конечным. Следовательно, существует

минимальное n_P такое, что $|P| \leq 2^{n_P}$. В этом случае, отношение see можно описать как функцию $see_{\mathbb{B}} : \mathbb{B}^{n_S} \to \mathbb{B}^{n_P}$, представленную разделяемой $OBDD$ из n_P компонентов от n_S переменных. Заметим, что от представления в виде функции $\mathbb{B}^{n_S} \to \mathbb{B}^{n_P}$ легко можно перейти к представлению в виде отношения $\mathbb{B}^{n_S+n_S} \to \mathbb{B}$, вычислив следующую функцию: $\bigcap_{i \in \{1,\ldots,n_P\}} ((see^i_{\mathbb{B}} \times S) \cap (S \times see^i_{\mathbb{B}}))$, где $see^i_{\mathbb{B}}$ есть i-ая компонента в представлении $see_{\mathbb{B}}$.

Представления агента $i|_{bel} \subseteq S \times A \times S$, аналогично описанию внешней среды env, могут быть представлены в виде одного $OBDD$ от $n_S + n_A + n_S$ переменных. Желания агента, являющиеся множеством формул $i|_{des} \subseteq L_\phi$, логичнее представлять с использованием списка или хэш-таблицы.

Для построения символического представления отношения переходов σ плана агента $i|_{plan}$ введем ограничение на максимальный размер множества состояний плана: $|I_{pln}| < 2^{n_{pln}}$. В этом случае отношение переходов $\sigma \subseteq P \times I_{pln} \times I_{pln} \times A$ может быть представлено одной $OBDD$ от $n_P + 2 \cdot n_{pln} + n_A$ переменных.

Символическое представление интерпретаций

Для начала рассмотрим символическое представление множества $Domain$. Поскольку это множество является конечным, существует минимальное $n_D \in \mathbb{N}$ такое, что $|Domain| \leq 2^{n_D}$. Следовательно, можно построить двоичную кодировку множеств $Domain$ следующего вида: $Domain_{\mathbb{B}} : Domain \to \mathbb{B}^{n_D}$.

Используя бинарную кодировку множества состояний внешней среды $S_{\mathbb{B}}$ и бинарную кодировку доменного множества, можно сопоставить каждому символу предметной переменной состояния $v_s \in Var_s$ функцию $\pi_s(v_s) : S \to Domain$. Для каждой такой функции можно построить соответствующее бинарное представление $\pi_s(v_s)_{\mathbb{B}} : \mathbb{B}^{n_S} \to \mathbb{B}^{n_D}$, которое, в свою очередь, описывается разделяемой $OBDD$ из n_D компонент от n_S переменных. Совокупность всех таких функций, представленная в виде разделяемой $OBDD$ из $n_D \cdot |Var_s|$ ком-

понент от n_S переменных, является символическим представление функции $\pi_s : S \times Var_s \to Domain$. Аналогичным образом, в виде разделяемой $OBDD$ из $n_D \cdot |Var_a|$ компонент от n_A переменных можно представить функцию π_a.

Интерпретацию функциональных символов Φ_i размерности i можно рассматривать как набор двоичных функций вида $\Phi_i(f_i)_{\mathbb{B}} : \mathbb{B}^{n_D \cdot i} \to \mathbb{B}^{n_D}$ по одной для каждого функционального символа $f_i \in Func_i$. Этот набор функций в совокупности может быть представлен разделяемой $OBDD$ из $n_D \cdot |Func_i|$ компонент от $n_D \cdot i$ переменных. В случае, если $i = 0$ (для символов констант), их символическое представление выродится в набор из $n_D \cdot |Func_0|$ графов, каждый из которых состоит ровно из одной терминальной вершины (0 или 1).

Аналогичным образом интерпретация предикатных символов Π_i размерности i может быть представлена как набор булевых функций $\Pi_i(P_i)_{\mathbb{B}} : \mathbb{B}^{n_D \cdot i} \to \mathbb{B}$ по одной для каждого предикатного символа $P_i \in Pred_i$. В совокупности этот набор может быть представлен разделяемой $OBDD$ из $|Pred_i|$ компонентов от $n_D \cdot i$ переменных. Для случая $i = 0$ (для логических констант), представление выродится в набор из $|Pred_0|$ одновершинных графов.

5.2.2. Алгоритмы

В данном подразделе приводятся основные алгоритмы верификации для логики спецификации интеллектуального агента в семантике со статическим ментальным состоянием. Предложенные алгоритмы используют символическое представление входных данных, описанное в подразделе 5.2.1. Доказательства корректности, полноты и анализ алгоритмической сложности предложенных алгоритмов приведен в приложении на странице 198.

Вычисление атомарных формул

Атомарными формулами логики спецификации интеллектуального агента являются формулы вида $P_i(\tau_1, \ldots, \tau_i)$, где $P_i \in Pred_i$ есть предикатный символ размерности i, а τ_j есть термы (состояния или действия). Следовательно, при построении алгоритма верификации необходимо решить задачу нахождения множества состояний внешней среды $[P_i(\tau_1^s, \ldots, \tau_i^s)]^s \subseteq S$, такого, что для любого $s \in [P_i(\tau_1^s, \ldots, \tau_i^s)]^s$ выполнено $\Pi_i(P_i)(\lceil\tau_1^s\rceil(s), \ldots, \lceil\tau_i^s\rceil(s)) = 1$.

Для вычисления искомого множества сначала рекурсивно вычислим для каждого из термов-аргументов τ_1^s представление функции $\lceil\tau_1^s\rceil_{\mathbb{B}} : \mathbb{B}^{n_S} \to \mathbb{B}^{n_D}$. Для термов, являющих предметными переменными, это представление совпадает с кодировкой интерпретаций соответствующей предметной переменной $\pi_s(v_s)_{\mathbb{B}} : \mathbb{B}^{n_S} \to \mathbb{B}^{n_D}$. В случае же, если терм является вычислением функционального символа, для построения $\lceil\tau_1^s\rceil_{\mathbb{B}}$ можно воспользоваться операцией *композиции* функций [41].

После того, как для каждого терма-аргумента построено $\lceil\tau_1^s\rceil_{\mathbb{B}}$, взяв соответствующую предикату P_i функцию $\Pi_i(P_i)_{\mathbb{B}} : \mathbb{B}^{n_D \cdot i} \to \mathbb{B}$ и используя композицию функций, можно получить функцию вида $\mathbb{B}^{n_S} \to \mathbb{B}$, которая и описывает искомое множество. Корректность и полнота данного алгоритма доказывается в утверждении 38, а его алгоритмическая сложность проанализирована в утверждении 39

Аналогичным образом можно рекурсивно вычислить множество действий $[P_i(\tau_1^a, \ldots, \tau_i^a)]^a \subseteq A$ такое, что для любого действия $a \in [P_i(\tau_1^a, \ldots, \tau_i^a)]^a$ выполнено $\Pi_i(P_i)(\lceil\tau_1^a\rceil(a), \ldots, \lceil\tau_i^a\rceil(a)) = 1$.

Вычисление прообраза

Как и в случае с символической верификацией CTL (см. раздел 5.1.3), базовой операцией алгоритма является операция вычисления *прообраза* множества

состояний $x \subseteq S \times I_{pln}$ в контексте $env_c \subseteq S \times A \times S$ как множества состояний $Pre_{\mathbf{E}}(x, env_c) \subseteq S \times I_{pln}$, из которого система *может* попасть в одно из состояний множества x:

$$Pre_{\mathbf{E}}(x) \triangleq \{(s, i_{pln}) \in S \times I_{pln} \mid \exists (s', i'_{pln}) \in x, a \in A : (s, a, s') \in env_c \\ and\ (see(s), i_{pln}, a, i'_{pln}) \in \sigma_{pln}\}, \tag{5.4}$$

где σ_{pln} есть отношение переходов плана агента.

Для облегчения вычисления прообраза отношение переходов $\sigma_{pln} \subseteq P \times I_{pln} \times A \times I_{pln}$ удобнее рассматривать как отношение $\sigma'_{pln} \subseteq S \times I_{pln} \times A \times I_{pln}$, построенное по следующему принципу:

$$\sigma'_{pln} \triangleq \{(s, i_{pln}, a, i'_{pln}) \in S \times I_{pln} \times A \times I_{pln} \mid (see(s), i_{pln}, a, i'_{pln}) \in \sigma_{pln}\} \tag{5.5}$$

Для построения $OBDD$-представления отношения σ'_{pln} вычислим композицию функций $\sigma_{\mathbb{B}}$ и $see_{\mathbb{B}}$. Заметим, что построение отношения σ'_{pln} достаточно провести только один раз, а затем полученный результат можно использовать во всех остальных операциях.

Далее вычислим символическое представление отношения $env_c \| \sigma'_{pln} \subseteq S \times I_{pln} \times S \times I_{pln}$, являющегося комбинацией описания поведения внешней среды и плана агента, построенной по следующему принципу:

$$env_c \| \sigma'_{pln} \triangleq \{(s, i_{pln}, s', i'_{pln}) \in S \times I_{pln} \times S \times I_{pln} \mid \exists a \in A : (s, a, s') \in env_c \\ and\ (s, i_{pln}, a, i'_{pln}) \in \sigma'_{pln}\}. \tag{5.6}$$

Отношение $env_c \| \sigma_{pln}$ может быть представлено одной $OBDD$ от $2 \cdot (n_S + n_{pln})$ переменных. Для построения этой $OBDD$ достаточно выполнить следующие действия:

1. провести переиндексирование представлений отношений env_c и σ'_{pln} таким образом, чтобы они соответствовали отношению типа $S \times I_{pln} \times A \times S \times I_{pln}$;

2. построить пересечение этих представлений;

3. заменить все нетерминальные вершины $v \in N$, соответствующие представлению действия, на $high(v) \cup low(v)$;

4. провести переиндексирование полученного отношения таким образом, чтобы оно соответствовало отношению типа $S \times I_{pln} \times S \times I_{pln}$.

Так же как и представление отношения σ'_{pln}, представление отношения $env_c \| \sigma'_{pln}$ не нужно вычислять каждый раз при нахождении прообраза — достаточно вычислять его только при смене текущего контекста.

После построения представления отношения $env_c \| \sigma'_{pln} \subseteq S \times I_{pln} \times S \times I_{pln}$, функцию $Pre_{\mathbf{E}}(x, env_c)$ можно вычислить аналогично CTL следующим образом.

1. В представлении x все переменные, представляющие текущую пару состояний, переименовать в переменные, представляющее следующую пару состоянии. Т.е. переопределить все значения $index$ для всех нетерминальных вершин соответствующим образом. Полученное множество можно рассматривать как множество $S \times I_{pln} \times x$.

2. Вычислить представления множества $env_c \| \sigma'_{pln} \cap (S \times I_{pln} \times x)$.

3. В представлении множества $env_c \| \sigma'_{pln} \cap (S \times I_{pln} \times x)$, каждую нетерминальную вершину $v \in V$, соответствующую кодировке следующего состояния, заменить на корневую вершину графа $low(v) \vee high(v)$.

Используя функцию вычисления прообраза $Pre_{\mathbf{E}}(x, env_c)$ можно определить функцию вычисления *строгого* прообраза следующим образом:

$$Pre_{\mathbf{A}}(x, env_c) \triangleq Pre_{\mathbf{E}}(x, env_c) \setminus Pre_{\mathbf{E}}(S \times I_{pln} \setminus x, env_c). \qquad (5.7)$$

Вычисление оператора $[\alpha]\varepsilon$

Для обработки оператора динамической логики $[\alpha]\varepsilon$ вычислим *ограниченный прообраз* множества состояний внешней среды $x \subseteq S$ как множества таких состояний внешней среды $Pre_{\mathbf{E}}(x, env_c, \alpha) \subseteq S$, что для любого состояния $s \in Pre_{\mathbf{E}}(x, env_c, \alpha)$ существует такое действие $a \in A$ и состояние внешней среды $s' \in x$, что $M, a \vDash_a \alpha$ и $(s, a, s') \in env_c$. Для решения этой задачи сначала вычислим максимальное множество действий $[\alpha] \subseteq A$ такое, что для любого $a \in [\alpha]$ выполнено $M, a \vDash_a \alpha$. Это множество легко вычисляется рекурсией по структуре формулы α как показано в алгоритме 16.

Алгоритм 16 (Вычисление множества действий)
switch α
  case $\alpha = P_i(\tau_1^a, \ldots, \tau_i^a)$:
    $[\alpha] := [P_i(\tau_1^a, \ldots, \tau_i^a)]^a$;
  case $\alpha = \neg\beta$:
    $[\alpha] := A \setminus [\beta]$;
  case $\alpha = \beta_1 \wedge \beta_2$:
    $[\alpha] := [\beta_1] \cap [\beta_2]$;
end switch

После того, как множество $[\alpha]$ вычислено, ограниченный прообраз можно вычислить следующим образом:

$$Pre_{\mathbf{E}}(x, env_c, \alpha) = ((S \times [\alpha] \times S) \cap env_c \cap (S \times A \times x))\,|_S. \qquad (5.8)$$

С точки зрения операций с $OBDD$ декартовое произведение выполняется посредством соответствующей переиндексации и расширения количества переменных, а проекция множества — посредством замещения нетерминальных вершин v с ненужными индексами на $high(v) \cup low(v)$ и обратной переиндексации.

Кроме того, можно вычислить *строгий* ограниченный прообраз множества $x \subseteq S$, содержащий только те состояния внешней среды $s \in Pre_{\mathbf{A}}(x, env_c, \alpha)$, в которых выполнение любого действия $a \in [\alpha]$ приведет среду в одно из состоя-

ний x:

$$Pre_{\mathbf{A}}(x, env_c, \alpha) = Pre_{\mathbf{E}}(x, env_c, \alpha) \setminus Pre_{\mathbf{E}}(S \setminus x, env_c, \alpha). \tag{5.9}$$

Основной алгоритм

Алгоритм верификации для логики спецификации интеллектуального агента в семантике со статическим ментальным состоянием (алгоритм 17) основан на рекурсивном спуске по структуре формулы и нахождении для каждой из подформул ϕ множества таких пар состояний $[\phi]_{env_c} \subseteq S \times I_{pln}$, что для любой пары $(s, i_{pln}) \in [\phi]_{env_c}$ выполнено $(M, i), env_c, (s, i_{pln}) \vDash_s \phi$. Нетрудно заметить, что решение изначальной задачи сводится к описанному следующим образом:

$$[\phi] = ([\phi]_{env} \cap S \times \{i_{0,pln}\})|_S. \tag{5.10}$$

В случае, если $\phi = P_i(\tau_1^s, \ldots, \tau_i^s)$ для вычисления множества $[\phi]_{env_c}$ используются алгоритм вычисления предикатов $[P_i(\tau_1^s, \ldots, \tau_i^s)]^s$, описанный в разделе 5.2.2, а для обработки логических связок используются стандартные методы — дополнение для $\neg$ и пересечение для $\wedge$.

Множество $[[\alpha]\psi]_{env_c}$ вычисляется как строгий ограниченный прообраз множества $[\psi]_{env_c}$, а множества $[\mathbf{E} \bigcirc \psi]_{env_c}$ ($[\mathbf{A} \bigcirc \psi]$) — как прообраз (строгий прообраз) множества $[\psi]_{env_c}$.

По аналогии с CTL, множество $[\mathbf{E}\, \psi_1\, \mathcal{U}_{\leq t} \psi_2]_{env_c}$ вычисляется как неподвижная точка оператора $x = x \cup (Pre_{\mathbf{E}}(x, env_c) \cap [\psi_1]_{rnv_c}$, начиная с $x = [\psi_2]_{env_c}$. Так как этот оператор является монотонным (множество x только растет), эта неподвижная точка будет найдена не более чем за $|S| \cdot |I_{pln}|$ итераций. Кроме того, при достижении t итераций поиск прекращается (срабатывает ограничение на время реакции). Для вычисления множества $[\mathbf{A}\, \psi_1\, \mathcal{U}_{\leq t} \psi_2]_{env_c}$ ищется неподвижная точка оператора $x = x \cup (Pre_{\mathbf{A}}(x, env_c) \cap [\psi_1]_{rnv_c}$.

Множество $[\mathsf{Bel}\ \psi]_{env_c}$ вычисляется как сужение множества $[\psi]_{i|_{bel}}$ (где $i|_{bel}$ есть представления агента) с учетом восприятия агента (каждое состояние либо

Алгоритм 17 (Алгоритм верификации)

switch ϕ

case $\phi = P_i(\tau_1^s, \ldots, \tau_i^s)$:

$[\phi]_{env_c} := [P_i(\tau_1^s, \ldots, \tau_i^s)]^s \times I_{pln}$;

case $\phi = \neg\psi$:

$[\phi]_{env_c} := S \times I_{pln} \setminus [\psi]_{env_c}$;

case $\phi = \psi_1 \wedge \psi_2$:

$[\phi]_{env_c} := [\psi_1]_{env_c} \cap [\psi_2]_{env_c}$;

case $\phi = [\alpha]\psi$:

$[\phi]_{env_c} := Pre_{\mathbf{A}}([\psi]_{env_c}|_S, env_c, \alpha) \times I_{pln}$;

case $\phi = \mathbf{E} \bigcirc \psi$:

$[\phi]_{env_c} := Pre_{\mathbf{E}}([\psi]_{env_c}, env_c)$;

case $\phi = \mathbf{A} \bigcirc \psi$:

$[\phi]_{env_c} := Pre_{\mathbf{A}}([\psi]_{env_c}, env_c)$;

case $\phi = \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$:

$x := \emptyset$; $\quad y := [\psi_2]_{env_c}$; $\quad m := t$;

while $y \not\subseteq x$ ***and*** $--m \geq 0$ ***do***

$x := x \cup y$; $\quad y := Pre_{\mathbf{E}}(x, env_c) \cap [\psi_1]_{env_c}$;

od;

$[\phi]_{env_c} := x \cup y$;

case $\phi = \mathbf{A}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$:

$x := \emptyset$; $\quad y := [\psi_2]_{env_c}$; $\quad m := t$;

while $y \not\subseteq x$ ***and*** $--m \geq 0$ ***do***

$x := x \cup y$; $\quad y := Pre_{\mathbf{A}}(x, env_c) \cap [\psi_1]_{env_c}$;

od;

$[\phi]_{env_c} := x \cup y$;

case $\phi = \mathsf{Bel}\ \psi$:

$[\phi]_{env_c} := \{(s, i_{pln}) \in S \times I_{pln} \mid \forall s' \in see(s) : (s', i_{pln}) \in [\psi]_{i|_{bel}}\}$;

case $\phi = \mathsf{Des}\ \psi$:

if $\psi \in i_{des}$ ***then*** $[\phi]_{env_c} := S \times I_{pln}$ ***else*** $[\phi]_{env_c} = \emptyset$ ***fi***;

case $\phi = \mathsf{Intend}\ \psi$:

$[\phi]_{env_c} := [\mathsf{Des}\ \psi]_{env_c} \cap [\mathsf{Bel}\ \psi]_{env_c}$;

end switch

входит в множество вместе со всем своим классом эквивалентности, либо не входит вообще):

$$[\mathsf{Bel}\ \psi]_{env_c} := \{(s, i_{pln}) \in S \times I_{pln} \mid \forall s' \in see(s) : (s', i_{pln}) \in [\psi]_{i|_{bel}}\}.$$

Для того, чтобы вычислить множество $[\mathsf{Bel}\ \psi]_{env_c}$, сначала вычислим множество $see(S \setminus [\psi]_{i|_{bel}}|_S) \triangleq \{s \in S \mid \exists s' \in S : (s, s') \in see\ and\ s' \notin [\psi]_{i|_{bel}}|_S\}$, после чего $[\mathsf{Bel}\ \psi]_{env_c}$ может быть вычислено как $[\psi]_{i|_{bel}} \setminus (see(S \setminus [\psi]_{i|_{bel}}|_S) \times I_{pln})$.

Для вычисления множества $see(x)$ удобно использовать представление see в виде отношения $S \times S$. В этом случае искомое множество вычисляется как прообраз множества $x \subseteq S$ относительно отношения see: $see(x) = (see \cap (S \times x))|_S$.

Проще вычисляется множество $[\mathsf{Des}\ \psi]_{env_c}$ — если формула ψ входит во множество желаний агента ($\psi \in i_{des}$), то $[\mathsf{Des}\ \psi]_{env_c} = S \times I_{pln}$, иначе $[\mathsf{Des}\ \psi]_{env_c} = \emptyset$. Таким образом, формулы вида $\mathsf{Des}\ \psi$ в случае зафиксированного ментального состояния всегда истинны или всегда ложны, в зависимости от этого ментального состояния.

Множество $[\mathsf{Intend}\ \psi]_{env_c}$ вычисляется как пересечение множеств $[\mathsf{Des}\ \psi]_{env_c}$ и $[\mathsf{Bel}\ \psi]_{env_c}$. Таким образом, в случае статического ментального состояния, намерения есть желания, которые, по представлениям агента, будут реализованы в результате выполнения агентом своего плана.

Глава 6

Логика спецификации мультиагентной системы

При разработке логики спецификации мультиагентной системы логику спецификации отдельного агента необходимо расширить средствами описания представлений, желаний и намерений групп агентов, а также средствами описания результатов их действий. Для решения этой проблемы в данной главе синтаксис логики спецификации интеллектуального агента расширяется квантором коалиции $\langle\langle C\rangle\rangle$, используемым для указания *текущей коалиции*. При этом семантика темпоральных и ментальных операторов определяется относительно текущей коалиции, рассматриваемой в качестве единого агента, в соответствии с результатами, полученными в главе 3.

6.1. Синтаксис

Определим синтаксис логики спецификации мультиагентной системы следующим образом.

Определение 18 *Синтаксис логики спецификации мультиагентной системы расширяет синтаксис логики спецификации интеллектуального агента (определение 8) дополнительным алфавитом* символов коалиций $Coal$ *и переопределяет грамматику построения формул состояния (уравнение 4.30) следующим образом:*

$$\begin{aligned}\phi \;::=\; & \varepsilon \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{A}\,\rho \mid \mathbf{E}\,\rho \mid \\ & \mathsf{Bel}\;\phi \mid \mathsf{Des}\;\phi \mid \mathsf{Intend}\;\phi \mid \langle\langle C\rangle\rangle\phi, \quad (C \in Coal).\end{aligned} \tag{6.1}$$

Таким образом, по сравнению с логикой спецификации интеллектуального агента, язык расширяется дополнительной конструкцией $\langle\langle C\rangle\rangle\phi$, интуитивно

интерпретируемой как “Для коалиции C выполнено свойство ϕ”. Квантор коалиции $\langle\langle C\rangle\rangle$ является наиболее слабым по приоритету оператором, при этом формулы, не находящиеся в области действия ни одного из кванторов, имеют интуитивную интерпретацию “для коалиции всех агентов выполнено свойство ϕ”.

6.2. Семантика

Описание семантики начнем с определения *модели.*

Определение 19 *Моделью для интерпретации формул логики спецификации мультиагентной системы (формул вида ϕ из определения 18) является набор* $M = \langle MAS, Domain, \pi_s, \pi_a, \{\Phi_i\}_{i\in\{0,\ldots,n_f\}}, \{\Pi_i\}_{i\in\{0,\ldots,n_p\}}, \gamma\rangle$, *где*

- $MAS = (S, AG, env)$ *есть модель мультиагентной системы (определение 5), где множества желаний агентов ограничены множествами логических формул* ($\forall ag \in AGI_{ag,D} = 2^{2^\phi}$);
- $Domain$, pi_s, pi_a, Φ_i *и* Π_i *есть доменное множество, интерпретации символов переменных, предикатных и функциональных символов аналогично определению 9;*
- $\gamma : Coal \to 2^{AG}$ *есть интерпретация символов коалиций, сопоставляющая каждому символу* $C \in Coal$ *некоторое подмножество агентов* $\gamma(C) \subseteq AG$.

По аналогии с логикой спецификации интеллектуального агента, для логики спецификации мультиагентной системы можно определить два вида семантики: с динамическим и со статическим ментальным состоянием. Первый вариант удобен для описания системы на больших участках времени, тогда как второй

более удобен для рассмотрения системы на этапе исполнения определенного плана.

6.2.1. Семантика с динамическим ментальным состоянием

Определение 20 *Семантику логики спецификации мультиагентной системы с динамическим ментальным состоянием зададим с помощью трех отношений:*

- *для формул действий* $M, a \models_a \alpha$*;*
- *для формул состояний* $M, C, (env, act),\ (s^C, i^C, i_{act}) \models_s \phi$*;*
- *для формул пути* $M, C, (env, act), \lambda \models_p \rho$*;*

где

- $M = \langle MAS, Domain, \pi_s, \pi_a,\ \{\Phi_i\}_{i\in\{0,\ldots,n_f\}}, \{\Pi_i\}_{i\in\{0,\ldots,n_p\}}, \gamma\rangle$ *есть модель;*
- $a \in ACS$ *есть некоторое действие системы;*
- $C \subseteq AG$ *есть текущая коалиция агентов;*
- $env \subseteq S^C \times A^C \times S^C$ *есть описание поведения внешней среды в текущем контексте относительно текущей коалиции;*
- $act \subseteq S^C \times I^C \times I_{act} \times A^C \times I_{act}$ *есть отношение, описывающее способ выбора действия в текущем контексте относительно текущей коалиции. При этом множество* I_{act} *описывает потенциальные дополнительные параметры выбора;*
- $(s^C, i^C, i_{act}) \in S^C \times I^C \times I_{act}$ *есть текущее состояние системы, включая состояние внешней среды* s^C*, внутреннее состояние текущей коалиции* i^C *и текущее значение дополнительного параметра выбора действия* i_{act}*;*

- $\lambda \in (S^C \times I^C \times I_{act})^+$ *есть некоторый путь, представленный цепочкой троек вида* $(s^C, i^C, i_{act}) \in S^C \times I^C \times I_{act}$.

Отношения выполнимости для логики спецификации мультиагентной системы в случае динамического ментального состояния (определение 20) расширяют аналогичные отношения для логики интеллектуального агента (определение 10) введением *текущей коалиции*, рассматриваемой как единый агент. Семантические правила для интерпретации этих отношений аналогичны случаю одного агента, при этом для интерпретации квантора коалиции вводится следующее дополнительное правило.

13. $M, C, (env^C, act^C), (s^C, i^C, i_{act}) \models_s \langle\langle D \rangle\rangle \rho$ тогда и только тогда, когда $\gamma(D) \subseteq C$ и $M, \gamma(D), (env^C|_D, act^C|_D), (s^C|_D, i^C|_D, i_{act}|_D) \models_p \rho$,

где $env^C|_D$ и $i^C|_D$ есть сужения описания внешней среды и состояния коалиции (см. уравнения 3.8 и 3.9), $s^C|_D = s^C$ (так как $S^D = S^C = S \times ACS \cup \{\varnothing\}$), а $act^C|_D$ есть сужение описания выбора действия, построенное следующим образом:

- если $I_{act} = \emptyset$, то $act^C|_D \triangleq act^C|_{S^D \times I^D \times A^D}$ и $i_{act}|_D = i_{act} = \varnothing$;
- если $I_{act} = I^C_{pln}$ (т.е. $act^C = act_{plan^C}$), то $act^C|_D \triangleq act_{plan^C|_D}$ и $i_{act}|_D \triangleq i_{act}|_{I^D_{pln}}$, где $plan^C|_D$ есть сужение плана коалиции (см. раздел 3.3.4).

6.2.2. Семантика со статическим ментальным состоянием

Так как высокоуровневая коммуникация агентов не влияет на исполнение текущих планов, при определении семантики со статическим ментальным состоянием ее можно не учитывать. Это ведет к тому, что действия системы можно исключить из описания внешней среды коалиции ($S^C = S$).

Определение 21 *Семантику логики спецификации мультиагентной системы со статическим ментальным состоянием зададим с помощью трех отношений:*

- *для формул действий* $M, a \models_a \alpha$*;*
- *для формул состояний* $(M, i^C), C, env, (s, i^C_{pln}) \models_s \phi$*;*
- *для формул пути* $(M, i^C), C, env, \lambda \models_p \rho$*;*

где

- $M = \langle MAS, Domain, \pi_s, \pi_a, \{\Phi_i\}_{i\in\{0,\ldots,n_f\}}, \{\Pi_i\}_{i\in\{0,\ldots,n_p\}}, \gamma\rangle$ *есть модель;*
- $a \in ACS$ *есть некоторое действие системы;*
- $C \subseteq AG$ *есть текущая коалиция агентов;*
- $env \subseteq S \times A^C \times S$ *есть описание поведения внешней среды в текущем контексте относительно текущей коалиции;*
- $(s, i^C_{pln}) \in S \times I^C_{pln}$ *есть пара состояний, где* $s \in S$ *есть состояние внешней среды, а* $i^C_{pln} \in I^C_{pln}$ *есть состояние плана коалиции;*
- $\lambda \in (S \times I^C_{pln})^+$ *есть некоторый путь, представленный цепочкой пар вида* $(s, i^C_{pln}) \in S \times I^C_{pln}$*.*

Отношения выполнимости для логики спецификации мультиагентной системы в случае статического ментального состояния (определение 21) расширяют аналогичные отношения для логики интеллектуального агента (определение 13) введением *текущей коалиции*, рассматриваемой как единый агент. Семантические правила для интерпретации этих отношений аналогичны случаю одного агента, при этом для интерпретации квантора коалиции вводится следующее дополнительное правило.

13. $(M, i^C), C, env^C, (s, i^C_{pln}) \models_s \langle\langle D\rangle\rangle\rho$ тогда и только тогда, когда $\gamma(D) \subseteq C$ и $(M, i^C|_D), \gamma(D), env^C|_D, (s, i^C_{pln}|_D) \models_p \rho$,

где $env^C|_D$ и $i^C|_D$ есть сужения описания внешней среды и состояния коалиции (см. уравнения 3.8 и 3.9), а $i^C_{pln}|_D = i^C_{pln}|_{I^D_{pln}}$.

Заметим, что при таком определение семантического правила, при соблюдении корректной вложенности кванторов коалиции (если квантор коалиции $\langle\langle D\rangle\rangle$ находится в области действия другого квантора коалиции $\langle\langle C\rangle\rangle$, то $D \subseteq C$), квантор коалиции является самодвойственным:

$$\vdash \neg\langle\langle C\rangle\rangle\phi \Leftrightarrow \langle\langle C\rangle\rangle\neg\phi. \tag{6.2}$$

6.3. Алгоритмы верификации

Задачу верификации для логики спецификации мультиагентной системы в семантике со статическим ментальным состоянием определим следующим образом: по заданной модели M, состоянию коалиции всех агентов системы $i^{AG} = (i_{ag_1}, \ldots, i_{ag_n}) \in I_{ag_1} \times \ldots \times I_{ag_n}$ и формуле ϕ определить такое множество состояний внешней среды $[\phi] \subseteq S$, что для любого состояния $s \in [\phi]$ выполнено $(M, i^A G), AG, env, (s, i^{AG}|_{0,pln}) \models \phi$, где $i^{AG}|_{0,pln}$ есть начальное состояние плана коалиции всех агентов, а env есть описание поведения внешней среды.

По сравнению с алгоритмами верификации логики спецификации интеллектуального агента, описанными в разделе 5.2, для случая мультиагентной системы необходимо решить следующие дополнительные задачи:

- разработать методы символического представления коалиции-агента;
- разработать алгоритмы построения символического представления коалиции-агента на основе представлений агентов коалиции;
- разработать алгоритмы построения сужения описания внешней среды и состояния коалиции.

Доказательства корректности, полноты и анализ алгоритмической сложности предложенных алгоритмов приведены в приложении А.2.

6.3.1. Символическое представление коалиции-агента

Главным свойством коалиции является множество агентов, в нее входящих. Так как множество всех агентов является относительно небольшим, для представления этого свойства удобнее использовать обычные списки, а не разрешающие диаграммы. Остальными свойствами коалиции, требующими символического представления, являются: описание внешней среды для коалиции, восприятие коалиции, представления и план коалиции.

Для конечных множеств состояний внешней среды S и действий системы ACS существуют двоичные кодировки $S_{\mathbb{B}} : S \to \mathbb{B}^{n_S}$ и $ACS_{\mathbb{B}} : ACS \to \mathbb{B}^{\sum\limits_{ag \in AG} n_{A_{ag}}}$, где n_S и $n_{A_{ag}}$ есть минимальные натуральные числа такие, что $|S| \leq 2^{n_S}$ и $|A_{ag}| \leq 2^{n_{A_{ag}}}$. Аналогично, для множества действий коалиции A^C можно предложить двоичную кодировку вида $A^C_{\mathbb{B}} : A^C \to \mathbb{B}^{\sum\limits_{ag \in C} n_{A_{ag}}}$.

Описание внешней среды коалиции $env^C \subseteq S \times A^C \times S$ можно представить в виде функции $env^C_{\mathbb{B}} : \mathbb{B}^{n_S + \sum\limits_{ag \in C} n_{A_{ag}} + n_S} \to \mathbb{B}$. Однако, для упрощения переключения между коалициями, более удобной формой представления будет функция $env^C_{\mathbb{B}} : \mathbb{B}^{n_S + \sum\limits_{ag \in AG} n_{A_{ag}} + n_S} \to \mathbb{B}$. Эту функцию можно представить в виде $OBDD$ от $n_S + \sum\limits_{ag \in AG} n_{A_{ag}} + n_S$ переменных, не содержащих нетерминальнальных вершин, соответствующих действиям агентов вне коалиции.

Восприятие коалиции $see^C \subseteq S \times S$ будем рассматривать как вектор-функцию вида $see^C : S \to \bigtimes\limits_{ag \in C} P^C$, заданную разделяемой $OBDD$ из $\sum\limits_{ag \in C} n_{P_{ag}}$ компонент от n_S переменных. Такое представление не является оптимальным по количеству компонентов, однако оно позволяет более просто реализовать комбинацию восприятий агентов.

Представления коалиции $bel^C \subseteq S \times A^C \times S$, аналогично описанию внешней

среды, можно задать с помощью функции $bel^C_{\mathbb{B}} : \mathbb{B}^{n_S+\sum\limits_{ag\in AG} n_{A_{ag}}+n_S} \to \mathbb{B}$, представленной $OBDD$ от $n_S + \sum\limits_{ag\in AG} n_{A_{ag}} + n_S$ переменных.

Отношение переходов плана коалиции $\sigma^C \subseteq P^C \times I^C_{pln} \times I^C_{pln} \times A^C$ рассмотрим как отношение вида $\sigma'^C \subseteq S \times I^C_{pln} \times I^C_{pln} \times ACS$ и представим как функцию $\sigma^C_{\mathbb{B}} : \mathbb{B}^{n_S+|C|\cdot n_{pln}+\sum\limits_{ag\in AG} n_{A_{ag}}} \to \mathbb{B}$ (где n_{pln} есть ограничение на размер множества внутренних состояний плана агента $|I_{pln}| \leq 2^{n_{pln}}$), заданную $OBDD$ от $n_S + |C| \cdot n_{pln} + \sum\limits_{ag\in AG} n_{A_{ag}}$ переменных.

Кроме того, для каждого из агентов необходимо описать функцию посылки сигналов $send_{ag,pln} : P_{ag} \times I_{pln} \times AG \to Sig_{ag}$, являющуюся частью плана агента. Эту функцию удобнее рассматривать как семейство функций $send_{ag,pln}(ag_i) : S \times I_{pln} \to Sig_{ag}$,[1] каждую из которых можно представить в виде двоичной функции $send_{ag,pln}(ag_i)_{\mathbb{B}} : \mathbb{B}^{n_S+n_{pln}} \to \mathbb{B}^{n_{sig}}$, где число n_{sig} определяет максимальный размер множества посылаемых сигналов ($\forall ag \in AG : |Sig_{ag}| \leq 2^{n_{sig}}$). В итоге, все семейство в целом может быть представлено разделяемой $OBDD$ из $|AG| \cdot n_{sig}$ компонент, зависящих от $n_S + n_{pln}$ переменных.

6.3.2. Восприятие, представления и планы коалиции

Наиболее простым является построение восприятия коалиции агента see^C, так как для него достаточно скомбинировать представления восприятий агентов коалиции в единую разделяемую $OBDD$.

Для построения $OBDD$ описания представлений коалиции необходимо вычислить множество $\bigcup\limits_{ag\in C}(bel_{ag}) \cap \bigcup\limits_{ag\in C}(sbel_{ag} \times S)$, после чего устранить ($v = low(v) \cup high(v)$) из него нетерминальные вершины, соответствующие действиям других агентов.

При построении отношения переходов плана коалиции $\sigma^C_{pln} \subseteq S \times I^C_{pln} \times I^C_{pln} \times A^C$ необходимо привести все планы агентов $\sigma_{ag,pln} \subseteq P_{ag} \times Sig_{ag_1} \times \ldots \times Sig_{ag_n} \times$

[1]Механизм перевода $OBDD$, зависящих от P, в $OBDD$, зависящие от S, предложен в параграфе 5.2.2.

$I_{ag,pln} \times I_{ag,pln} \times A_{ag}$ к виду $\sigma'_{ag,pln} \subseteq S \times I^C_{pln} \times I^C_{pln} \times A^C$. После чего план коалиции может быть построен как пересечение планов агентов: $\sigma^C_{pln} = \bigcap\limits_{ag \in C} \sigma'_{ag,pln}$.

Для этого преобразования из всех планов агентов коалиции устраним ($v = low(v) \cap high(v)$) из представления плана вершины, соответствующие сигналам агентов вне коалиции, оставив тем самым только те правила, которые гарантированно не зависят от сигналов извне. Полученное множество обозначим как $\sigma^r_{ag,pln}$ После чего для каждого из планов агентов, принявших вид $\sigma^r_{ag,pln} \subseteq P_{ag} \times (\underset{ag' \in C}{\times} Sig_{ag'}) \times I_{ag,pln} \times I_{ag,pln} \times A_{ag}$, вычислим композицию с функцией $sas : S \times (\underset{ag' \in C}{\times} I_{ag',pln} \rightarrow P_{ag} \times (\underset{ag' \in C}{\times} Sig_{ag'})$, заданной следующим образом: $sas(s, i^C_{pln}) = (see_{ag}(s), (send_{ag',pln}(s, i^C_{pln}|_{I_{ag,pln}}))_{ag' \in C})$. После чего достаточно простой переиндексации, чтобы планы приняли нужный вид.

6.3.3. Сужение коалиции

Для реализации алгоритмов верификации логики спецификации мультиагентной системы необходимо расширить шаг рекурсии алгоритма 17 $[\phi]_{env}$ до $[\phi]_{env,C}$, где C есть текущая коалиция. С учетом представления коалиции как единого агента, построенного с помощью алгоритмов, описанных в подразделе 6.3.2, все операции алгоритма 17 адаптируются очевидным образом.

Кроме того, алгоритм 17 необходимо расширить обработкой квантора коалиции $\langle\langle D \rangle\rangle$, которая заключается в построении сужения текущего описания внешней среды $env \subseteq S \times ACS \times S$ относительно новой коалиции $\gamma(D)$. Это сужение реализуется с помощь устранения ($v = low(v) \cup high(v)$) в представлении env нетерминальных вершин, соответствующих действиям агентов вне коалиции $\gamma(D)$. Кроме того, при переключении к новой коалиции необходимо заново построить $OBDD$-представление плана коалиции, её восприятия и представлений. Заметим также, что, если $\gamma(D) \nsubseteq C$, результатом алгоритма должно быть пустое множество (нарушена корректность вложенности).

6.4. Пример

Для иллюстрации предложенных концепций рассмотрим пример мультиагентной системы управления автономным исследовательским зондом, оснащенным солнечными батареями и аккумуляторами, шасси, позволяющим перемещение и повороты на месте, а также манипулятором, позволяющим поднимать предметы из окружающего мира.

Любая выполняемая зондом операция ведет к расходу энергии и может истощить аккумуляторы, чего управляющая система не должна допустить ни в коем случае. Для предотвращения разрядки аккумуляторов зонд может использовать солнечные батареи — если батареи раскрыты, то аккумулятор автоматически заряжается, но только в светлое время суток.

Кроме того, некоторые явления внешней среды, например, осадки или песчаные бури, могут оказать вредное воздействие на солнечные батареи, чего допускать нельзя. Для предотвращения вредного воздействия зонд должен сворачивать солнечные батареи, после чего они перестанут заряжать аккумулятор, пока их снова не развернуть.

При перемещениях по внешнему миру зонд может встретить препятствия, столкновений с которыми он должен избегать.

Осторожность требуется и при работе с манипулятором — зонд должен пытаться поднимать предметы только определенных размеров, чтобы избежать повреждения. Кроме того, манипулятором нельзя пользоваться во время движения и операций с солнечными батареями.

6.4.1. Спецификация возможных действий

Систему управления таким зондом логично компоновать из трех агентов.

- Управления солнечными батареями SB с возможными действиями: открыть батареи $open$, закрыть батареи $close$ и ничего не делать $noop_{SB}$.

- Управления движением MV с возможными действиями: переместиться вперед $move$, повернуться $turn$ и ничего не делать $noop_{MV}$.

- Управления манипулятором MN с двумя возможными действиями: поднять предмет $pickup$ и ничего не делать $noop_{MN}$.

6.4.2. Спецификация внешней среды

Далее мы перейдем к описанию состояния внешней среды мультиагентной системы. Заметим, что к внешней среде будет относиться как окружающий зонд мир, так и собственная аппаратура зонда. Введем следующие предметные переменные:

- Состояние батарей описывает переменная $batteryOpen$, интерпретирующаяся как 1 в том случае, если батареи открыты, и 0 иначе.

- Для описания состояния аккумулятора используем предметную переменную $charge$, принимающую значения из множества $\{0, 1, 2, 3\}$.

- Переменной $isSunny$ будем обозначать тот факт, что в данный момент солнца достаточно для зарядки батарей (1, если достаточно, и 0 иначе).

- Наличие опасных для солнечных батарей явлений будем обозначать переменной $danger$ на множестве $\{0, 1\}$.

- Для описания окружающей зонд обстановки будем использовать семейство переменных $obj_{i,j}$ $(0 < i, j \leq n)$, принимающее значения $obstacle$, $sample$ или $nothing$ и описывающих *карту мира* агента. Значение $obstacle$ указывает на то, что в позиции (i, j) находится препятствие, $sample$ — предмет, который можно поднять, а $nothing$ — пустота.

- Текущую позицию зонда в мире обозначим двумя переменными cur_x и cur_y, принимающих значения на множестве $\{1, \ldots, n\}$, а ориентацию зонда переменной $orient$, принимающей значения на множестве $\{N, W, S, E\}$.

- Количество подобранных предметов опишем переменной $fitnes$, принимающей значения на $\{0, \ldots, n^2\}$.

- Факт того, что зонд получил повреждения, мы будем обозначать переменными $hullDamaged$, $armDamaged$ и $batteryDamaged$, принимающими значения на $\{0, 1\}$. Первая пропозиция будет обозначать повреждение корпуса, вторая будет обозначать повреждение манипулятора, а третья — повреждение солнечных батарей.

При описании формул мы будем использовать функциональные символы стандартных арифметических операций, предикат равенства и арифметического сравнения, константы, соответствующие каждому из объектов предметного множества, а также логические константы $true$ и $false$. Для переменных, принимающих значения на множестве $\{0, 1\}$ будем опускать запись $v = 1$ и писать вместо нее просто v.

Теперь формализуем ранее описанные неформально закономерности внешней среды, используя $MASL$. Для начала опишем закономерности, связанные с зарядом аккумулятора и состоянием солнечных батарей.

- $(batteryOpen \wedge isSunny) \Rightarrow \bigwedge\limits_{0 \le i \le 3} (charge \ge i \Rightarrow [true]charge > i)$ — это утверждение описывает закономерности изменения заряда аккумулятора. В условиях наличия солнца и открытых батарей действия агентов не ведут к ухудшению заряда аккумулятора.

- $\bigwedge\limits_{0 \le i \le 3} (charge = i \Rightarrow [(noop_{SB} \vee open) \wedge noop_{MV} \wedge noop_{MN}]charge = i)$ — операция открытия батарей или полное бездействие не ведет к изменению заряда аккумулятора.

- $(batteryOpen \Rightarrow [close \wedge noop_{MV} \wedge noop_{MN}] \neg batteryOpen \wedge \neg batteryDamaged) \wedge (\neg batteryOpen \wedge \neg danger \Rightarrow [open \wedge noop_{MV} \wedge noop_{MN}] batteryOpen \wedge \neg batteryDamaged)$ — это утверждение описывает последствия операций открытия или закрытия батарей, которые выполнялись в условиях отсутствия других параллельных действий. Заметим, что безопасность операции открытия батарей обеспечена только при отсутствии опасных явлений во внешнем мире.

- $\neg danger \vee \neg batteryOpen \Rightarrow [\neg((close \vee open) \wedge (move \vee turn \vee pickup))] \neg batteryDamaged$ — при отсутствии опасных явлений во внешнем мире или в условиях закрытых батарей действия, не являющиеся комбинацией операций над батареями с другими действиями, не ведут к повреждению батарей.

Рассмотрим закономерности, связанные с перемещениями агента и работой с манипулятором.

- Для упрощения спецификации определим дополнительный функциональный символ $infront$, соответствующий отображению вида $\{1, \ldots, n\} \times \{1, \ldots, n\} \times \{N, W, S, E\} \rightarrow \{obstacle, sample, nothing\}$ и сопоставляющий позиции зонда и его ориентации находящийся перед ним предмет.

- $(infront(cur_x, cur_y, orient) \neq obstacle \Rightarrow [move](\neg hullDamaged \wedge moved)) \wedge [\neg move] \neg hullDamaged$ — это утверждение описывает безопасные условия для перемещения зонда. Движение при отсутствии препятствий и любая комбинация действий без движения не ведет к повреждению корпуса.

- $infront(cur_x, cur_y, orient) = sample \Rightarrow \bigwedge\limits_{0 \leq i < n^2} (fitnes \geq i \Rightarrow [pickup \wedge noop_{MV}] \neg armDamaged \wedge fitnes \geq i+1) \wedge [\neg pickup] \neg armDamaged$ — если

перед зондом находится предмет, который можно поднять, то выполнение действия *pickup* приведет к тому, что будет подобран полезный предмет, если зонд не попытается переместиться или развернуться. Кроме того, только действие *pickup* может повредить манипулятор.

- $\bigwedge\limits_{0 \le i \le n^2} (fitnes = i \Rightarrow [\neg pickup] fitnes = i)$ — только действие *pickup* может приводить к тому, что будет подобран полезный предмет из внешнего мира.

6.4.3. Спецификация свойств системы

В этом разделе мы приведем примеры формализации некоторых свойств системы, состав, возможности и среду для которой мы описали выше. Как правило, при спецификации мультиагентной системы наибольший интерес представляют *инвариантные* свойства, то есть свойства, выполненные на протяжении всей жизни системы.

Рассмотрим формулировку некоторых наиболее простых свойств, заключающихся в отсутствии негативных событий.

- $\langle\langle \{SB, MV, MN\} \rangle\rangle \mathbf{A}\, G \neg (charge = 0 \vee batteryDamaged)$ — это свойство требует того, чтобы солнечные батареи зонда всегда были в порядке, а аккумулятор не был разряжен. Добиться выполнения этого свойства может только вся система в целом.

- $\langle\langle \{MV, MN\} \rangle\rangle \mathbf{A}\, G \neg armDamaged$ — это свойство требует того, чтобы манипулятор системы всегда не был поврежден. Добиваться этого система должна даже в том случае, если агент SB выйдет из строя.

- $\langle\langle \{MV\} \rangle\rangle \mathbf{A}\, G \neg hullDamaged$ — это свойство требует того, чтобы корпус зонда всегда оставался в порядке. Добиваться этого система должна даже в том случае, если нормально функционировать будет только агент MV.

Все приведенные свойства для описываемой системы могут быть выполнены при правильном и согласованном поведении агентов. При этом наиболее сложно достижение первого свойства, так как для этого необходимы согласованные действия именно всех агентов. Свойство вида $\langle\langle\{SB, MV\}\rangle\rangle \mathbf{A}\, G\neg(charge = 0 \vee batteryDamaged)$ уже не может быть выполнено в описываемых условиях, так как действие агента MN, несогласованное с действиями других агентов, может привести к повреждению солнечных батарей или разряду аккумулятора.

Немного более сложно описываются свойства, ограничивающие время реакции системы на появление негативных факторов.

- $\langle\langle\{SB, MV, MN\}\rangle\rangle \mathbf{A}\, G danger \;\Rightarrow\; \langle\langle\{SB, MV, MN\}\rangle\rangle \mathbf{A}\, \mathcal{F}_{<5} danger \;\Rightarrow\; \neg batteryOpen$ — это свойство требует того, чтобы всегда при возникновении опасных явлений система закрывала солнечные батареи не позднее чем через пять тактов времени, если к этому моменту опасность не устранится. Заметим, что это свойство слабее, чем свойство $\langle\langle\{SB, MV, MN\}\rangle\rangle \mathbf{A}\, G\neg batteryDamaged$, так как оставляет возможность для повреждения батарей.

- $\langle\langle\{SB, MV, MN\}\rangle\rangle \mathbf{A}\, G isSunny \wedge \neg danger \Rightarrow \langle\langle\{SB, MV, MN\}\rangle\rangle \mathbf{A}\, \mathcal{F}_{<10}$ $isSunny \wedge \neg danger \Rightarrow batteryOpen$ — это свойство требует, чтобы всегда при наличии солнца и отсутствии опасности система открывала солнечные батареи не позднее чем через 10 тактов времени.

Эти свойства могут быть выполнены при правильно согласованном поведении всех агентов системы.

Еще одним часто встречающимся классом свойств мультиагентной системы, являются свойства достижимости позитивных результатов, например.

- $\langle\langle\{SB, MV, MN\}\rangle\rangle \mathbf{E}\, \mathcal{F}\, fitnes \geq 1$ — это свойство требует того, чтобы

зонд когда-нибудь подобрал хотя бы один предмет хотя бы на одном из возможных путей.

6.4.4. Описание ментальных состояний

Предположим, что агент MN отвечает за восприятие предмета, находящегося перед зондом и количества собранных предметов, агент SB контролирует наличие солнца и опасных явлений, а также знает о текущем состоянии аккумулятора и закономерностях изменения заряда, а агент MV знает о возможных последствиях своих действий. В этом случае их знания об окружающей среде можно описать следующими выражениями.

- $\bigwedge_{obj \in \{obstacle, sample, nothing\}} (infront(cur_x, cur_y, orient) = obj \Leftrightarrow \langle\langle MN \rangle\rangle \mathsf{Bel}\ infront(cur_x, cur_y, orient) = obj)$.
- $(isSunny \Leftrightarrow \langle\langle SB \rangle\rangle \mathsf{Bel}\ isSunny) \wedge (danger \Leftrightarrow \langle\langle SB \rangle\rangle \mathsf{Bel}\ danger)$.
- $\bigwedge_{i \in \{0,1,2,3\}} (charge = i \Leftrightarrow \langle\langle SB \rangle\rangle \mathsf{Bel}\ charge = i)$.
- $\langle\langle MV \rangle\rangle \mathsf{Bel}\ \mathbf{E} \bigcirc hullDamaged$ — в одиночку агент MV не может исключить вариант негативного развития событий.
- $infront(cur_x, cur_y, orient) = nothing \Rightarrow \langle\langle MN, MV \rangle\rangle \mathsf{Bel}\ \mathbf{A} \bigcirc \neg hullDamaged$ — совместно агенты могут точнее прогнозировать будущее.

О желаниях агентов можно сформулировать следующие утверждения.

- $\langle\langle SB \rangle\rangle \mathsf{Des}\ \mathbf{A}\, G\, charge > 0 \wedge \neg batteryDamaged$ — целью агента SB является не допустить разряда аккумулятора и повреждения солнечных батарей.
- $\langle\langle MN \rangle\rangle \mathsf{Des}\ (\mathbf{A}\, G \neg armDamaged) \wedge \mathbf{A}\, \mathcal{F}_{t \leq 50} fitnes > 5$ — целью агента MN является не допустить повреждения манипулятора, а также собрать более десяти предметов не более чем за пятьдесят тактов.

- $\langle\langle MV\rangle\rangle$Des $\mathbf{A}\, G \neg hullDamaged$ — целью агента MV является не допустить повреждения корпуса.

Для всех трех агентов системы можно построить планы, удовлетворяющие следующим условиям.

- $\langle\langle SB, MN, MV\rangle\rangle$Intend $\mathbf{A}\, G charge > 0 \wedge \neg batteryDamaged$ — совместно все агенты планируют добиться того, что аккумулятор не разрядится и батареи не будут повреждены.

- $\langle\langle MN, MV\rangle\rangle$Intend $\mathbf{A}\, G \neg armDamaged \wedge \neg hullDamaged$ — агенты MN и MV планируют избежать повреждений.

- $\langle\langle MN, MV\rangle\rangle \mathbf{E}\, \mathcal{F}\, fitnes > 10$ — агенты MN и MV планируют собрать более десяти предметов, если будет такая возможность.

Таким образом, приведенный выше пример наглядно иллюстрирует богатые возможности логики спецификации мультиагентной системы по описанию различных свойств поведения системы в целом и отдельных её частей. Кроме того, пример демонстрирует возможности логики по спецификации ограничений на время реакции системы, а также по описанию различных свойств ментальных состояний агентов и коалиций.

Глава 7

Планирование в ограничениях

Одной из наиболее сложных задач мультиагентной системы является задача планирования действий для достижения поставленных целей. В данной главе предлагается алгоритм планирования в ограничениях, сформулированных в виде набора формул логики спецификации мультиагентных систем (глава 6) без использования операторов Bel , Des и Intend . Планы, полученные в результате работы предложенного алгоритма, представлены в виде сети взаимодействующих конечных автоматов, что соответствует модели плана, описанной в подразделе 3.1.1.

Предложенный алгоритм осуществляет поиск в пространстве планов, представляя план как множество, используя двоичные разрешающие диаграммы $OBDD$. Основная идея алгоритма заключается в рекурсивном уточнении плана от подформул к формуле через удаление из текущего плана действий, нарушающих формулу или, наоборот, через добавление действий, которые могут формулу реализовать. Таким образом, предложенный алгоритм построения плана во многом аналогичен алгоритму верификации формул. Основным отличием алгоритма планирования является то, что после построения плана для формулы, в некоторых случаях, необходимо произвести уточнение планов для подформул.

В приложении А.3 приведены доказательства полноты предложенных алгоритмов, а также анализ их алгоритмической сложности.

7.1. Существующие подходы к планированию

Проактивность является одним из основных свойств интеллектуального агента и подразумевает способность агента к построению *планов* своих действий на несколько шагов вперед. Следовательно, актуальной является задача

разработки эффективных алгоритмов планирования.

В настоящий момент существует большое количество различных подходов и алгоритмов планирования, большинство из которых позволяют решать задачу следующего вида:

- по данным S, A, R, F_0, G, V, где
  - S есть непустое конечное множество состояний системы;
  - A есть непустое конечное множество действий системы;
  - $R \subseteq S \times A \times S$ есть отношение переходов, описывающее возможные изменения состояния системы при выполнении определенных действий;
  - $F_0 \subseteq S$ есть множество *начальных* состояний;
  - $G \subseteq S$ есть множество *целевых* состояний;
  - $V \subseteq S$ есть множество *допустимых* состояний;
- найти такое множество пар $Plan \subseteq S \times A$, что для любой цепочки $\lambda = (s_1, s_2, \ldots) \in S^*$ такой, что:
  - $\lambda[0] \in F_0$;
  - для любого $0 \leq i < |\lambda|$ существует такое $a \in Plan(\lambda[i])$ (где $Plan(s) = \{a \in A \mid (a, s) \in Plan\}$), что $(\lambda[i], a, \lambda[i+1]) \in R$;

  выполнено:
  - существует такое $0 \leq j < |\lambda|$, что $\lambda[j] \in G$;
  - для любого $0 \leq k < j$ выполнено $\lambda[k] \in V$.

Таким образом, система, начав действовать в одном из состояний множества F_0 и выбирая в состоянии $s \in S$ действия из множества $Plan(s)$, когда-нибудь попадет в одно из состояний множества G, посещая при этом только состояния

из множества V. В случае если отношение $Plan$ является сериальным, то такой план является *сильным универсальным* планом. В случае, если условие "для любой цепочки λ ..." заменить на условие "существует такая цепочка λ, что ...", то получившийся сериальный план будет *слабым универсальным* планом [96].

Подходы к решению этой задачи можно условно разбить на три группы.

Классические алгоритмы. Алгоритмы, в основе которых лежит один из вариантов классического эвристического алгоритма поиска пути в графе A^* [141].

Символические алгоритмы. Алгоритмы, манипулирующие со множествами состояний, а не с отдельными состояниями, используя для этого особые структуры данных, в том числе и $OBDD$ [47, 96, 98].

Вероятностные алгоритмы. Алгоритмы этой группы рассматривают вероятности тех или иных изменений состояния и строят план, максимизирующий вероятность попадания в целевое состояние [65].

Другие подходы. За более чем 30 лет развития в области алгоритмов автоматического планирования выработалось достаточно много подходов, как уникальных, так и являющихся комбинацией других подходов. Краткий обзор наиболее значимых результатов этой группы мы проведем в разделе 7.1.6.

В изложенной выше постановке задачи планы строятся для достижения относительно простых целей — попасть в определенное состояние, проходя на своем пути только допустимые состояния. Более сложные планы можно строить, решая задачу синтеза управляющего супервизора (supervisor control synthesis) для дискретной системы событий (Discrete Event System, DES) [28, 100]. Этот подход мы более подробно рассмотрим в разделе 7.1.4.

Еще одним подходом, позволяющим строить более гибкие планы, является планирование с *темпорально расширенными целями* [31]. В этом случая цель плана описывается в виде выражения некоторой темпоральной логики, что позволяет описывать намного более сложные цели, чем просто указание целевого множества. Этот подход мы более подробно рассмотрим в разделе 7.1.5.

7.1.1. Эвристический поиск

Достаточно часто задача планирования сводится к задаче поиска пути в графе, узлы которого соответствуют состояниям системы S, а дуги — элемента отношения переходов $R \subseteq S \times A \times S$ (т.е. действиям системы и их результатам). При этом каждому плану соответствует некоторый *путь* в этом графе, который и нужно найти.

Одним из подходов к решению задачи поиска пути в графе является *эвристический поиск*. Основная идея этого метода достаточно проста — для каждой вершины графа (каждого состояния $s \in S$) определяется значение *эвристической функции* (эвристики) $h^*(s) \in \mathbb{R}$. Это значение характеризует удаленность данной вершины от цели (чем больше значение эвристической функции, тем дальше от цели находится вершина). Таким образом, для нахождения пути достаточно, начав в одном из начальных состояний, итеративно перемещаться в соседние состояния с наименьшими значениями эвристическою функции.

Основной проблемой такого подхода является построение эвристической функции. Как правило, точное построение эвристики невозможно за допустимое время, однако существует немало способов приближенного её построения на основании описания задачи [36, 121]. Кроме того, было предложено несколько алгоритмов, позволяющих строить и уточнять эвристику постепенно, а не единовременно. Примерами таких алгоритмов являются Real-Time A^* (RTA^*) и Learning Real-Time A^* ($LRTA^*$) [104].

Наиболее известной реализацией планировщика на основе эвристического

поиска является система Heuristic Search Planner (HSP) [35], представляющая собой целое семейство эвристических алгоритмов планирования. Другим интересным семейством эвристических планировщиков является Fast-Forward (FF) [79], регулярно занимающая высокие места на международном чемпионате планировщиков (International Planning Competition), проходящем ежегодно в рамках конференции International Conference on Automated Planning and Scheduling ($ICAPS$) [85].

7.1.2. Символическое планирование

Классические алгоритмы эвристического поиска рассматривают каждое состояние в отдельности, что затрудняет их применение для систем с большим числом состояний. Частично это проблема решается с помощью применения символических алгоритмов, оперирующих над множествами состояний (по аналогии с символическими алгоритмами, рассмотренными в главе 5.1).

Работа [95] предлагает способы построения планов, использующие бинарные разрешающие диаграммы и технику, во многом аналогичную технике верификации, описанной в разделе 5.1.3. Основу символических алгоритмов такого рода составляет операция вычисления *прообраза* (pre-image), по заданному множеству состояний $x \subseteq S$ определяющая такое множество состояний y, что из любого состояние $s \in y$ за одно действие можно попасть в некоторое состояние $s' \in x$. Операцию вычисления прообраза можно относительно просто и эффективно реализовать с помощью $OBDD$.

Таким образом, начав с множества целевых состояний, можно последовательно строить цепочку прообразов, пока рассмотренное множество не включит множество начальных состояний.

Примерами реализации символического планирования являются системы UMOP [99] и BIFROST [95], а также система MAX-BDD.

7.1.3. Вероятностное планирование

Важным ограничением описанных выше подходов является то, что они либо не допускают недетерминированного поведения системы вообще, либо допускают, но не позволяют учитывать вероятности того или иного развития ситуации[1]. Это ограничение преодолевается в подходах вероятностного планирования.

Как правило, в подобных подходах вместо отношения переходов $R \subseteq S \times A \times S$ используется Марковский процесс принятия решения (см. раздел 2.1.3), а задачей планирования является построение плана, реализующего цель с определенной вероятностью.

Большинство подобных планировщиков сводят задачу построения плана к задаче поиска максимума системы линейных функций с линейными ограничениями (эта задача широко известна под названием *задача линейного программирования*).

Для реализации планировщиков этой группы часто используют алгебраические разрешающие диаграммы ADD (см. 5.1.2), динамические Байесовские сети (dynamic Bayesian network) [51], а также разряженные матрицы (sparse matrix) [55]. Описание реализации вероятностного планировщика можно найти в работе [65].

7.1.4. Синтез супервизора

Задача синтеза супервизора для системы дискретных событий идейно близка задаче планирования. Система дискретных событий (Discrete Event System, DES) представляет собой четверку $M = (S, \Sigma, \sigma_M, s_0)$, где

- S есть непустое конечное множество состояний системы;

[1]Здесь следует оговорится, что некоторые работы по реализации механизмов работы с вероятностями в рамках символических подходов ведутся, например [97].

- Σ есть непустое конечное множество *событий*, которое состоит из двух непересекающихся множеств *контролируемых* событий Σ_c и *неконтролируемых* событий Σ_u ($\Sigma = \Sigma_c \cup \Sigma_u$, $\Sigma_c \cap \Sigma_u = \emptyset$);

- $\sigma_M : S \times 2^\Sigma \to S$ есть функция переходов, сопоставляющая текущему состоянию системы и множеству произошедших событий следующее состояние;

- $s_0 \in S$ есть начальное состояние системы.

Супервизором для системы дискретных событий M называется набор $Sp_M = (Y, \sigma_{Sp}, y_0)$, где

- Y есть непустое конечное множество состояний супервизора;

- $\sigma_{Sp} : Y \times S \times 2^\Sigma \to 2^Y$ есть функция переходов супервизора, сопоставляющая текущему состояний супервизора, текущему состоянию системы и множеству произошедших событий множество возможных следующих состояний супервизора (возможно пустое);

- $y_0 \in Y$ есть начальное состояние супервизора.

Таким образом, супервизор можно рассматривать как недетерминированный конечный автомат с множеством состояний Y и входным алфавитом $S \times 2^\Sigma$. Важным моментом является то, что функция σ_{Sp} может возвращать пустые множества состояний и таким образом *отключать* некоторые события. Событие $\varepsilon \in \Sigma$ является отключенным в состоянии системы $s \in S$ и состоянии супервизора $y \in Y$ тогда и только тогда, когда не существует такого подмножества событий $\Omega \subseteq \Sigma$, что $\varepsilon \in \Omega$ и $\sigma_{Sp}(y, s, \Omega) \neq \emptyset$. Супервизор является *совместимым по управлению* с системой M тогда и только тогда, когда он отключает только контролируемые события Σ_c (т.е. для любых $\varepsilon \in \Sigma_u$, $s \in S$ и $y \in Y$ существует такое множество $\Omega \subseteq \Sigma$, что $\sigma_{Sp}(y, s, \Omega) \neq \emptyset$).

Задача синтеза супервизора формулируется следующим образом: по заданной системе дискретных событий $M = (S, \Sigma, \sigma_M, s_0)$ и заданной формуле логики ветвящегося времени ϕ с интерпретацией пропозициональных символов $\pi : S \to 2^{Prop}$ (см. раздел 4.1.3) построить такой супервизор $Sp_M = (Y, \sigma_{Sp}, y_0)$, что $\langle S \times Y, R_{M\|Sp}, \pi\rangle, (s_0, y_0) \vDash \phi$, где $R_{M\|Sp}$ определяется следующим образом: $R_{M\|Sp} \triangleq \{(s, y, s', y') \in S \times Y \times S \times Y \mid \exists \Omega \subseteq \Sigma : s' = \sigma_M(s, \Omega) \quad and \quad y' \in \sigma_{Sp}(s, y, \Omega)\}$.

Моделируя действия системы с помощью контролируемых событий Σ_c и недетерминированные результаты действий с помощью неконтролируемых событий Σ_u можно свести задачу построения плана к задаче построения супервизора. При этом описать ограничения на искомый план можно будет намного более точно, используя всю выразительную мощность логики ветвящегося времени CTL или её расширенного варианта CTL^*.

Алгоритмы для синтеза супервизора, сводящие эту задачу к задаче проверки выполнимости формулы CTL, можно найти в работе [100]. Более простые алгоритмы ранее предлагались в работе [28], однако [100] содержит ряд опровергающих примеров для этих алгоритмов.

7.1.5. Темпорально расширенные цели

Другим подходом, позволяющим строить более гибкие планы, является планирование с темпорально расширенными целями. В этом случае ограничения на план, как правило, выражаются формулой логики линейного времени LTL а сам план строится как последовательность действий (а не как отношение состояние–действие). Большинство предлагаемых подходов к решению этой задачи требуют детерминированного поведения системы и полной наблюдаемости состояний [31, 102], однако есть и работы, позволяющие решать эту задачу для более общего случая [145, 146].

Один из первых алгоритмов, решающих эту задачу был, предложен в рабо-

те [31] и реализован в планировщике $TLPlan$. Основная идея этого алгоритма заключается в построении *прогрессии* формулы по особым правилам и поиску состояний, удовлетворяющих той части формулы, которая соответствует текущему моменту.

Символический алгоритм планирования с темпорально расширенными целями был предложен в работе [145] и реализован в планировщике Model Based Planner (MBP) [120]. Этот алгоритм так же использует прогрессию формулы, однако в фазе построения плана оперирует множествами состояний, а не отдельными состояниями.

7.1.6. Другие подходы

Интересным подходом, дающим хорошие результаты на практике, является подход сведения задачи планирования к задаче проверки выполнимости пропозициональной формулы, примененный в планировщике $SATPLAN$ [160], а также в планировщике $MAXPLAN$ [118], поддерживающем, в том числе, и вероятностное планирование. Оба этих планировщика разделили первое место на международном соревновании планировщиков в рамках конференции ICAPS-2006 [85] в категории оптимальных планировщиков (строящих оптимальный план по достижению цели).

Подход разбиения задачи на подзадачи часто позволяет значительно ускорить процесс построения плана. Этот подход был использован в планировщике $SGPlan$ [46], занявшем первое место среди *неоптимальных* планировщиков (строящих какой-то план, не обязательно оптимальный, для достижения поставленной цели) на ICAPS-2006. Похожий подход, получивший название иерархическая сеть задач (Hierarchical Task Network, HTN), был с успехом применен в планировщике SHOP2 [163].

7.2. Постановка задачи

Задачу планирования в ограничениях сформулируем следующим образом.

Определение 22 *Для заданного набора* $\langle M, \widetilde{s}, \widetilde{env}, \widetilde{goal}\rangle$, *где*

- $M = \langle MAS, Domain, \pi_s, \pi_a, \{\Phi_i\}_{i\in\{0,\ldots,n_f\}}, \{\Pi_i\}_{i\in\{0,\ldots,n_p\}}, \gamma\rangle$ *есть модель логики спецификации мультиагентной системы (определение 19);*

- $\widetilde{s} = \{\varepsilon_i\}_{i\in\{1,\ldots,n_s\}}$ *есть спецификация начального состояния, где* ε_i *есть базовая формула состояния (уравнение 4.29), не содержащая оператора* $[\,]$;

- $\widetilde{env} = \{\varepsilon_i \Rightarrow [\alpha_i]\varepsilon'_i\}_{i\in\{1,\ldots,n_e\}}$ *есть спецификация внешней среды, где* α_i *есть формула действия, а* ε_i *и* ε'_i *есть базовые формулы состояний (уравнение 4.29), не содержащие оператора* $[\,]$;

- $\widetilde{goal} = \{\phi_i\}_{i\in\{1,\ldots,n_g\}}$, *где* ϕ_i *есть формула состояния (уравнение 6.1), не содержащая операторов* Bel , Des *и* Intend ;

найти множество $S_0 \subseteq S$, *отношение* $env \subseteq S \times ACS \times S$ *и план* $plan_{ag_j} = (P_{ag_j}, A_{ag_j}, Sig_{ag_1}\times\ldots\times Sig_{ag_n}, Sig_{ag_j}, I_{ag_j,pln}, send_{ag_j,pln}, \sigma_{ag_j,pln}, i_{ag_j,pln,0})$ *для каждого из агентов коалиции* $ag_j \in AG$, *удовлетворяющие следующим условиям.*

1. S_0 *есть максимальное множество такое, что для любых* $s \in S_0$ *и* $\varepsilon_i \in \widetilde{s}$ *выполнено* $(M, i), AG, S \times ACS \times S, (s, i_{pln}) \vDash_s \varepsilon_i$, *где* i *есть некоторое ментальное состояние коалиции всех агентов, а* i_{pln} *есть некоторое состояние плана этой коалиции*[2].

2. env *есть максимальное отношение такое, что для любых* $s \in S$ *и* $(\varepsilon_i \Rightarrow [\alpha_i]\varepsilon'_i) \in \widetilde{env}$ *выполнено* $(M, i), AG, env_s, (s, i_{pln}) \vDash_s \varepsilon_i \Rightarrow [\alpha_i]\varepsilon'_i$.

[2]Заметим, что, так как формулы ε_i и ε'_i являются *базовыми* формулами состояний, конкретная коалиция и её состояние не влияет на их интерпретацию

3. *Для любого* $s \in S_0$ *и* $\phi_i \in \widetilde{goal}$ *выполнено* $(M, i^{AG}), AG, env_s, (s, i^{AG}_{0,pln}) \models_s \phi_i$, *где* i^{AG} *есть некоторое ментальное состояние коалиции всех агентов, планы в котором соответствуют найденным планам* $plan_{ag_j}$.

Таким образом, основной задачей алгоритма является нахождение таких планов $plan_{ag_j}$ для каждого из агентов $ag_j \in AG$, что мультиагентная система, взаимодействуя со внешней средой env, удовлетворяющей спецификации $\widetilde{env}$ и обладающей максимальным недетерминизмом, начав взаимодействие в одном из состояний $s \in S_0$, удовлетворяющем спецификации $\widetilde{s}$, реализует все цели $\widetilde{goal}$.

7.3. Основная структура алгоритма

Решение задачи 22 можно разбить на следующие этапы.

1. Проверить корректность вложенности кванторов коалиции.
2. Спустить все отрицания в формулах $\phi_i \in \widetilde{goal}$ за темпоральные операторы, используя правила двойственности (уравнения 4.42–4.47 и 6.2) и развернуть все ограниченные операторы $\psi_1 \mathcal{U}_{\leq t} \psi_2$ и $\psi_1 \mathcal{W}_{\leq t} \psi_2$ при $t < \infty$ используя правила рекурсивного вычисления 4.52–4.55.
3. Построить соответствующие множества env_s и S_0.
4. Определить структуру множества состояний плана I_{pln} на основе структуры формул ϕ_i.
5. Для каждого из агентов ag системы построить отношение $pln \subseteq P \times I_{pln} \times I_{pln} \times A_{ag}$, где $P = \underset{ag' \in AG}{\times} P_{ag'}$.
6. На основе полученного набора отношений сформировать планы для агентов системы.

7.3.1. Построение S_0 и env

Множество S_0 строится как решение задачи верификации для формулы $\bigwedge\limits_{i\in\{1,\ldots,n_s\}} \varepsilon_i$:

$$S_0 = \left[\bigwedge_{i\in\{1,\ldots,n_s\}} \varepsilon_i \right]. \tag{7.1}$$

Отношение $env \subseteq S \times ACS \times S$ может быть построено как пересечение множеств env_i ($env = \bigcap\limits_{i\in\{1,\ldots,n_e\}} env_i$), где каждое из env_i строится относительно $\varepsilon_i \Rightarrow [\alpha_i]\varepsilon'_i$ следующим образом:

$$env_i \triangleq ([\varepsilon_i] \times [\alpha_i] \times [\varepsilon'_i]) \cup (S \setminus [\varepsilon_i] \times ACS \times S) \cup (S \times ACS \setminus [\alpha_i] \times S). \tag{7.2}$$

При выполнении начальных условий для состояния ($s \in [\varepsilon'_i]$) и действия ($a \in [\alpha_i]$), результат определен правилом (для любого $s' \in [\varepsilon'_i]$ выполнено $(s, a, s') \in env_i$ и наоборот). Если же хотя бы одно из начальных условий нарушено ($s \notin [\varepsilon'_i]$ или $a \notin [\alpha_i]$), то результаты непредсказуемы (для любого $s' \in S$ выполнено $(s, a, s') \in env_i$).

7.3.2. Построение I_{pln}

Структура множества I_{pln} определяется исключительно структурой формул $\phi_i \in \widetilde{goal}$ следующим образом:

$$I_{pln} = \underset{\psi\in mark}{\times} \{\psi, \neg\psi\}, \tag{7.3}$$

где $mark \subseteq 2^{\phi}$ есть множество формул ψ, входящих в хотя бы одну из подформул вида $\psi \vee \varphi$, $\varphi \vee \psi$, $\bigcirc\psi$, $\varphi\, \mathcal{U}_{\leq t} \psi$ или $\varphi\, \mathcal{W}_{\leq t} \psi$ одной из формул ϕ_i.

Основная роль внутреннего состояния плана при планировании с темпорально расширенными целями — отслеживать под-дерево формулы, которое реализуется в данный момент. Поэтому в каждый момент времени состояние плана

представляет собой набор маркеров, определяющих активен ли в данный момент план для реализации определенных подформул. Маркеры используются для подформул следующего вида.

- Для операторов "или" ($\varphi \vee \psi$). В этом случае маркер добавляется для обеих частей подформулы для того, чтобы отличать ситуации, в которых действует план для первой подформулы (маркер отсутствует), от ситуаций, в которых действует план для второй подформулы (маркер присутствует) или план для обеих подформул сразу. В этом случае имеет смысл добавлять маркер только если хотя бы одна из формул, содержит хотя бы один темпоральный оператор.

- Для темпоральных операторов $\bigcirc\psi$, $\varphi \; \mathcal{U}_{\leq t} \psi$ или $\varphi \; \mathcal{W}_{\leq t} \psi$ добавляется маркер для формулы, находящейся "в будущем" (в правой части оператора) и используется для того, чтобы отличать ситуацию, когда план движется к состоянию, из которого достижима ψ (маркер не присутствует), от ситуации, когда план реализует непосредственно ψ (маркер присутствует).

- Для темпорального оператора $\varphi \; \mathcal{U}_{\leq t} \psi$ добавляется маркер для формул в левой части (φ), если она содержит хотя бы один темпоральный оператор. Этот маркер позволяет отличать ситуацию, когда агенту необходимо заботиться о выполнении свойства φ (маркер присутствует), от ситуации, когда сразу был активирован план для построения ψ и заботиться о φ не надо (маркер отсутствует).[3]

[3]Вторая ситуация возникает в том случае, если формула ψ верна для текущего состояния и перед ним может не быть ни одного состояния с выполненным φ, тогда как первая ситуация возникает в том случае, если формула ψ выполняется только через несколько шагов и при этом на всех предыдущих состояниях вплоть до текущего должна выполняться формула φ. В итоге, в первой ситуации план для φ был активирован и должен быть реализован.

Помимо добавленных таким образом маркеров, множество I_{pln} содержит дополнительный элемент $i_{0,pln}$, используемый для обозначения начального состояния плана.

7.3.3. Построение общего плана

Построение общего плана является основным этапом алгоритма и реализуется с помощью рекурсии по структуре формулы $\bigwedge_{\phi_i \in \widetilde{goal}} \phi_i$. При этом модель M фиксирована, а на каждом этапе решается задача следующего вида.

Определение 23 *Для данных:*

- *Формулы ψ;*
- *Текущей коалиции $C \subseteq AG$;*
- *Описания внешней среды $env \subseteq S \times A^C \times S$;*
- *Верхней границы свободы $sup_{ag} \subseteq P^C \times I_{pln} \times I_{pln} \times A_{ag}$ для каждого из агентов коалиции $ag \in C$;*

найти множество пар $ES_0 \subseteq P^C \times I_{pln}$ и набор отношений $pln_{ag} \subseteq sup_{ag}$ для каждого из агентов коалиции C, такие, что для любой пары $(p, i_{pln}) \in ES_0$ и для любого $s \in p$ выполнено $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \psi$, где ментальное состояние $i^C_{\{pln_{ag}\}}$ содержит планы агентов коалиции $plan_{ag}$, построенные следующим образом.

- *$Sig_{ag} = P_{ag}$, $send_{ag}(p_{ag}, i_{pln}, ag') = p_{ag}$ — посылаемые каждым агентом сигналы соответствуют его восприятию.*
- *$\sigma_{ag,pln} = pln_{ag}$ — отношение переходов плана совпадает с отношением pln_{ag} агента.*

- $i_{ag,0,pln} = i_{0,pln}$ — *начальное состояние совпадает с выделенным элементов* $i_{0,pln} \in I_{pln}$. *Так как истинность формулы зависит от определенного состояния плана* i_{pln}, *то начальное состояние плана, в данном случае, не имеет значения.*

Функцию, решающую задачу 23, обозначим $find(\psi, C, env_s, \{sup_{ag}\})$. При этом для решения исходной задачи (построения общего плана) необходимо вычислить $find(\bigwedge\limits_{\phi_i \in \widetilde{goal}} \phi_i, AG, env, \{P^{AG} \times (I_{pln} \setminus \{i_{0,pln}\}) \times (I_{pln} \setminus \{i_{0,pln}\}) \times A_{ag}\}_{ag \in AG})$ и проверить, что $see^{AG}(S_0) \subseteq ES_0|_{P^{AG}}$.

7.3.4. Выделение отдельных планов

После того, как функция $(ES_0, \{pln_{ag}\}) = find(\bigwedge\limits_{\phi_i \in \widetilde{goal}} \phi_i, AG, env, \{i_{0,pln}\}) \times (I_{pln} \setminus \{i_{0,pln}\}) \times A_{ag}\}_{ag \in AG})$ вычислена, выделить планы агентов из полученного набора планов достаточно легко.

Сначала добавим к каждому из полученных множеств правил отдельные правила для обработки начального состояния: $pln'_{ag} = \{(p, i_{0,pln}, i'_{pln}, a) \in P^{AG} \times I_{pln} \times I_{pln} \times A_{ag} \mid \exists i''_{pln} \in I_{pln} : (p, i''_{pln}) \in ES_0 \; and \; (p, i''_{pln}, i'_{pln}, a) \in pln_{ag}\}$. С точки зрения операций над $OBDD$ для получения искомого множества нужно вычислить пересечение $ES_0 \times I_{pln} \times A_{ag} \cap pln_{ag}$, устранить в нем все вершины, соответствующие первому состоянию, после чего вычислить пересечение с множеством $P^{AG} \times \{i_{0,pln}\} \times I_{pln} \times A_{ag}$.

Для каждого агента $ag \in AG$ план строится на основе полученного для него отношения pln_{ag} следующим образом.

- $Sig_{ag} = P_{ag}$, $send_{ag}(p_{ag}, i_{pln}, ag') = p_{ag}$ — посылаемые каждым агентом сигналы соответствуют его восприятию.

- $\sigma_{ag,pln} = pln'_{ag} \cup pln_{ag}$ — отношение переходов плана совпадает с отношением pln_{ag}, расширенным правилами для начального состояния.

- $i_{ag,0,pln} = i_{0,pln}$ — начальное состояние совпадает с найденным на этапе построения I_{pln}.

Заметим, что полученный план можно оптимизировать соответствующим образом, уменьшив количество возможных сигналов для агентов. Однако эта оптимизация выходит за рамки данной работы.

7.4. Построение планов для формул

Основную сложность при построении плана составляет рекурсивное вычисление функции $find(\psi, C, env, \{sup_{ag}\})$, при котором могут встретиться формулы, не содержащие темпоральных операторов или формулы вида $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$, $\langle\langle D \rangle\rangle \psi_1$, $\mathbf{A} \bigcirc \psi_1$, $\mathbf{E} \bigcirc \psi_1$, $\neg \mathbf{E} \bigcirc true$, $\mathbf{A}\, \psi_1\, \mathcal{U}_{\leq t} \psi_2$, $\mathbf{E}\, \psi_1\, \mathcal{U}_{\leq t} \psi_2$, $\mathbf{A}\, \psi_1\, \mathcal{W}_{\leq t} \psi_2$ и $\mathbf{E}\, \psi_1\, \mathcal{W}_{\leq t} \psi_2$.

Заметим, что отрицание может встречаться только в формулах, не содержащих темпоральных операторов, и в формулах вида $\neg \mathbf{E} \bigcirc true$, так как все отрицания были спущены на подготовительном этапе.

Для формул ψ, не содержащих темпоральных операторов, функция $find$ вычисляется следующим образом:

$$find(\psi, C, env, \{sup_{ag}\}) = (P^C \setminus see^C(S \setminus [\psi]_{env,C}\,|_S) \times I_{pln}, \{sup_{ag}\}). \quad (7.4)$$

Планы агентов совпадают с максимально допустимыми ($\{sup_{ag}\}$), а множество начальных восприятий определяется множеством состояний, удовлетворяющих ψ вместе со всем своим классом эквивалентности относительно восприятия коалиции ($P^C \setminus see^C(S \setminus [\psi]_{env,C}\,|_S)$).

Доказательства корректности предложенных алгоритмов и анализ их сложности приведен в приложении на странице 213.

7.4.1. Формулы вида $\psi_1 \wedge \psi_2$

Если $\psi = \psi_1 \wedge \psi_2$, то общий результат мы будем строить как пересечение результатов для подформул. Однако, результат простого пересечения не обязательно будет искомым, поэтому это пересечение используется в качестве нового ограничения $\{sup_{ag}\}$ в повторном вычислении планов для подформул. Алгоритм продолжается до достижения неподвижной точки, как показано в алгоритме 24.

Алгоритм 24 (Вычисление $find(\psi_1 \wedge \psi_2, C, env, \{sup_{ag}\})$**)**
$\{pln_{ag}\} := \{sup_{ag}\}$; $ES_0 := P^C \times I_{pln}$;
do
$(ES_0^1, \{pln_{ag}^1\}) := find(\psi_1, C, env, \{pln_{ag}\})$;
$(ES_0^2, \{pln_{ag}^2\}) := find(\psi_2, C, env, \{pln_{ag}\})$;
$ES_0 := ES_0^1 \cap ES_0^2$;
$\{pln_{ag}\} := \{pln_{ag}^1 \cap pln_{ag}^2\}$;
while $\{pln_{ag}^1\} \neq \{pln_{ag}^2\}$;
return $(ES_0, \{pln_{ag}\})$;

7.4.2. Формулы вида $\psi_1 \vee \psi_2$

При вычислении $find(\psi_1 \vee \psi_2, C, env, \{sup_{ag}\})$ сначала вычислим планы для подформул $(ES_0^1, \{pln_{ag}^1\}) = find(\psi_1, C, env, \{sup_{ag}\})$ и $(ES_0^2, \{pln_{ag}^2\}) = find(\psi_2, C, env, \{sup_{ag}\})$, а также план, реализующий обе подформулы сразу $(ES_0^{12}, \{pln_{ag}^{12}\}) = find(\psi_1 \wedge \psi_2, C, env, \{sup_{ag}\})$. Затем для каждого агента $ag \in C$ вычислим pln_{ag} следующим образом.

- Ограничим полученные от подформул результаты следующим образом: $pln_{ag}^1 = pln_{ag}^1 \cap P^C \times C(\psi_1) \cap C(\neg\psi_2) \times C(\psi_1) \cap C(\neg\psi_2) \times A_{ag}$, $pln_{ag}^2 = pln_{ag}^2 \cap P^C \times C(\neg\psi_1) \cap C(\psi_2) \times C(\neg\psi_1) \cap C(\psi_2) \times A_{ag}$, $pln_{ag}^{12} = pln_{ag}^{12} \cap P^C \times C(\psi_1) \cap C(\psi_2) \times C(\psi_1) \cap C(\psi_2) \times A_{ag}$.

- Вычислим искомое отношение как $pln_{ag} = pln_{ag}^1 \cup pln_{ag}^2 \cup pln_{ag}^{12}$.

Аналогичным образом вычисляется и множество начальных состояний ES_0:

$$\begin{aligned} ES_0 \;=\; & ES_0^1 \cap (P^C \times (C(\psi_1) \cap C(\neg\psi_2))) \cup ES_0^2 \cap (P^C \times (C(\neg\psi_1) \cap C(\psi_2))) \\ & \cup ES_0^{12} \cap (P^C \times (C(\psi_1) \cap C(\psi_2))). \end{aligned}$$

7.4.3. Формулы вида $\langle\langle D\rangle\rangle\psi_1$

При обработке квантора коалиций перед рекурсивным вызовом необходимо проделать следующие действия:

1. расширить описание внешней среды env до $env|_D$, устранив ($v = high(v) \cup low(v)$) все вершины, соответствующие восприятию и действиям агентов вне коалиции;

2. сузить верхнюю границу свободы $\{sup_{ag}\}_{ag \in C}$ до $\{sup'_{ag}\}_{ag \in D}$, заменив во всех описаниях вершины, соответствующие восприятию агентов вне коалиции на *пересечение* дочерних вершин ($v = high(v) \cap low(v)$), а также исключив ограничения на действия агентов вне коалиции.

Кроме того, после вычисления рекурсивного вызова $(ES_0, \{pln_{ag}\}_{ag \in D}) = find(\psi_1, D, env', \{sup'_{ag}\}_{ag \in D})$, необходимо расширить результат отношениями pln_{ag} для агентов $ag \in C \setminus D$, взяв в качестве этих отношений sup_{ag}. Учитывая то, что в представлении $OBDD$ полученные множества $ES_0 \subseteq P^D \times I_{pln}$ и $pln_{ag} \subseteq P^D \times I_{pln} \times I_{pln} \times A_{ag}$ можно трактовать как множества $ES_0 \subseteq P^C \times I_{pln}$ и $pln_{ag} \subseteq P^C \times I_{pln} \times I_{pln} \times A_{ag}$, не зависящие от переменных, соответствующих восприятию агентов вне D, для преобразования результатов к нужному формату дополнительных действий не требуется.

7.4.4. Вычисление прообраза

Для обработки темпоральных операторов введем функцию вычисления прообраза $Pre_{\mathbf{E}} : 2^{P^C \times I_{pln}} \times 2^{P^C \times ACS \times P^C} \times 2^{P^C \times I_{pln} \times I_{pln} \times ACS} \to 2^{P^C \times I_{pln} \times I_{pln} \times ACS}$, со-

поставляющую множеству пар $x \subseteq P^C \times I_{pln}$, описанию внешней среды $env^C \subseteq P^C \times ACS \times P^C$ и ограничению на выбор действий $sup \subseteq P^C \times I_{pln} \times I_{pln} \times ACS$ множество таких правил $\{(p, i_{pln}, i'_{pln}, a)\} \subseteq P^C \times I_{pln} \times I_{pln} \times ACS$, что выбор действия не противоречит sup, и одним из возможных результатов действия является переход в одно из состояний x:

$$Pre_{\mathbf{E}}(x, env^C, sup) \triangleq \{(p, i_{pln}, i'_{pln}, a) \in sup \mid \exists p' \in P^C : (p', i'_{pln}) \in x \; and \; (p, a, p') \in env^C\}. \quad (7.5)$$

При этом описание $env^C \subseteq P^C \times ACS \times P^C$ строится на основе $env \subseteq S \times ACS \times S$ и $see^C : S \to P^C$ следующим образом: $env^C \triangleq \{(p, a, p') \in P^C \times ACS \times P^C \mid \exists s, s' \in S : p = see(s) \; and \; p' = see(s') \; and \; (s, acs, s') \in env\}$.

С точки зрения операций над $OBDD$ прообраз можно вычислить следующим способом:

1. вычислить прообраз x относительно env^C как $pre^1 = (env^C \cap (P^C \times ACS \times x|_{P^C}))|_{P^C \times ACS}$;
2. вычислить прообраз x относительно sup как $pre^2 = (sup \cap (P^C \times I_{pln} \times x|_{I_{pln}} \times ACS))$;
3. вычислить пересечение этих прообразов $pre = (pre^1 \times I_{pln} \times I_{pln}) \cap pre^2$.

Кроме того, введем функцию строгого прообраза $Pre_{\mathbf{A}}(x, env, sup)$, возвращающую только те правила, результаты которых гарантированно ведут в x:

$$Pre_{\mathbf{A}}(x, env, sup) \triangleq Pre_{\mathbf{E}}(x, env, sup) \setminus Pre_{\mathbf{E}}(P^C \times I_{pln} \setminus x, env, sup). \quad (7.6)$$

Для реализации обработки темпоральных операторов необходимо определить методы преобразования раздельных ограничений $\{sup_{ag}\}$ в общее ограничение sup, а также преобразования общего результата $y \subseteq P^C \times I_{pln} \times I_{pln} \times ACS$ к представлению, удобному для конкретного агента $y|_{ag} \subseteq P^C \times I_{pln} \times$

$I_{pln} \times A_{ag}$. Сужение результата можно сделать с помощью проекции ($y|_{ag} \triangleq y|_{P^C \times I_{pln} \times I_{pln} \times A_{ag}}$), а обобщение ограничения вычисляется как пересечение расширенных индивидуальных ограничений:

$$sup \triangleq \bigcap_{sup_{ag} \in \{sup_{ag}\}} (sup_{ag} \times \underset{ag' \in AG \setminus \{ag\}}{\times} A_{ag'}). \tag{7.7}$$

Кроме того, необходимо определить способ построения env^C по env. Это преобразование можно вычислить используя представление восприятия как отношения $see^C \subseteq S \times P^C$ следующим образом: $env^C = ((P^C \times env \times P^C) \cap (see^C \times ACS \times see^C))|_{P^C \times ACS \times P^C}$. При этом для построения $see^C \subseteq S \times P^C$ достаточно расширить каждую компоненту в представлении функции $see^C_{\mathbb{B}} : \mathbb{B}^{n_S} \to \mathbb{B}^{\sum\limits_{ag \in C} n_{P_{ag}}}$ двумя дополнительными терминальными вершинами, описывающими соответствующий бит P^C, после чего построить пересечение всех компонент. Заметим, что нет необходимости строить env^C каждый раз — достаточно в начале работы алгоритма построить env^{AG}, после чего на каждом сужении коалиции устранять вершины, представляющие восприятие агентов вне коалиции.

Для упрощения выкладок будем опускать эти преобразования в тех случаях, когда это не ведет к двусмысленности.

7.4.5. Формулы вида $\mathbf{A} \bigcirc \psi_1$, $\mathbf{E} \bigcirc \psi_1$ и $\neg\mathbf{E} \bigcirc true$

Функцию $find(\mathbf{A} \bigcirc \psi_1, C, env, \{sup_{ag}\})$ можно вычислить, взяв строгий прообраз от состояний, полученных из $find(\psi_1, C, env, \{sup_{ag}\})$, после чего объединить результат для $\mathbf{A} \bigcirc \psi_1$ с результатом для ψ_1, разделив их маркерами $\neg\psi_1$ и ψ_1, как показано в алгоритме 25, где $D(\{pln_{ag}\}) \triangleq \bigcap\limits_{ag \in C} pln_{ag}|_{P^C \times I_{pln}}$ есть область определения плана.

Значение $find(\mathbf{E} \bigcirc \psi_1, C, env, \{sup_{ag}\})$ вычисляется аналогичным образом (алгоритм 26), но с использованием функции $Pre_{\mathbf{E}}$, а также без наложения дополнительных ограничений на план — ведь в данном случае достаточно только того, чтобы правильным был хотя бы один выбор из списка.

Алгоритм 25 (Вычисление $find(\mathbf{A} \bigcirc \psi_1, C, env, \{sup_{ag}\})$**)**
$(ES_0^1, \{pln_{ag}^1\}) := find(\psi_1, C, env, \{sup_{ag}\});$
$ES_0^1 := ES_0^1 \cap (P^C \times C(\psi_1));$
$\{pln_{ag}^1\} := \{pln_{ag}^1 \cap (P^C \times C(\psi_1) \times C(\psi_1) \times A_{ag})\};$
$\{pln_{ag}\} := Pre_{\mathbf{A}}(ES_0^1, env, \{sup_{ag} \cap P^C \times C(\neg\psi_1) \times C(\psi_1) \times A_{ag}\});$
$ES_0 := D(\{pln_{ag}\});$
$\{pln_{ag}\} := \{pln_{ag} \cup pln_{ag}^1\}$
return $(ES_0, \{pln_{ag}\});$

Алгоритм 26 (Вычисление $find(\mathbf{E} \bigcirc \psi_1, C, env, \{sup_{ag}\})$**)**
$(ES_0^1, \{pln_{ag}^1\}) := find(\psi_1, C, env, \{sup_{ag}\});$
$ES_0^1 := ES_0^1 \cap (P^C \times C(\psi_1));$
$\{pln_{ag}^1\} := \{pln_{ag}^1 \cap (P^C \times C(\psi_1) \times C(\psi_1) \times A_{ag})\};$
$\{pln_{ag}\} := Pre_{\mathbf{E}}(ES_0^1, env, \{sup_{ag}\});$
return $(D(\{pln_{ag}\}), \{(sup_{ag} \cap (P^C \times C(\neg\psi_1) \times I_{pln} \times A_{ag})) \cup pln_{ag}^1\});$

Для вычисления $find(\neg\mathbf{E} \bigcirc true, C, env, \{sup_{ag}\})$ необходимо найти такое множество пар ES_0 и набор правил $\{pln_{ag}\}$, что ни для одной пары в ES_0 не задано ни одного правила. Очевидным способом добиться искомого результата является пара $(P^C \times I_{pln}, \{pln_{ag} = \emptyset\})$.

7.4.6. Формулы вида $\mathbf{A}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$ и $\mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$

Функция $find(\mathbf{A}\,\psi_1\;\mathcal{U}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$ вычисляется через поиск двух неподвижных точек (алгоритм 27): внутренней точки, вычисляемой аналогично неподвижной точке в алгоритмах верификации для оператора $\mathbf{A}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$, и внешней точки, вычисляемой через ограничение плана для ψ_1 с учетом построенного при вычислении внутренней точки плана как ограничения.

Алгоритм 27 начинается с вычисления $(ES_0^1, \{pln_{ag}^1\})$ как плана для подформулы ψ_1 с учетом переданного ограничения $\{sup_{ag}\}$, после чего начинается поиск внешней неподвижной точки.

Каждая итерация вычисления внешней неподвижной точки начинается с вычисления $(ES_0^2, \{pln_{ag}^2\})$ как плана для ψ_2, при этом в качестве ограниче-

Алгоритм 27 (Вычисление $find(\mathbf{A}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$**)**
$(ES_0^1, \{pln_{ag}^1\}) := find(\psi_1, C, env, \{sup_{ag} \cap (P^C \times C(\psi_1) \times C(\psi_1) \times A_{ag})\})$;
do
$(ES_0^2, \{pln_{ag}^2\}) := find(\psi_2, C, env, \{pln_{ag}^1 \cap (P^C \times C(\psi_2) \times C(\psi_2) \times A_{ag})$
$\cup sup_{ag} \cap (P^C \times C(\psi_2, \neg\psi_1) \times C(\psi_2, \neg\psi_1) \times A_{ag})\})$;
$ES_0 := ES_0^2 \cap (P^C \times C(\psi_2))$; $\{pln_{ag}\} := \{pln_{ag}^2\}$; $EF = \emptyset$;
while $EF \neq ES_0$ ***do***
$EF := ES_0$;
$\{pln_{ag}\} := \{pln_{ag} \cup (Pre_{\mathbf{A}}(ES_0, env, \{pln_{ag}^1\})|_{ag}$
$\cap (ES_0^1 \setminus ES_0) \times I_{pln} \times A_{ag})\}$;
$ES_0 = ES_0 \cup (D(\{pln_{ag}\}) \cap (P^C \times C(\psi_1)))$;
od;
$(ES_0^1, \{pln_{ag}^1\}) := find(\psi_1, C, env, \{pln_{ag} \cap (P^C \times C(\psi_1) \times C(\psi_1) \times A_{ag})\})$;
while $\{pln_{ag} \cap (P^C \times C(\psi_1) \times C(\psi_1) \times A_{ag})\} \neq \{pln_{ag}^1\}$;
return $(ES_0, \{pln_{ag}\})$;

ния используется помеченная маркером ψ_2 часть плана для ψ_1, объединенная с верхней границей $\{sup_{ag}\}$, помеченной маркерами ψ_2 и $\neg\psi_1$. Таким образом, план для первой подформулы в некоторых случаях накладывает ограничения на план для второй подформулы. Например, для формулы $\mathbf{A}\,(\mathbf{A}\,G\psi_1)\,\mathcal{U}_{\leq t}\psi_2$ ограничение $\mathbf{A}\,G\psi_1$, в соответствии с семантическим правилом 8, распространяется и на состояния после достижения ψ_2.

Вычисленные для ψ_2 начальные состояния (с учетом разметки) и план используются в качестве начальных данных для поиска внутренней неподвижной точки ES_0 и $\{pln_{ag}\}$. Каждая итерация вычисления внутренней неподвижной точки заключается в расширении $\{pln_{ag}\}$ теми правилами из плана $\{pln_{ag}^1\}$, которые гарантировано приводят в одно из состояний ES_0 $(Pre_{\mathbf{A}}(ES_0, env, \{pln_{ag}^1\}))$ и левые части которых не выходят за рамки не обработанной до сих пор части ES_0^1 $(ES_0^1 \setminus ES_0)$. После этого множество ES_0 расширяется в соответствии с вновь добавленными правилами. Вычисление внутренней неподвижной точки завершается, когда множество ES_0 перестает расти.

После того, как внутренняя неподвижная точка вычислена и план $\{pln_{ag}\}$

построен, план $\{pln^1_{ag}\}$ для ψ_1 уточняется с учетом части $\{pln_{ag}\}$, помеченной маркером ψ_1, в качестве ограничения. Вычисление внешней неподвижной точки завершается в тот момент, когда планы $\{pln_{ag}\}$ и $\{pln^1_{ag}\}$ прекратят уменьшаться и сойдутся к единому значению в части, помеченной маркером ψ_1.

Алгоритм 28 (Вычисление $find(\mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$**)**
$(ES^1_0, \{pln^1_{ag}\}) := find(\psi_1, C, env, \{sup_{ag} \cap (P \times C(\psi_1) \times C(\psi_1) \times A_{ag})\})$;
$(ES^2_0, \{pln^2_{ag}\}) := find(\psi_2, C, env, \{pln^1_{ag} \cap (P \times C(\psi_2) \times C(\psi_2) \times A_{ag})$
$\cup sup_{ag} \cap (P \times C(\psi_2, \neg\psi_1) \times C(\psi_2, \neg\psi_1) \times A_{ag})\})$;
$ES_0 := ES^2_0 \cap (P \times C(\psi_2))$; $\{pln_{ag}\} := \{pln^2_{ag}\}$; $EF = \emptyset$;
while $EF \neq ES_0$ ***do***
$EF := ES_0$;
$\{pln_{ag}\} := \{pln_{ag} \cup (Pre_{\mathbf{E}}(ES_0, env, \{pln^1_{ag}\})|_{ag} \cap ES^1_0 \times I_{pln} \times A_{ag})\}$***;***
$ES_0 = ES_0 \cup D(\{pln_{ag}\})$***;***
od;
return $(ES_0, \{pln^1_{ag} \cup pln^2_{ag}\})$;

Функция $find(\mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$ вычисляется (алгоритм 28) аналогичным образом. Основным отличием алгоритма 28 от алгоритма 27 является то, что он вычисляется с помощью только одной неподвижной точки, соответствующей внутренней точке алгоритма 27. Это обусловлено тем, что при поиске *возможного* пути не накладывается ограничения на план агента (главное, чтобы в плане были соответствующие действия). Кроме того, при вычислении неподвижной точки алгоритм 28 использует слабый прообраз $Pre_{\mathbf{E}}$ вместо строгого прообраза $Pre_{\mathbf{A}}$. Конечным результатом алгоритма является объединение планов $\{pln^1_{ag} \cup pln^2_{ag}\}$ и найденное в результате вычисления неподвижной точки множество состояний ES_0 таких, из которых *существует* путь в одно из состояний ES^2_0.

7.4.7. Формулы вида $\mathbf{A}\,\psi_1\,\mathcal{W}_{\leq t}\psi_2$ и $\mathbf{E}\,\psi_1\,\mathcal{W}_{\leq t}\psi_2$

Для вычисления $find(\mathbf{A}\,\psi_1\,\mathcal{W}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$ (алгоритм 29), так же как и в алгоритме 27, используется принцип поиска двух неподвижных точек.

Вешняя неподвижная точка, как и в случае с алгоритмом 27, ищется до тех пор, пока не прекратит уменьшаться план $\{pln^1_{ag}\}$, уточняемый на каждой итерации с учетом результатов поиска внутренней неподвижной точки.

Алгоритм 29 (Вычисление $find(\mathbf{A}\,\psi_1\,\mathcal{W}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$**)**
$(ES^1_0, \{pln^1_{ag}\}) := find(\psi_1, C, env, \{sup_{ag}\});$
do
  $(ES^2_0, \{pln^2_{ag}\}) := find(\psi_2, C, env, \{pln^1_{ag} \cap (P^C \times C(\psi_2) \times C(\psi_2) \times A_{ag})\});$
  $ES_0 := (ES^1_0 \setminus ES^2_0) \cap (P^C \times C(\neg\psi_2)) \cup ES^1_0 \cap ES^2_0 \cap (P^C \times C(\psi_2));$
  $\{pln_{ag}\} := \{pln^1_{ag} \cap (P^C \times C(\neg\psi_2) \times I_{pln} \times A_{ag}) \cup pln^2_{ag}\};$
  $EF := \emptyset;$
  while $EF \neq ES_0$ ***do***
    $EF := ES_0;$
    $\{pln_{ag}\} := \{pln_{ag} \cap (Pre_{\mathbf{A}}(ES_0, env, \{sup_{ag}\})|_{ag} \cup P^C \times C(\psi_2) \times C(\psi_2) \times A_{ag})\};$
    $ES_0 = ES_0 \cap D(\{pln_{ag}\});$
  od;
  $(ES^1_0, \{pln^1_{ag}\}) := find(\psi_1, C, env, \{pln_{ag}\});$
while $\{pln_{ag}\} \neq \{pln^1_{ag}\};$
return $(ES_0, \{pln_{ag}\})$

Поиск же внутренней точки алгоритма 29 достаточно сильно отличается от алгоритма 27. В качестве начального множества состояний ES_0 берется множество ES^1_0, помеченного маркером $\neg\psi_2$ в той части, которая не пересекается с ES^2_0, и маркером ψ_2 в той части, которая пересекается с ES^2_0. В качестве начального плана $\{pln_{ag}\}$ берется объединение правил $\{pln^1_{ag}\}$, помеченных маркером $\neg\psi_2$ в левых частях, и правил $\{pln^2_{ag}\}$, которые уже помечены маркером ψ_2 по условиям их построения.

На каждой итерации вычисления внутренней неподвижной точки из множества правил $\{pln_{ag}\}$ удаляются те, которые помечены маркером $\neg\psi_2$ в правой части и за один ход могут привести систему вовне текущего множества ES_0. После уточнения множества правил уточняется и множество состояний ES_0 с помощью исключения тех состояний, которые не вошли в левую часть хотя бы одного правила $\{pln_{ag}\}$. Вычисление останавливается в том случае, если мно-

жество ES_0 перестанет уменьшаться.

После вычисления внутренней точки, уточняется план для подформулы ψ_1 с учетом новых ограничений и, если он прекратил уменьшаться, поиск останавливается. Заметим, что, так как на каждом этапе поиска внутренней неподвижной точки устраняются только правила, содержащие $\neg\psi_2$ в правой части, ни одно из правил $\{pln^2_{ag}\}$ исключено быть не может.

Алгоритм 30 (Вычисление $find(\mathbf{E}\,\psi_1\,\mathcal{W}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$**)**
$(ES^1_0, \{pln^1_{ag}\}) := find(\psi_1, C, env, \{sup_{ag}\})$;
$(ES^2_0, \{pln^2_{ag}\}) := find(\psi_2, C, env, \{pln^1_{ag} \cap (P^C \times C(\psi_2) \times C(\psi_2) \times A_{ag})\})$;
$ES_0 := (ES^1_0 \setminus ES^2_0) \cap (P^C \times C(\neg\psi_2)) \cup ES^1_0 \cap ES^2_0 \cap (P^C \times C(\psi_2))$;
$\{pln_{ag}\} := \{pln^1_{ag} \cap (P^C \times C(\neg\psi_2) \times I_{pln} \times A_{ag}) \cup pln^2_{ag}\}$;
$EF := \emptyset$;
while $EF \neq ES_0$ ***do***;
  $EF := ES_0$;
  $\{pln_{ag}\} := \{pln_{ag} \cap (Pre_{\mathbf{E}}(ES_0, env, \{sup_{ag}\})|_{ag} \cup P^C \times C(\psi_2) \times C(\psi_2) \times A_{ag})\}$***;***
  $ES_0 = ES_0 \cap D(\{pln_{ag}\})$***;***
od;
return $(ES_0, \{pln^1_{ag} \cap (P^C \times C(\neg\psi_2) \times I_{pln} \times A_{ag}) \cup pln^2_{ag}\})$

Для вычисления $find(\mathbf{E}\,\psi_1\,\mathcal{W}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$ (алгоритм 30) достаточно один раз вычислить внутреннюю неподвижную точку, исключая только те правила, которые гарантировано ведут вне текущего множества ES_0. После того, как *возможный* путь будет найден, результатом вычисления будет полученное множество состояний ES_0 и объединение планов для подформул с соответствующей разметкой.

Глава 8

Накопление и анализ опыта

В данной главе предлагаются методы накопления и анализа опыта для мультиагентной системы, основу которых составляет символическое представление данных, что позволяет в относительно компактном виде хранить опыт системы, расширять накопленный опыт, а также объединять опыт нескольких агентов. Кроме того, предложены методы анализа накопленного опыта и построения обобщений, использующие анализ структуры графа.

8.1. Символическое представление опыта и операции с ним

Для интеллектуального агента основным источником опыта является его взаимодействие с внешней средой, а информация о том, какая реакция внешней среды возможна при выполнении им тех или иных действий, позволяет ему прогнозировать последствия своих действий и строить планы. Следовательно, основу накопленного опыта агента составляет информация о поведении внешней среды. Кроме того, если агент является частью мультиагентной системы, важной для него информацией может быть то, какие действия могут выбирать другие агенты в зависимости от сложившихся условий.

Опыт агента ag можно представить в виде функции $res_{ag} : P_{F,ag} \times P_{ag} \to \mathbb{N}$, где $P_{F,ag}$ есть множество полных восприятий (определение 6), а P_{ag} есть множество восприятий агента (см. подраздел 2.3.1). Эта функция для каждой пары ситуация–последствия $(p_F, p) \in P_{F,ag} \times P_{ag}$ возвращает число $k \in \mathbb{N}$, отражающее то, сколько раз последствием ситуации p_F являлось состояние внешней среды, воспринимаемое как p.

Так как множества $P_{F,ag}$ и P_{ag} являются конечным, для них можно построить двоичные кодировки $P_{F,ag,\mathbb{B}} : P_{F,ag} \to \mathbb{B}^{n_{P_{F,ag}}}$ и $P_{ag,\mathbb{B}} : P_{ag} \to \mathbb{B}^{n_{P_{ag}}}$, где $n_{P_{F,ag}}$ и $n_{P_{ag}}$ есть минимальные числа, для которого выполнено $|P_{F,ag}| \leq 2^{n_{P_{F,ag}}}$ и $|P_{ag}| \leq 2^{n_{P_{ag}}}$.

Для того, чтобы построить символическое представление этой функции, ограничим максимальное число $k < 2^n$. В этом случае функцию res_{ag} можно представить как двоичную функцию $res_{ag,\mathbb{B}} : \mathbb{B}^{n_{P_{F,ag}}+n_{P_{ag}}} \to \mathbb{B}^n$, заданную разделяемой $OBDD$ из n компонент от $n_{P_{F,ag}} + n_{P_{ag}}$ переменных.

8.1.1. Добавление факта

Каждый раз, когда агент попадает из некоторой ситуации $p_{F,ag}$ в состояние, воспринимаемое как p_{ag}, необходимо вычислить обновленную функцию:

$$res_{ag} \;:=\; res_{ag} + add(p_{F,ag}, p_{ag}), \tag{8.1}$$

где $add(p_{F,ag}, p_{ag})$ есть функция, возвращающая 0 во всех точках, кроме $(p_{F,ag}, p_{ag})$, а в этой точке возвращающая 1. После такого преобразования счетчик результатов будет увеличен на единицу для ситуации $p_{F,ag}$ и результата p_{ag}.

Однако, после увеличения на 1 значение $res_{ag}(p_{F,ag}, p_{ag})$ может превысить максимальное допустимое с точки зрения кодировки число $2^n - 1$. Если такое произойдет, то необходимо произвести следующие дополнительные вычисления:

$$res_{ag} \;:=\; \frac{res_{ag}}{div(p_{F,ag})}, \tag{8.2}$$

где $div(p_{F,ag})$ есть функция, возвращающая 1 во всех точках, кроме точек вида $\{(p'_{F,ag}, p'_{ag}) \in P_{F,ag} \times P_a g \mid p'_{F,ag} = p_{F,ag}\}$, а в этих точках возвращающая 2. После этого преобразования функция res_{ag} будет удовлетворять условиям кодировки. Заметим, что для сохранения пропорций необходимо выполнять деление на два

не только в точке $(p_{F,ag}, p_{ag})$, но и во всех точках с соответствующим значением $p_{F,ag}$.

Для выполнения операции прибавления единицы с $OBDD$ необходимо проделать следующие действия. В цикле по компонентам в кодировке функции $res_{ag,\mathbb{B}}$ от младших битов к старшим выполнить следующие действия:

1. инвертировать значение компоненты в точке $(p_{F,ag}, p_{ag})$;

2. если значение компоненты в точке $(p_{F,ag}, p_{ag})$ равно 1, прервать цикл.

Если цикл по компонентам $res_{ag,\mathbb{B}}$ дошел до несуществующей компоненты, представляющей бит n, следовательно, необходимо выполнять деление на 2.

Для выполнения деления необходимо в цикле по компонентам от старших битов к младшим выполнить действия, описанные в алгоритме 31.

Алгоритм 31 (Деление на 2)
$shift := \{(p_{F,ag}, p_{ag})\}$; $target = \{(p'_{F,ag}, p'_{ag}) \in P_{F,ag} \times P_ag \mid p'_{F,ag} = p_{F,ag}\}$;
for $i = n - 1$ ***to*** 0 ***do***
  $shift' := res_i \cap target$;
  $res_i := (res_i \setminus target) \cup shift$;
  $shift := shift'$;
od;

Так как сложность операции получения и изменения значения в точке не превосходит количества переменных функции, учитывая, что эти операции нужно выполнить для каждой компоненты функции, сложность операции добавления факта в предложенном представлении не превосходит $O(n \cdot \log(|P_{F,ag}| \cdot |P_{ag}|))$, но только в том случае, если нет необходимости производить деление на 2. Если же происходит переполнение и необходимо выполнять деление, то сложность алгоритма возрастает до $O(n \cdot (|P_{F,ag}| \cdot |P_{ag}|)^2)$, так как необходимо выполнить несколько операций над множествами.

8.1.2. Объединение накопленного опыта

В случае, если несколько агентов действуют совместно в рамках единой системы, встает задача интеграция накопленного ими опыта. Для этого сначала необходимо перевести их опыт в единый формат $res_{ag} : S \times ACS \times S \to \mathbb{N}$, используя композицию функций. После этого все полученные функции необходимо просуммировать, поделив результаты, выходящие за рамки кодировки на 2.

Алгоритм 32 (Суммирование функций)
$extra := \emptyset$;
for $i = 0$ ***to*** $n - 1$ ***do***
  $add := res'_i$; $j := i - 1$;
  while $add \neq \emptyset$ ***and*** $++j < n$ ***do***
    $add' := (res_j \cap add)$;
    $res_j := (res_j \cup add) \setminus add'$;
    $add := add'$;
  od;
  $extra := extra \cup add$;
od;

Суммирование двух функций res и res' можно провести двойным циклом по компонентам от младших битов к старшим, как показано в алгоритме 32. На внешнем цикле происходит итерация по компонентам res' и суммирования их с компонентами res. Суммирование начинается с компоненты с соответствующим индексом и продолжается то того момента, пока не опустеет множество add, представляющее множество перенесенных на старший бит значений, или пока не будет достигнут предел имеющихся в наличии компонентов. На каждой итерации внутреннего цикла вычисляется сложение для текущего бита j. Значение 1 для этого бита функция должна принимать на тех элементах, которые входят только в одно из рассматриваемых множеств res_j или add, а для тех элементов, которые входят в оба множества, 1 должна быть перенесена на старший бит. Заметим, что так как максимальное значение обеих входных функций не

превосходит $2^n - 1$, следовательно, максимальное значение их суммы не будет превышать $2^{n+1} - 2$, а значит для каждого из элементов возможно только одно перенесение за старший бит.

Кроме того, если внутренний цикл закончится, не опустошив множество add, значит для всех ситуаций $p_{F,ag}$, оставшихся в этом множестве, необходимо поделить значение функции на 2, как показано в алгоритме 31.

Таким образом, последовательно суммируя накопленный агентами опыт, можно получить искомую функцию и $res : S \times ACS \times S \to \mathbb{N}$.

Согласно [41], сложность операций со множествами не превосходит произведения максимального размера этих множеств, а сложность вычисление композиции функций не превосходит произведения квадрата размера представления внешней функции с размером представления внутренней функции. Следовательно, сложность алгоритма построения общего опыта системы не превосходит $O(|AG| \cdot n^2 \cdot |S|^5 \times |ACS|^2)$.

8.1.3. Перевод в отношение $bel \subseteq S \times ACS \times S$

Для того, чтобы полученный агентами опыт мог служить входными данными для алгоритма планирования или алгоритмов верификации, необходимо преобразовать накопленный системой опыт в представление вида $bel \subseteq S \times ACS \times S$. Наиболее простым способом перевода является $\bigcup\limits_{i<n} res_i \cup (((S \times ACS) \setminus \bigcup\limits_{i<n} res_i|_{S \times ACS}) \times S)$. В этом случае для известных ситуаций возможными последствиями считаются те, для которых ненулевым является результат функции res, а для неизвестных ситуаций (тех пар $(s, a) \in S \times ACS$, для которых не существует такого $s' \in S$, что $res(s, a, s') \neq 0$) возможным считается любой результат.

Однако, подобное преобразование приведет к тому, что итоговое множество представлений будет допускать для каждой из ситуаций все возможные последствия, включая самые маловероятные. Символическое представление результи-

рующего множества может оказаться относительно большим, кроме того, маловероятные последствия могут сильно усложнить работу алгоритмов планирования.

Альтернативой полному преобразованию является преобразование с использованием *порога отсечения* $0 \leq \theta \leq 1$. Определим функцию $\max(res) : S \times ACS \to \mathbb{N}$ следующим образом:

$$\max(res)(s,a) \triangleq \max_{s' \in S} res(s,a,s'). \tag{8.3}$$

Для заданной пары $(s,a) \in S \times ACS$ функция $\max(res)$ возвращает максимально возможное значение функции res. После того, как функция $\max(res)$ вычислена, отношение bel_θ можно построить следующим образом:

$$bel_\theta = \{(s,a,s') \in S \times ACS \times S \mid res(s,a,s') \geq \theta \cdot \max(res)(s,a)\}. \tag{8.4}$$

Таким образом, для ситуации $(s,a) \in S \times ACS$ множество bel_θ включает только те результаты, которые были получены в не менее чем в $\theta \cdot \max(res)(s,a)$ случаях. При $\theta = 1$ допустимым считаются только максимально вероятные результаты, а при $\theta = 0$ — все результаты, включая невозможные. Подобное преобразование позволяет отсекать маловероятные последствия, уменьшая тем самым размер символического представления и упрощая работу алгоритмов планирования.

С точки зрения операций с $OBDD$ функцию $\max(res)$ можно вычислить, как показано в алгоритме 33. Для каждой компоненты в представлении res в цикле от старших битов к младшим вычисляется множество тех ситуаций, для которых этот бит в представлении функции $\max(res)$ равняется 1. При этом на каждом шаге отдельно рассматриваются множества ситуаций, для которых хотя бы один из уже вычисленных старших битов не равен 1, и для которых все старшие биты $\max(res)$ равны 0.

После того, как вычислено представление функции $\max(res)$, необходимо вычислить и представление функции $\theta \cdot \max(res)$. Если $\theta = 1$, то это вычисление

Алгоритм 33 (Поиск максимума)
$found := \emptyset$;
for $i = n - 1$ ***to*** 0 ***do***
  $\max(res)_i := (res_i \cap (found \times S))|_{S \times ACS} \cup (res_i \setminus (found \times S))|_{S \times ACS}$;
  $found := found \cup \max(res)_i$;
od;

тривиально. Если же $\theta < 1$, возьмем максимальное число $\theta_{\mathbb{B}} < 2^n$, для которого выполнено $\frac{\theta_{\mathbb{B}}}{2^n} \leq \theta$, после чего умножим представление функции $\max(res)$ на это число, как показано в алгоритме 34, и возьмем первые n компонентов полученного результата. Для умножения функции на число в цикле для каждого бита числа, равного единице, текущий результат суммируется с представлением функции $\max(res)$, сдвинутым на i битов вправо, где i есть номер текущего бита.

Алгоритм 34 (Вычисление $\theta \cdot \max(res)$)
$result := 0$;
for $i = n - 1$ ***to*** 0 ***do***
  if $\theta_{\mathbb{B},i} = 1$ ***then*** $result := result + (\max(res)_{bool} \cdot 2^i)$;
od;

Заключительным этапом в построении представления отношения bel_θ является преобразование представлений функций res и $\theta \cdot \max(res)$, показанное в алгоритме 35. В цикле по компонентам res и $\theta \cdot \max(res)$ от старших битов к младшим к множеству bel_θ на каждом этапе добавляются те элементы, старшие биты которых в представлениях res и $\theta \cdot \max(res)$ совпадают и при этом текущий бит в res равен 1, а в $\theta \cdot \max(res)$ — 0. После чего из множества эквивалентных элементов исключаются те элементы, текущие биты которых в представлениях res и $\theta \cdot \max(res)$ различаются. По завершению цикла к множеству bel_θ добавляются те элементы, все биты которых совпадают.

Алгоритм 35 (Вычисление bel_θ)
$bel_\theta := \emptyset$; $eqv := S \times ACS \times S$;
for $i = n - 1$ ***to*** 0 ***do***
  $bel_\theta := bel_\theta \cup ((res_i \cap eqv) \setminus (\theta \cdot \max(res))_i)$;
  $eqv := eqv \setminus (res_i \text{ XOR } (\theta \cdot \max(res))_i)$;
od;
$bel_\theta := bel_\theta \cup eqv$;

8.2. Методы анализа и построения обобщений

Для проведения анализа накопленного опыта и построения обобщений удобнее работать с функцией res, представленной не набором двоичных разрешающих диаграмм, а одной алгебраической диаграммой. Для перевода из $OBDD$ в ADD можно воспользоваться следующей формулой:

$$res_{add} = \sum_{0 \le i < n} res_i \cdot 2^i. \tag{8.5}$$

При этом каждую компоненту res_i в двоичном представлении можно рассматривать как алгебраическую диаграмму со значениями 0 и 1.

Обратный перевод можно сделать аналогичным образом, как показано в алгоритме 36. В цикле от старших битов к младшим текущей компоненте в двоичном представлении присваивается множество таких троек (s, a, s'), в которых значение res_{add} превосходит 2^i, где i есть номер текущего бита, после чего значение res_{add} уменьшается на 2^i во всех точках.

Алгоритм 36 (Преобразование ADD в $OBDD$)
for $i = n - 1$ ***to*** 0 ***do***
  $res_i := \{(s, a, s') \in S \times ACS \times S \mid res_{add}(s, a, s') \ge 2^i\}$;
  $res_{add} := res_{add} - 2^i$;
od;

8.2.1. Проекция редукции

Одним из самых простых способов оптимизации накопленного опыта и его анализа является *проекция редукции* представления отношения bel, построенного на основе накопленного опыта, на представление res_{add}. Редукция, проведенная в представлении отношение bel, позволяет естественным образом выделить ситуации и результаты со схожим поведением, что можно использовать для оптимизации представления res_{add}. Проекция редукции заключается в устранении тех вершин в представлении res_{add}, которые отсутствуют в представлении bel.

С точки зрения реализации, проекция редукции производится с помощью рекурсии по структуре графов. Функцию, вычисляющую проекцию редукции, обозначим $Project(v, v')$, где v есть некоторая вершина представления res_{add}, а v' есть некоторая вершина представления bel. Функция $Project$ вычисляется по следующим правилам:

1. если v является терминальной вершиной, то $Project(v, v') = v$;
2. если $index(v') > index(v)$, то $Project(v, v') = Project(low(v) + high(v), v')$;
3. если $index(v') = index(v)$, то $Project(v, v') = v''$, где $index(v'') = index(v)$, $low(v'') = Project(low(v), low(v'))$, а $high(v'') = Project(high(v), high(v'))$;
4. Если $index(v') < index(v)$, то $Project(v, v') = Project(v, low(v') \cup high(v'))$.

Однако, после подобного преобразования значения функции res_{add} в некоторых точках могут превысить максимум, доступный для двоичной кодировки. Для представления в виде ADD это превышение не является препятствием, так как алгебраические диаграммы могут использоваться для представления бóльших чисел, но преобразовать эту алгебраическую диаграмму в набор двоичных, используя алгоритм 36, невозможно. Для решения этой проблемы необходимо пропорционально уменьшить значение функции res_{add}.

Самым простым способом уменьшения является деление значений всех терминальных вершин на соответствующий понижающий коэффициент, однако такое преобразование может привести к существенной потере информации. Альтернативой подобному подходу является уменьшение значения функции только в тех точках, которые выходят за пределы кодировки, как показано в алгоритме 37.

Алгоритм 37 (Ограниченное деление)
$reduce := \{(s,a,s') \in S \times ACS \times S \mid res_{add}(s,a,s') \geq 2^n\}|_{S \times ACS}$;
while $reduce \neq \emptyset$ ***do***
  $div(s,a,s') := (s,a) \in reduce \; ? \; 2 : 1$;
  $res_{add} := res_{add} \div div$;
  $reduce := \{(s,a,s') \in S \times ACS \times S \mid res_{add}(s,a,s') \geq 2^n\}|_{S \times ACS}$;
od;

8.2.2. Расширенная редукция

Другим способом построения обобщений является расширенная редукция схожих ситуаций. В основе расширенной редукции лежит *критерий схожести* и *коэффициент пропорциональности* ситуаций, позволяющие для двух функций res и res' определить, насколько эти функции "похожи" друг на друга. В качестве критерия и коэффициента можно использовать самые различные виды метрик и способы вычисления пропорций, эффективность которых будет зависеть от конкретных задач. Исследование возможных критериев и коэффициентов, а также анализ их эффективности выходит за рамки данной работы, поэтому, для примера, воспользуемся одними из наиболее общих способов их вычисления:

$$dist(res, res') \triangleq \sum_{s,s' \in S, a \in ACS} \frac{|res(s,a,s') - res'(s,a,s')|}{|S|^2 \cdot |ACS|}, \tag{8.6}$$

$$\tau \triangleq \min_{s,s' \in S, a \in ACS} \frac{res(s,a,s')}{res'(s,a,s')}. \tag{8.7}$$

При этом, если $dist(res, \tau \cdot res') < \theta$, где θ есть некоторое предопределенное пороговое значение, то функции считаются схожими.

Вычислить критерий $dist$ для двух функций res и res', представленных в виде алгебраических разрешающих диаграмм, можно следующим образом. Сначала вычислить представление функции $|res - \tau \cdot res'|$, используя стандартные операции с разрешающими диаграммами. После этого, вычислить $\sum_{s,s' \in S, a \in ACS} res(s, a, s') - \tau \cdot res'(s, a, s')$, используя рекурсивную функцию $Sum(v, 2^{2 \cdot n_{P_{F,ag}} + n_{P_{ag}}})$, заданную следующими правилами:

1. если v является терминальное вершиной, то $Sum(v, i) = value(v) \cdot i$;
2. иначе, $Sum(v, i) = Sum(low(v), \frac{i}{2}) Sum(high(v), \frac{i}{2})$.

После чего остается только поделить полученное число на $2^{2 \cdot n_{P_{F,ag}} + n_{P_{ag}}}$.

Используя некоторый критерий схожести функций, расширенную редукцию можно реализовать с помощью рекурсивной функции $Reduce(v)$, удовлетворяющей следующим правилам:

1. если v является терминальной вершиной, то $Reduce(v) = v$;
2. если $Reduce(low(v))$ и $Reduce(high(v))$ являются схожими относительно критерия схожести, то $Reduce(v) = \frac{Reduce(low(v)) + Reduce(high(v))}{2}$;
3. Иначе $Reduce(v) = v'$, где $index(v') = index(v)$, $low(v) = Reduce(low(v))$, а $high(v) = Reduce(high(v))$.

8.2.3. Присоединение малознакомых ситуаций

Для того, чтобы лучше прогнозировать реакцию внешней среды в малознакомых ситуациях, можно объединить эти ситуации со схожими лучше изученными ситуациями. При присоединении ситуаций из представления res удаляются те вершины, которые соответствуют малозначимым битам в представлении пары $(s, a) \times S \times ACS$ и дочерние вершины которых представляют схожие

функции. Под малозначимыми битами понимаются те биты, изменение которых слабо влияет на представляемое значение. Например, в представлении целых чисел малозначимыми битами являются младшие биты числа.

Реализуется присоединение малознакомых ситуаций с помощью рекурсивной функции $Concat(v)$, заданной следующими правилами:

1. если v является терминальной вершиной, то $Reduce(v) = v$;

2. если $index(v)$ соответствует одному из малозначимых битов, значение $Sum(Reduce(low(v)), 2^{index(v)})$ значительно больше или значительно меньше $Sum(Reduce(high(v)), 2^{index(v)})$, а также вершины $Reduce(low(v))$ и $Reduce(high(v))$ представляют функции, схожие относительно критерия схожести, то $Reduce(v) = \frac{Reduce(low(v))+Reduce(high(v))}{2}$;

3. иначе $Reduce(v) = v'$, где $index(v') = index(v)$, $low(v) = Reduce(low(v))$, а $high(v) = Reduce(high(v))$.

Внешне алгоритм присоединения малознакомых ситуаций похож на алгоритм расширенной редукции, однако, между ними есть несколько существенных отличий:

1. при присоединении малознакомых ситуаций устраняются только вершины, соответствующие малозначимым битам, следовательно, присоединение малознакомых ситуаций ведет к меньшему искажению;

2. объединяются вершины только в том случае, если информации о ситуациях на одной из ветвей значительно больше, чем на другой.

Таким образом, присоединение малознакомых ситуаций позволяет уменьшить дисбаланс значений функции res в близких точках. Следовательно, присоединение малознакомых ситуаций имеет смысл выполнять *до* проведения расширенной редукции, чтобы уменьшить возможные искажения при объединении

схожих функций с большим или, наоборот, с малым коэффициентом пропорциональности.

Заключение

В данной работе предложен комплекс формально-логических методов спецификации самонастраивающихся мультиагентных систем с временными ограничениями. Предложенный в работе комплекс методов может быть использован для формальной спецификации, верификации и автоматического тестирования мультиагентных систем, а также для проектирования и реализации агентов, способных к планированию действий и к накоплению и анализу опыта. Кроме того, предложенные методы могут быть использованы при обучении методам проектирования мультиагентных систем.

Наиболее перспективным представляется применение предложенных методов для разработки систем управления устройствами и комплексами, соответствующими требованиям разработанной модели, в условиях наличия у системы времени на предварительное планирование действий. К этому классу можно отнести, например, следующие системы:

- системы автоматического перемещения грузов;
- автономные исследовательские зонды;
- системы управления автоматизированным производством;
- системы автоматического поиска предметов и обхода территории.

Основные результаты

В рамках диссертации получены следующие результаты.

1. Разработана математическая модель интеллектуального агента, описывающая такие аспекты деятельности агента, как взаимодействие с внешней

средой, неполнота информации, способность к анализу и накоплению опыта, ограничения на время реакции, управляемое целями поведение и планирование действий.

2. Разработана математическая модель мультиагентной системы, включающая такие аспекты деятельности сообщества интеллектуальных агентов, как коммуникация, координация и кооперация агентов.

3. Разработан логический формализм для спецификации свойств интеллектуального агента, позволяющий описывать свойства математической модели интеллектуального агента.

4. Разработан логический формализм для спецификации мультиагентной системы с временными ограничениями $MASL$, включающий логику спецификации интеллектуального агента и расширяющий её средствами описания свойств определенных групп агентов (коалиций).

5. Разработаны и реализованы алгоритмы верификации ("проверки модели", model checking) систем по спецификациям $MASL$, имеющие полиномиальную сложность и использующие технику символической верификации. Корректность, завершаемость и полнота алгоритмов доказаны, кроме того, доказано, что их алгоритмическая сложность не превосходит полинома от размера модели и длины формулы.

6. Разработаны и реализованы алгоритмы построения мультиагентных планов, удовлетворяющих спецификации $MASL$ и использующие технику символического планирования. Корректность и завершаемость алгоритмов доказаны, кроме того, доказано, что их алгоритмическая сложность не превосходит полинома от размера модели и экспоненты от длины формулы.

7. Предложены методы накопления и анализа опыта в символической форме с помощью разрешающих диаграмм, позволяющих в относительно компактной форме хранить весь опыт системы и быстро преобразовывать этот опыт в структуры, удобные для символических алгоритмов построения планов и верификации.

Приложение А

Анализ алгоритмов

В данном приложении приведены доказательства корректности и полноты, а также анализ сложности предложенных алгоритмов верификации и планирования. Для доказательства используется индукция по структуре формулы и рассматривается алгоритм для каждого типа подформул в отдельности.

А.1. Анализ корректности, полноты и сложности алгоритмов верификации для логики спецификации интеллектуального агента

А.1.1. Вычисление предикатов и логических связок

Корректность и полнота алгоритма вычисления множества $[P_i(\tau_1^s, \ldots, \tau_i^s)]^s$ доказывается в утверждении 38, а анализ алгоритмической сложности этого процесса приведен в утверждении 39. Корректность и полнота вычисления логических связок очевидна, а их алгоритмическая сложность не превосходит сложности вычисления пересечения для множеств вида $S \times I_{pln}$, то есть не превосходит $O((|S| \cdot |I_{pln}|)^2)$.

Утверждение 38 *Для любого терма τ^s, для любых $s \in S$ и $d \in Domain$, если $\lceil \tau^s \rceil(s) = d$, то $\lceil \tau^s \rceil_{\mathbb{B}}(S_{\mathbb{B}}(s)) = Domain_{\mathbb{B}}(d)$ и наоборот.*

Доказательство. Для доказательства воспользуемся индукцией по количеству рекурсивных вызовов функциональных символов. Если терм τ^s является предметной переменной v_s, то $\lceil \tau^s \rceil = \pi(v_s)$, а $\lceil \tau^s \rceil_{\mathbb{B}} = \pi(v_s)_{\mathbb{B}}$. Предположим, что существуют такие $s \in S$ и $d \in Domain$, что $\lceil \tau^s \rceil(s) = d$ и $\lceil \tau^s \rceil_{\mathbb{B}}(S_{\mathbb{B}}(s)) \neq Domain_{\mathbb{B}}(d)$. При этом, если $\pi(v_s)(s) = d$, то по определению $\pi(v_s)_{\mathbb{B}}$ выполнено

$\lceil\tau^s\rceil_{\mathbb{B}}(S_{\mathbb{B}}(s)) = Domain_{\mathbb{B}}(d)$, что противоречит исходному утверждению. База доказана.

Если терм τ^s является функциональным символом $f(\tau_1^s, \ldots, \tau_i^s)$ и для всех его параметров функция $\lceil\tau_i^s\rceil_{\mathbb{B}}$ вычислена, то $\lceil f(\tau_1^s, \ldots, \tau_i^s)\rceil(s) = \Phi_i(f)(\lceil\tau_1^s\rceil(s), \ldots, \lceil\tau_i^s\rceil(s))$. Предположим, что существуют такие $s \in S$ и $d \in Domain$, что $\lceil f(\tau_1^s, \ldots, \tau_i^s)\rceil(s) = d$ и $\lceil f(\tau_1^s, \ldots, \tau_i^s)\rceil_{\mathbb{B}}(S_{\mathbb{B}}(s)) \neq Domain_{\mathbb{B}}(d)$. При этом, если $\Phi_i(f)(\lceil\tau_1^s\rceil(s), \ldots, \lceil\tau_i^s\rceil(s)) = d$, то $Domain_{\mathbb{B}}(d) = \Phi_i(f)_{\mathbb{B}}(\lceil\tau_1^s\rceil_{\mathbb{B}}(S_{\mathbb{B}}(s)), \ldots, \lceil\tau_i^s\rceil_{\mathbb{B}}(S_{\mathbb{B}}(s))) = \lceil f(\tau_1^s, \ldots, \tau_i^s)\rceil_{\mathbb{B}}(S_{\mathbb{B}}(s))$, что противоречит исходному предположению.

Доказательство в обратную сторону можно провести аналогичным образом. ■

Утверждение 39 *Алгоритмическая сложность вычисления* $[P_i(\tau_1^s, \ldots, \tau_i^s)]$ *не превосходит* $O(|P_i(\tau_1^s, \ldots, \tau_i^s)| \cdot i_{\max} \cdot \log(|S|)^2 \cdot |S|^{3+2\cdot i_{max}})$, *где* $|P_i(\tau_1^s, \ldots, \tau_i^s)|$ *есть количество вхождений предметных переменных, предикатных и функциональных символов в формулу* $P_i(\tau_1^s, \ldots, \tau_i^s)$, $i_{\max}$ *есть максимальная размерность предикатных и функциональных символов, а* $|S|$ *есть мощность множества* S.

Доказательство. Вычисление множества $P_i(\tau_1^s, \ldots, \tau_i^s)^s$ состоит из последовательного вычисления операции композиции. Количество диаграмм, для которых надо вычислить композицию, совпадает с числом вхождений функциональных символов умноженным на n_D (в соответствии с количеством компонентов) плюс один (для самого предиката), что не превышает $|P_i(\tau_1^s, \ldots, \tau_i^s)| \cdot n_D$. При этом, для каждой из этих диаграмм композицию нужно вычислить для каждого из параметров, от которых она зависит, что в сумме дает $|P_i(\tau_1^s, \ldots, \tau_i^s)| \cdot i_{\max} \cdot n_D^2$. Согласно [41], сложность вычисления композиции не превышает $O(|G_1|^2 \cdot |G_2|)$, где $|G_1|$ есть максимальное количество вершин в исходной функции, а $|G_2|$ есть максимальное количество вершин в функции-параметре композиции. В нашем случае $|G_1| \leq 2^{n_S + n_D \cdot i_{\max}}$, а $|G_2| = 2^{n_S}$, что в итоге дает $O(2^{2 \cdot n_D \cdot i_{\max} + n_S})$.

Учитывая, что $n_D \approx \log(|Domain|)$ и $n_S \approx \log(|S|)$, а также то, что не умаляя общности можно считать, что $|Domain| \leq |S|$, получаем общую оценку $O(|P_i(\tau_1^s, \ldots, \tau_i^s)| \cdot i_{\max} \cdot \log(|S|)^2 \cdot |S|^{3+2 \cdot i_{max}})$. ■

A.1.2. Вычисление прообраза

Алгоритм вычисления прообраза $Pre_{\mathbf{E}}(x, env_c)$ при вычисленном множестве $env_c \| \sigma'$ совпадает со стандартным алгоритмом для CTL, следовательно, он является корректным и полным, а его сложность не превосходит $O(|S \times I_{pln} \times S \times I_{pln}| \cdot |S \times I_{pln}|) \leq O((|S| \cdot |I_{pln}|)^3)$. Для построения множества σ' используется операция композиции функций, корректность и полнота которой очевидна. Корректность и полнота алгоритма вычисления множества $env_c \| \sigma'$ доказывается в утверждении 40. Алгоритмическая сложность этих процессов проанализирована в утверждении 41.

Утверждение 40 $((env_c \times I_{pln} \times I_{pln}) \cap (\sigma' \times S))|_{S \times I_{pln} \times S \times I_{pln}} = env_c \| \sigma' = \{(s, i_{pln}, s', i'_{pln}) \in S \times I_{pln} \times S \times I_{pln} \mid \exists a \in A : (s, a, s') \in env_c \; and \; (s, i_{pln}, a, i'_{pln}) \in \sigma'_{pln}\}$.

Доказательство. Предположим, что существуют $(s, i_{pln}, s', i'_{pln}) \in ((env_c \times I_{pln} \times I_{pln}) \cap (\sigma' \times S))|_{S \times I_{pln} \times S \times I_{pln}}$, такие, что $(s, i_{pln}, s', i'_{pln}) \notin env_c \| \sigma'$. Следовательно, для любого $a \in A$ либо $(s, a, s') \notin env_c$, либо $(s, i_{pln}, a, i'_{pln}) \notin \sigma'_{pln}$ (либо и то, и то). Если $(s, a, s') \notin env_c$, то для любых $i_{pln}, i'_{pln} \in I_{pln}$ $(s, i_{pln}, a, s', i'_{pln}) \notin (env_c \times I_{pln} \times I_{pln})$, следовательно, $(s, i_{pln}, s', i'_{pln}) \notin ((env_c \times I_{pln} \times I_{pln}) \cap (\sigma' \times S))|_{S \times I_{pln} \times S \times I_{pln}}$. Аналогично, если $(s, i_{pln}, a, i'_{pln}) \notin \sigma'_{pln}$, то для любого $s' \in S$ $(s, i_{pln}, a, s', i'_{pln}) \notin (\sigma' \times S)$, следовательно, $(s, i_{pln}, s', i'_{pln}) \notin ((env_c \times I_{pln} \times I_{pln}) \cap (\sigma' \times S))|_{S \times I_{pln} \times S \times I_{pln}}$. Значит, исходное предположение неверно.

Предположим, что существуют $(s, i_{pln}, s', i'_{pln}) \in env_c \| \sigma'$, такие, что $(s, i_{pln}, s', i'_{pln}) \notin ((env_c \times I_{pln} \times I_{pln}) \cap (\sigma' \times S))|_{S \times I_{pln} \times S \times I_{pln}}$. Следовательно, существует $a \in A$, такое, что $(s, a, s') \in env_c$ и $(s, i_{pln}, a, i'_{pln}) \in \sigma'_{pln}$, значит $(s, i_{pln}, s', i'_{pln}) \in$

$(env_c \times I_{pln} \times I_{pln})|_{S\times I_{pln}\times S\times I_{pln}}$ и $(s, i_{pln}, s', i'_{pln}) \in (\sigma' \times S)|_{S\times I_{pln}\times S\times I_{pln}}$. Следовательно, $(s, i_{pln}, s', i'_{pln}) \in ((env_c \times I_{pln} \times I_{pln}) \cap (\sigma' \times S))|_{S\times I_{pln}\times S\times I_{pln}}$, что и требовалось доказать. ■

Утверждение 41 *Сложность вычисления множеств σ' и $env_c||\sigma'$ не превосходит $O(|S|^5 \cdot |I_{pln}|^4 \cdot |A|^2 \cdot \log(|S|))$.*

Доказательство. Вычисление функции σ' включает вычисление n_P композиций функции от $n_S + n_P + 2 \cdot n_{pln} + n_A$ переменных с функцией от n_S переменных. Следовательно, сложность вычисления σ' не превосходит $O(n_P \cdot 2^{2\cdot(n_S+n_P+2\cdot n_{pln}+n_A)+n_S})$. Значит, так как $|P| \leq |S|$ сложность алгоритма не превосходит $O(|S|^5 \cdot |I_{pln}|^4 \cdot \cdot|A|)^2 \cdot \log(|S|))$.

Наиболее сложной операцией в вычисление $env_c||\sigma'$ является построение пересечения множества $env_c \times I_{pln} \times I_{pln}$ с множеством $\sigma' \times S$, сложность которого не превосходит $O(|S \times A \times S| \cdot |S \times I_{pln} \times I_{pln} \times A|) \leq O(|S^3| \cdot |I_{pln}|^2 \cdot |A|^2)$. Следовательно, общая сложность не превосходит $O(|S|^5 \cdot |I_{pln}|^4 \cdot |A|^2 \cdot \log(|S|))$, что и требовалось доказать. ■

A.1.3. Вычисление оператора $[\alpha]\varepsilon$

Алгоритм 16 вычисления множества $[\alpha]$, с учетом корректности и полноты функции вычисления $[P_i(\tau_1^a, \ldots, \tau_i^a)]$ (утверждения 38), очевидно корректен и полон. Его алгоритмическая сложность, как показано в утверждении 42, не превосходит $O(|A|^{3+2\cdot i_{\max}} \cdot |\alpha| \cdot \log(|A|) \cdot i_{\max})$.

Корректность и полнота алгоритма вычисления ограниченного прообраза $Pre_{\mathbf{E}}(x, env_c, \alpha)$ доказывается в утверждении 43. А его алгоритмическая сложность очевидно не превосходит $O(|A|^2 \cdot |S|^3)$, так как он вычисляется с помощью пересечения множеств, размер представления которых не превышает $|A|$, $|S \times A \times S|$ и $|S|$.

Утверждение 42 *Сложность вычисления* $[\alpha]$ *не превосходит* $O(|A|^{3+2\cdot i_{\max}} \cdot |\alpha| \cdot \log(|A|)^2 \cdot i_{\max})$, *где* $|A|$ *есть мощность множества действий,* $i_{\max}$ *есть максимальная размерность предикатных и функциональных символов, а* $|\alpha|$ *есть количество логических связок, функциональных и предикатных символов в формуле* α.

Доказательство. Множество $[\alpha]$ вычисляется рекурсивно по структуре формулы, при этом число рекурсивных вызовов не превосходит $|\alpha|$. На каждом шаге рекурсии либо производится манипуляция с двумя множествами, размер которых не превосходит $|A|$ (для вычисления предикатных символов), либо $n_D^2 \cdot i_{\max}$ раз вычисляется композиция функции от не более чем $n_A + n_D \cdot i_{\max}$ переменных с функцией от n_A переменных (при обработке функциональных и предикатных символов). В первом случае сложность не превосходит $O(|A|^2)$, а во втором $O(|A|^{3+2\cdot i_{\max}} \cdot \log(|A|) \cdot i_{\max})$. Следовательно, общая сложность не превосходит $O(|A|^{3+2\cdot i_{\max}} \cdot |\alpha| \cdot \log(|A|)^2 \cdot i_{\max})$. ■

Утверждение 43 $((S \times [\alpha] \times S) \cap env_c \cap (S \times A \times x))\,|_S = \{s \in S \mid \exists a \in A, s' \in A : (s, a, s') \in env_c \; and \; a \in [\alpha] \; and \; s' \in x\}$.

Доказательство. Предположим, что существует $s \in ((S \times [\alpha] \times S) \cap env_c \cap (S \times A \times x))|_S$, такое, что $s \notin \{s \in S \mid \exists a \in A, s' \in A : (s, a, s') \in env_c \; and \; a \in [\alpha] \; and \; s' \in x\}$. Следовательно, для любых $a \in A$ и $s' \in S$ нарушено хотя бы одно из условий: $(s, a, s') \notin env_c$ или $a \notin [\alpha]$ (значит $(s, a, s') \notin S \times [\alpha] \times S$) или $s' \notin x$ (значит $(s, a, s') \notin S \times A \times x$). Следовательно, $s \notin ((S \times [\alpha] \times S) \cap env_c \cap (S \times A \times x))|_S$, что противоречит исходному предположению.

Предположим, что существует $s \notin ((S \times [\alpha] \times S) \cap env_c \cap (S \times A \times x))|_S$, такое, что $s \in \{s \in S \mid \exists a \in A, s' \in A : (s, a, s') \in env_c \; and \; a \in [\alpha] \; and \; s' \in x\}$. Следовательно, существуют такие $a \in A$ и $s \in S'$, что $(s, a, s') \in env_c$ и $a \in [\alpha]$ (значит $(s, a, s') \in S \times [\alpha] \times S$) и $s' \in x$ (значит $(s, a, s') \in S \times A \times x$). Следовательно, $s \in ((S \times [\alpha] \times S) \cap env_c \cap (S \times A \times x))|_S$, что противоречит

исходному предположению. ■

A.1.4. Вычисление темпоральных операторов

Для доказательства корректности и полноты алгоритмов вычисления темпоральных операторов рассмотрим только случаи с квантором пути $\mathbf{E}$, а случаи с $\mathbf{A}$ легко могут быть доказаны по аналогии.

Доказательство корректности и полноты вычисления $[\mathbf{E} \bigcirc \phi]$ приведено в утверждении 44, при этом его алгоритмическая сложность, очевидно, совпадает с алгоритмической сложностью вычисления $Pre_{\mathbf{E}}([\phi]_{env_c}, env_c)$, которая не превосходит $O((|S| \cdot |I_{pln}|)^3)$.

Доказательство корректности и полноты вычисления $[\mathbf{E}\, \phi_1\, \mathcal{U}_{\leq t} \phi_2]_{env_c}$ приведено в утверждении 45, а алгоритмическая сложность этого вычисления, как показано в утверждении 46, не превосходит $O((|S| \cdot |I_{pln}|)^5)$.

Утверждение 44 $Pre_{\mathbf{E}}([\phi]_{env_c}, env_c) = \{(s, i_{pln}) \in S \times I_{pln} \mid \exists (s', i'_{pln}) \in [\phi]_{env_c}, a \in A : (s, a, s') \in env_c \; and \; (see(s), i_{pln}, a, i'_{pln}) \in \sigma_{pln}\} = [\mathbf{E} \bigcirc \phi]_{env_c}$.

Доказательство. Предположим, что существует $(s, i_{pln}) \in Pre_{\mathbf{E}}(x, env_c)$, такие, что $(s, i_{pln}) \notin [\mathbf{E} \bigcirc \phi]_{env_c}$. Значит, для любого пути $\lambda \in out_{env_c}(s, i_{pln})$ выполнено $|\lambda| \leq 1$ или для любого пути $\lambda \in out_{env_c}(s, i_{pln})$ выполнено $\lambda[1] \notin [\phi]_{env_c}$. Из $|\lambda| \leq 1$, по определению $out_{env_c}(s, i_{pln})$, следует, что не существует таких $a \in A$, $i'_{pln} \in I_{pln}$ и $s' \in S$, что $(s, a, s') \in env_c$ и $(see(s), i_{pln}, a, i'_{pln}) \in \sigma_{pln}$, значит $(s, i_{pln}) = \lambda[0] \notin Pre_{\mathbf{E}}(x, env_c)$. Если для любого пути $\lambda \in out_{env_c}(s, i_{pln})$ выполнено $\lambda[1] \notin [\phi]_{env_c}$, то не существует такого действия $a' \in A$, что $(\lambda[0]|_S, a, \lambda[1]|_S) \in env_c$ и $(\lambda[0], \lambda[1]|_{I_{pln}}, a) \in \sigma_{pln}$, следовательно, $(s, i_{pln}) = \lambda[0] \notin Pre_{\mathbf{E}}(x, env_c)$, значит исходное предположение неверно.

Предположим, что существует $(s, i_{pln}) \notin Pre_{\mathbf{E}}(x, env_c)$, такие, что $(s, i_{pln}) \in [\mathbf{E} \bigcirc \phi]_{env_c}$. Значит, существует путь $\lambda \in out_{env_c}(s, i_{pln})$, для которого $\lambda[1] \in [\phi]_{env_c}$, следовательно, по определению $out_{env_c}(s, i_{pln})$, существуют такие $a \in A$,

$i'_{pln} \in I_{pln}$ и $s' \in S$, что $\lambda[1] = (s', i'_{pln}) \in [\phi]_{env_c}$, $(s, a, s') \in env_c$ и $(see(s), i_{pln}, a, i'_{pln}) \in \sigma_{pln}$. Следовательно, $(s, i_{pln}) \in Pre_{\mathbf{E}}(x, env_c)$, значит исходное предположение неверно, что и требовалось доказать. ■

Утверждение 45 *Алгоритм 17 вычисления множества* $[\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t}\phi_2]_{env_c}$ *корректен и полон.*

Доказательство. Для доказательства воспользуемся индукцией по t. Если $t = 0$, то, в соответствии с семантическим правилом, существует путь $\lambda \in out_{env_c}(s, i_{pln})$, такой, что $\lambda[0] \in [\phi_2]_{env_c}$, следовательно, $(s, i_{pln}) = \lambda[0] \in [\phi_2]_{env_c}$. Значит, в случае $t = 0$ множество $[\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t}\phi_2]_{env_c}$ совпадает с множеством $[\phi_2]_{env_c}$, которое и возвращается алгоритмом 17.

При условии, что множество $x = [\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t-1}\phi_2]_{env_c}$ вычислено, воспользовавшись правилом рекурсивного вычисления 4.53 $[\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t}\phi_2]_{env_c}$ можно вычислить следующим образом: $[\phi_2]_{env_c} \cup [\phi_1]_{env_c} \cap [\mathbf{E} \bigcirc \mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t-1}\phi_2]_{env_c} = [\phi_2]_{env_c} \cup Pre_{\mathbf{E}}([\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t-1}\phi_2]_{env_c}, env_c) \cap [\phi_1]_{env_c}$. Так как $[\phi_2]_{env_c} \subseteq [\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t-1}\phi_2]_{env_c} \subseteq [\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t}\phi_2]_{env_c}$, эту формулу можно переписать в виде: $[\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t-1}\phi_2]_{env_c} \cup Pre_{\mathbf{E}}([\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t-1}\phi_2]_{env_c}, env_c) \cap [\phi_1]_{env_c}$, что в точности соответствует одной итерации цикла алгоритма 17. Следовательно, для любых конечных t корректность и полнота алгоритма доказана 17.

Если $t = \infty$, то, согласно семантическому правилу для $\phi_1\,\mathcal{U}_{\leq t}\phi_2$, для любых $(s, i_{pln}) \in [\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t}\phi_2]_{env_c}$ существуют конечное i и путь $\lambda \in out_{env_c}(s, i_{pln})$ такие, что $\lambda[i+1]\,[\phi_2]_{env_c}$ и для любого $0 \leq j \leq i$ выполнено $\lambda[j] \in [\phi_1]$. Следовательно, $(s, i_{pln}) \in [\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq i+1}\phi_2]_{env_c}$. Так как множество $S \times I_{pln}$ конечно, существует максимальное $i = k$, значит $[\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t}\phi_2]_{env_c} \subseteq [\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq k+1}\phi_2]_{env_c}$. А так как множество $[\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t}\phi_2]_{env_c}$ монотонно растет с ростом t, следовательно, $[\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t}\phi_2]_{env_c} = [\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq k+1}\phi_2]_{env_c}$, значит искомое множество будет найдено на $k+1$-ой итерации цикла, что и требовалось доказать. ■

Утверждение 46 *Сложность алгоритма 17 для вычисления множества* $[\mathbf{E}\,\phi_1\,\mathcal{U}_{\leq t}\phi_2]_{env_c}$ *не превосходит* $O((|S|\cdot|I_{pln}|)^4)$.

Доказательство. Алгоритм производит вычисления в цикле, количество итерации которого не превосходит $|S\times I_{pln}|$, так как множество x монотонно растет. При этом на каждом шаге цикла производится вычисление прообраза (сложность не больше $O((|S|\times|I_{pln}|)^3)$) и две теоретико-множественные операции (сложность не больше $O((|S|\times|I_{pln}|)^2)$), следовательно, сложность каждой итерации цикла не превосходит $O((|S|\cdot|I_{pln}|)^3)$. Значит, общая сложность алгоритма не превосходит $O((|S|\cdot|I_{pln}|)^4)$, что и требовалось доказать. ■

А.1.5. Вычисление операторов Bel , Des и Intend

Доказательство корректности и полноты алгоритма вычисления $[\mathsf{Bel}\,\phi]_{env_c}$ приведены в утверждении 47. Алгоритмическая сложность $[\mathsf{Bel}\,\phi]_{env_c}$, как показано в утверждении 48, не превосходит $O(|S|^3\cdot|I_{pln}|)$. Корректность и полнота вычисления $[\mathsf{Des}\,]_{env_c}$ и $[\mathsf{Intend}\,]_{env_c}$ очевидна, так как алгоритм вычисления в точности соответствует семантическому правилу. Алгоритмическая сложность вычисления $[\mathsf{Des}\,]_{env_c}$ ограничена константой (проверка наличия элемента в хэш-таблице), а сложность вычисления $[\mathsf{Intend}\,]_{env_c}$ не превосходит сложности вычисления $[\mathsf{Bel}\,\phi]_{env_c}$ и сложности вычисления пересечения множеств, что в сумме дает оценку $O(|S|^3\cdot|I_{pln}|^2)$.

Утверждение 47 *Для любых* $s\in S$ *и* $i\in I_{pln}$, *если* $(M,i),env_c,(s,i_{pln})\models_s \mathsf{Bel}\,\phi$, *то* $(s,i_{pln})\in[\mathsf{Bel}\,\phi]_{env_c}$ *и наоборот.*

Доказательство.Предположим, что существуют $s\in S$ и $i\in I_{pln}$, такие, что $(M,i),env_c,(s,i_{pln})\models_s \mathsf{Bel}\,\phi$ и $(s,i_{pln})\notin[\mathsf{Bel}\,\phi]_{env_c}$. Если $(M,i),env_c,(s,i_{pln})\models_s \mathsf{Bel}\,\phi$, то для любого $s'\in see(s)$ выполнено $(M,i),i|_{bel},(s',i_{pln})\models_s\phi$, следовательно, для любого $s'\in see(s)$ выполнено $(s',i_{pln})\in[\phi]_{i_{bel}}$. Следовательно, $(s,i_{pln})\in\{(s,i_{pln})\in S\times I_{pln}\mid\forall s'\in see(s):(s',i_{pln})\in[\psi]_{i|_{bel}}\}=[\mathsf{Bel}\,\psi]_{env_c}$, что

противоречит исходному предположению.

Предположим, что существуют $s \in S$ и $i \in I_{pln}$, такие, что $(M, i), env_c, (s, i_{pln}) \not\models_s \mathsf{Bel}\ \phi$ и $(s, i_{pln}) \in [\mathsf{Bel}\ \phi]_{env_c}$. Если $(M, i), env_c, (s, i_{pln}) \not\models_s \mathsf{Bel}\ \phi$, то существует $s' \in see(s)$, для которого $(M, i), i|_{bel}, (s', i_{pln}) \not\models_s \phi$, следовательно, $(s', i_{pln}) \in [\phi]_{i_{bel}}$. Следовательно, $(s, i_{pln}) \notin \{(s, i_{pln}) \in S \times I_{pln} \mid \forall s' \in see(s) : (s', i_{pln}) \in [\psi]_{i|_{bel}}\} = [\mathsf{Bel}\ \psi]_{env_c}$, что противоречит исходному предположению, что и требовалось доказать. ■

Утверждение 48 *Алгоритмическая сложность вычисления множества* $[\mathsf{Bel}\ \phi]_{env_c}$ *не превосходит* $O(|S|^3 \cdot |I_{pln}|)$.

Доказательство. Сложность вычисления множества $[\mathsf{Bel}\ \phi]_{env_c}$ не превосходит сложности вычисления прообраза множества $S \setminus [\phi]_{i|_{bel}}|_S$ относительно отношения *see* и вычисления пересечения $[\phi]_{i|_{bel}} \cap (S \setminus see(S \setminus [\phi]_{i|_{bel}}|_S)) \times I_{pln}$. Учитывая то, что сложность построения дополнения множества является константой, сложность построения проекции не превосходит $O(|S| \cdot |I_{pln}| \cdot log(|S|))$, прообраза — $O(|S|^3)$, а построения пересечения — $O(|S|^2 \times I_{pln})$, общая сложность алгоритма не превосходит $O(|S|^3 \cdot |I_{pln}|)$. ■

A.1.6. Общая алгоритмическая сложность

Общая вычислительная сложность алгоритма 17, как показано в утверждении 49, не превосходит полинома от максимальной размерности функциональных и предикатных символов и мощности множеств A, S и I_{pln}. Заметим, что приведенные оценки являются грубыми оценками сверху и практическая реализация алгоритмов с учетом способов оптимизации, применяемых в современных инструментах символической верификации, могут существенно эту оценку уменьшить.

Утверждение 49 *Вычислительная сложность алгоритма 17 не превосходит* $O(|\phi| \cdot |S|^{5+2 \cdot i_{\max}} \cdot |A|^{3+2 \cdot i_{\max}} \cdot \log(|A|)^2 \cdot |I_{pln}|^5 \cdot i_{\max})$, *где* $|\phi|$ *есть длинна фор-*

мулы[1], $i_{\max}$ *есть максимальная размерность функциональных и предикатных символов, а* $|S|$, $|A|$ *и* $|I_{pln}|$ *есть мощность соответствующего множества.*

Доказательство. Алгоритм вычисляется рекурсивно по структуре формулы, при этом число рекурсивных вызовов не превосходит размера формулы $|\phi|$, а каждый шаг может состоять из следующих действий.

- Вычисление суперпозиции функций при обработке пердикатных и функциональных символов в термах. Сложность этого шага не превышает $O(|S|^{3+2\cdot i_{\max}} \cdot \log(|S|)^2 \cdot i_{\max})$ для термов состояний и $O(|A|^{3+2\cdot i_{\max}} \cdot \log(|A|)^2 \cdot i_{\max})$ для термов действий.

- Вычисляется пересечение, объединение или дополнение множества для обработки логических связок. Для формул состоянии сложность операции в данном случае не превосходит $O((|S| \cdot |I_{pln}|)^2)$, а для формул действий - $O(|A|^2)$.

- Вычисление ограниченного прообраза при обработке оператора $[\alpha]\phi$. Сложность этого шага не превосходит $O(|A|^2 \cdot |S|^3)$.

- Вычисление прообраза при обработке оператора $\bigcirc\phi$, сложность которого не превосходит $O((|S| \cdot |I_{pln}|)^3)$.

- Вычисление цикла при обработке оператора $\phi_1 \, \mathcal{U}_{\leq t} \phi_2$, сложность которого не превосходит $O((|S| \cdot |I_{pln}|)^4)$.

- Вычисление оператора $\mathsf{Bel}\ \phi$, сложность которого не превосходит $O(|S|^3 \cdot |I_{pln}|)$.

- Вычисление оператора $\mathsf{Des}\ \phi$, сложность которого не превышает константы (поиск в хэш-таблице).

[1]Число логических связок, предметных переменных, темпоральных операторов и операторов динамической логики, предикатных и функциональных символов в формуле ϕ.

- Вычисление оператора $\mathsf{Intend}\ \phi$, сложность которого не превышает $O(|S|^3 \cdot |I_{pln}|^2)$.

Кроме того, алгоритм включает шаг предварительной подготовки — вычисление множества $env_c||\sigma'$, сложность которого не превосходит $O(|S|^5 \cdot |I_{pln}|^4 \cdot |A|^2 \cdot \log(|S|))$.
В итоге, общая алгоритмическая сложность не превышает $O(|\phi| \cdot |S|^{5+2\cdot i_{\max}} \cdot |A|^{3+2\cdot i_{\max}} \cdot \log(|A|)^2 \cdot |I_{pln}|^4 \cdot i_{\max})$. ■

A.2. Анализ корректности, полноты и сложности алгоритмов верификации для логики спецификации мультиагентной системы

Корректность и полнота алгоритмов для логики спецификации мультиагентной системы очевидным образом вытекает из корректности полноты алгоритма 17 и корректности и полноты построения представления коалиции-агента и сужения коалиции. При этом для построения представления коалиции-агента используются только очевидные теоретико-множественные операции и функции, корректность и полнота которых были доказаны ранее, поэтому доказательство их полноты и корректности можно провести методом “от противного”, по аналогии с утверждениями 43 и 40.

A.2.1. Построение восприятие коалиции

Сложность построения восприятия коалиции не превосходит сложности обхода списка из $|AG| \cdot \log(|S|)$ компонент, то есть, не превосходит $O(|AG| \cdot \log(|S|))$. Корректность алгоритма построения восприятия коалиции доказывается в утверждении 50.

Утверждение 50 *Для любых $s, s' \in S$ если $see^C(s) = see^C(s')$, то для любого агента коалиции $ag \in C$ выполнено $see_{ag}(s) = see_{ag}(s')$ и наоборот.*

Доказательство. Предположим, что существуют такие $s, s' \in S$, что $see^C(s) = see^C(s')$ и существует такой агент $ag \in C$, что $see_{ag}(s) \neq see_{ag}(s')$. Следовательно, $(s, s') \notin see_{ag}$, а значит и $(s, s') \notin \bigcap\limits_{ag \in C} see_{ag}$. Следовательно, по определению see^C, $(s, s') \notin see_{ag}$ и $see^C(s) \neq see^C(s')$.

Предположим, что существуют такие $s, s' \in S$, что $see^C(s) \neq see^C(s')$ и для любого агента $ag \in C$ выполнено $see_{ag}(s) \neq see_{ag}(s')$. Следовательно, $(s, s') \in see_{ag}$ для любого $ag \in C$, а значит и $(s, s') \in \bigcap\limits_{ag \in C} see_{ag}$. Следовательно, по определению see^C, $(s, s') \in see_{ag}$ и $see^C(s) = see^C(s')$, что и требовалось доказать. ■

А.2.2. Построение представлений коалиции

Вычисление представлений коалиции включает вычисление объединения $|AG|$ множеств из не более чем $|S| \cdot |ACS|$ элементов, объединения $|AG|$ множеств из не более чем $|S|^2 \cdot |ACS|$ элементов и пересечения множества из не более чем $|S| \cdot |ACS|$ элементов со множеством из не более чем $|S|^2 \cdot |ACS|$ элементов, что дает верхнюю оценку сложности $O(|AG| \cdot |S|^2 \cdot |ACS|)$. Корректность предложенного метода построения представлений коалиции доказана в утверждении 51.

Утверждение 51 $\bigcup\limits_{ag \in C} (bel_{ag}) \cap \bigcup\limits_{ag \in C} (sbel_{ag} \times S) = bel^C$.

Доказательство. Предположим, что существуют такие $s, s' \in S$ и $a \in ACS$, что $(s, a, s') \in \bigcup\limits_{ag \in C} (bel_{ag}) \cap \bigcup\limits_{ag \in C} (sbel_{ag} \times S)$ и $(s, a, s') \notin bel^C$. Следовательно, по определению bel^C, для любого агента $ag \in C$ выполнено $(s, a, s') \notin bel_{ag}$ или $(s, a) \notin sbel_{ag}$. Значит $(s, a, s') \notin \bigcup\limits_{ag \in C} (bel_{ag})$ или $(s, a, s') \notin \bigcup\limits_{ag \in C} (sbel_{ag} \times S)$, следовательно, $(s, a, s') \notin \bigcup\limits_{ag \in C} (bel_{ag}) \cap \bigcup\limits_{ag \in C} (sbel_{ag} \times S)$.

Предположим, что существуют такие $s, s' \in S$ и $a \in ACS$, что $(s,a,s') \notin \bigcup\limits_{ag\in C}(bel_{ag}) \cap \bigcup\limits_{ag\in C}(sbel_{ag} \times S)$ и $(s,a,s') \in bel^C$. Следовательно, по определению bel^C, существует агент такой агент $ag \in C$, что $(s,a,s') \in bel_{ag}$, и такой агент $ag' \in C$, что $(s,a) \in sbel_{ag}$. Значит $(s,a,s') \in \bigcup\limits_{ag\in C}(bel_{ag})$ и $(s,a,s') \in \bigcup\limits_{ag\in C}(sbel_{ag} \times S)$, следовательно, $(s,a,s') \in \bigcup\limits_{ag\in C}(bel_{ag}) \cap \bigcup\limits_{ag\in C}(sbel_{ag} \times S)$, что и требовалось доказать. ■

A.2.3. Построение плана коалиции

Корректность и полнота построения плана коалиции доказана в утверждении 52. При этом наиболее сложной операцией при вычислении σ^C_{pln} является построение не более чем $|AG| \cdot \log(|S| \cdot \prod\limits_{ag\in AG}|Sig_{ag}|)$ композиций функции не более чем от $\log(|S|^2 \cdot \prod\limits_{ag\in AG}|Sig_{ag}| \cdot |I^{AG}_{pln}|^2 \cdot |ACS|)$ переменных с функцией не более чем от $\log(|S| \cdot |I^{AG}_{pln}|)$ переменных. Следовательно, алгоритмическая сложность построения плана коалиции не превосходит $O(|S|^5 \cdot |I^{AG}_{pln}|^5 \cdot (\prod\limits_{ag\in AG}|Sig_{ag}|)^2 \cdot |ACS|^2 \cdot |AG| \cdot \log(|S| \cdot \prod\limits_{ag\in AG}|Sig_{ag}|))$.

Утверждение 52 $\bigcap\limits_{ag\in C}\sigma^r_{ag,pln}(sas(s, i^C_{pln})) = \sigma^C_{pln}$.

Доказательство. Предположим, что существуют такие $s \in S$, $i^C_{pln}, i'^C_{pln} \in I^C_{pln}$ и $a \in A^C$, что $(s, i^C_{pln}, i'^C_{pln}, a) \in \sigma^C_{pln}$ и $(s, i^C_{pln}, i'^C_{pln}, a) \notin \bigcap\limits_{ag\in C}\sigma^r_{ag,pln}(sas(s, i^C_{pln}))$. Следовательно, по определению σ^C_{pln}, для любого агента коалиции $ag \in C$ и для любого набора сигналов всех агентов системы $sig = (sig_{ag_1}, \ldots, sig_{ag_n}) \in Sig_{ag_1} \times \ldots \times Sig_{ag_n}$, если для любого агента коалиции $ag' \in C$ отправленный им сигнал определяется планом $sig|_{Sig_{ag'}} = send_{ag',pln}(p^C|_{ag'}, i^C_{pln}|_{I_{ag',pln}}, ag)$, то $(p^C|_{ag}, sig, i^C_{pln}|_{I_{ag,pln}}, i'^C_{pln}|_{I_{ag,pln}}, a^C|_{A_{ag}}) \in \sigma^r_{ag,pln}$. Значит, для любого агента $ag \in C$ выполнено $(see(s), (send_{ag',pln}(s, i^C_{pln}|_{I_{ag,pln}}))_{ag'\in C}, i^C_{pln}|_{I_{ag,pln}}, i'^C_{pln}|_{I_{ag,pln}}, a^C|_{A_{ag}}) \in \sigma^r_{ag,pln}$, а значит, по определению sas, и $(s, i^C_{pln}, i'^C_{pln}, a) \in \sigma^r_{ag,pln}(sas(s, i^C_{pln}))$. Следовательно, $(s, i^C_{pln}, i'^C_{pln}, a) \in \bigcap\limits_{ag\in C}\sigma^r_{ag,pln}(sas(s, i^C_{pln}))$.

Предположим, что существуют такие $s \in S$, $i^C_{pln}, i'^C_{pln} \in I^C_{pln}$ и $a \in A^C$, что $(s, i^C_{pln}, i'^C_{pln}, a) \notin \sigma^C_{pln}$ и $(s, i^C_{pln}, i'^C_{pln}, a) \in \bigcap\limits_{ag \in C} \sigma^r_{ag,pln}(sas(s, i^C_{pln}))$. Следовательно, по определению σ^C_{pln}, существуют такие агент коалиции $ag \in C$ и набор сигналов всех агентов системы $sig = (sig_{ag_1}, \ldots, sig_{ag_n}) \in Sig_{ag_1} \times \ldots \times Sig_{ag_n}$, что для любого агента коалиции $ag' \in C$ отправленный им сигнал определяется планом $sig|_{Sig_{ag'}} = send_{ag',pln}(p^C|_{ag'}, i^C_{pln}|_{I_{ag',pln}}, ag)$ и $(p^C|_{ag}, sig, i^C_{pln}|_{I_{ag,pln}}, i'^C_{pln}|_{I_{ag,pln}}, a^C|_{A_{ag}}) \notin \sigma^r_{ag,pln}$. Значит, для этого агента $ag \in C$ выполнено $(see(s), (send_{ag',pln}(s, i^C_{pln}|_{I_{ag,pln}}))_{ag' \in C}, i^C_{pln}|_{I_{ag,pln}}, i'^C_{pln}|_{I_{ag,pln}}, a^C|_{A_{ag}}) \notin \sigma^r_{ag,pln}$, а значит, по определению sas, и $(s, i^C_{pln}, i'^C_{pln}, a) \notin \sigma^r_{ag,pln}(sas(s, i^C_{pln}))$. Следовательно, $(s, i^C_{pln}, i'^C_{pln}, a) \in \bigcap\limits_{ag \notin C} \sigma^r_{ag,pln}(sas(s, i^C_{pln}))$, что и требовалось доказать. ■

A.2.4. Построение сужения коалиции

Сложность построения сужения коалиции определяется сложностью построения восприятия, представлений, плана и внешней среды коалиции. Из перечисленных шагов наиболее сложным является построение плана коалиции, сложность которого не превышает $O(|S|^5 \cdot |I^{AG}_{pln}|^5 \cdot (\prod\limits_{ag \in AG} |Sig_{ag}|)^2 \cdot |ACS|^2 \cdot |AG| \cdot \log(|S| \cdot \prod\limits_{ag \in AG} |Sig_{ag}|))$. При этом корректность и полнота алгоритма построения внешней среды коалиции доказана в утверждении 53, а корректность и полнота остальных шагов уже была доказана выше.

Утверждение 53 $env^C|_{S \times A^D \times S} = env^C|_D$.

Доказательство. Предположим, что существуют такие $s, s' \in S$ и $a^D \in A^D$, что $(s, a^D, s') \in env^C|_{S \times A^D \times S}$ и $(s, a^D, s') \notin env^C|_D$. Следовательно, по определению $env|_D$, для любого набора действий $a^{C \setminus D} \in A^{C \setminus D}$ выполнено $(s, a^D + a^{C \setminus D}, s') \notin env^C$. Следовательно, $(s, a, s') \notin env^C|_{S \times A^D \times S}$.

Предположим, что существуют такие $s, s' \in S$ и $a^D \in A^D$, что $(s, a^D, s') \notin env^C|_{S \times A^D \times S}$ и $(s, a^D, s') \in env^C|_D$. Следовательно, по определению $env|_D$, существует такой набор действий $a^{C \setminus D} \in A^{C \setminus D}$, что $(s, a^D + a^{C \setminus D}, s') \in env^C$.

Следовательно, $(s, a, s') \in env^C|_{S \times A^D \times S}$, что и требовалось доказать. ■

A.2.5. Общая алгоритмическая сложность

Утверждение 54 *Вычислительная сложность алгоритма верификации для логики спецификации мультиагентной системы не превосходит* $O(|\phi| \cdot |S|^{5+2 \cdot i_{\max}} \cdot |ACS|^{3+2 \cdot i_{\max}} \cdot \log(|ACS|)^2 \cdot |I_{pln}^{AG}|^5 \cdot (\prod_{ag \in AG} |Sig_{ag}|)^2 \cdot |AG| \cdot \log(\prod_{ag \in AG} |Sig_{ag}|) \cdot i_{\max})$, *где* $|\phi|$ *есть длинна формулы,* $i_{\max}$ *есть максимальная размерность функциональных и предикатных символов, а* $|S|$, $|ACS|$, $|AG|$, $|Sig_{ag}|$ *и* $|I_{pln}|$ *есть мощность соответствующего множества.*

Доказательство. Алгоритм вычисляется рекурсивно по структуре формулы, при этом число рекурсивных вызовов не превосходит размера формулы $|\phi|$, а каждый шаг может состоять из следующих действий.

- Вычисление суперпозиции функций при обработке пердикатных и функциональных символов в термах. Сложность этого шага не превышает $O(|S|^{3+2 \cdot i_{\max}} \cdot \log(|S|)^2 \cdot i_{\max})$ для термов состояний и $O(|ACS|^{3+2 \cdot i_{\max}} \cdot \log(|ACS|)^2 \cdot i_{\max})$ для термов действий.

- Вычисляется пересечение, объединение или дополнение множества для обработки логических связок. Для формул состоянии сложность операции в данном случае не превосходит $O((|S| \cdot |I_{pln}^{AG}|)^2)$, а для формул действий - $O(|ACS|^2)$.

- Вычисление ограниченного прообраза при обработке оператора $[\alpha]\phi$. Сложность этого шага не превосходит $O(|ACS|^2 \cdot |S|^3)$.

- Вычисление прообраза при обработке оператора $\bigcirc\phi$, сложность которого не превосходит $O((|S| \cdot |I_{pln}^{AG}|)^3)$.

- Вычисление цикла при обработке оператора $\phi_1 \, \mathcal{U}_{\leq t} \phi_2$, сложность которого не превосходит $O((|S| \cdot |I_{pln}^{AG}|)^4)$.

- Вычисление оператора $\mathsf{Bel}\ \phi$, сложность которого не превосходит $O(|S|^3 \cdot |I_{pln}^{AG}|)$.
- Вычисление оператора $\mathsf{Des}\ \phi$, сложность которого не превышает константы (поиск в хэш-таблице).
- Вычисление оператора $\mathsf{Intend}\ \phi$, сложность которого не превышает $O(|S|^3 \cdot |I_{pln}^{AG}|^2)$.
- Вычисление сужения коалиции, сложность которого не превосходит $O(|S|^5 \cdot |I_{pln}^{AG}|^5 \cdot (\prod\limits_{ag \in AG} |Sig_{ag}|)^2 \cdot |ACS|^2 \cdot |AG| \cdot \log(|S| \cdot \prod\limits_{ag \in AG} |Sig_{ag}|))$.

В итоге, общая алгоритмическая сложность не превышает $O(|\phi| \cdot |S|^{5+2 \cdot i_{\max}} \cdot |ACS|^{3+2 \cdot i_{\max}} \cdot \log(|ACS|)^2 \cdot |I_{pln}^{AG}|^5 \cdot (\prod\limits_{ag \in AG} |Sig_{ag}|)^2 \cdot |AG| \cdot \log(\prod\limits_{ag \in AG} |Sig_{ag}|) \cdot i_{\max})$.

■

A.3. Анализ корректности и сложности алгоритмов планирования

A.3.1. Формулы без темпоральных операторов

Утверждение 55 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\psi, C, env, \{sup_{ag}\}) = (P^C \setminus see^C(S \setminus [\psi]_{env,C}\,|_S)) \times I_{pln}, \{sup_{ag}\})$, *где* ψ *есть формула, не содержащая темпоральных операторов. Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \psi$.

Доказательство. Набор правил $\{pln_{ag}\} = \{sup_{ag}\}$ очевидно является максимальным набором, содержащимся в $\{sup_{ag}\}$. Кроме того, так как формула ψ не содержит темпоральных операторов, её интерпретация зависит только от состояния внешней среды s.

Предположим, что существует такая пара $(p, i_{pln}) \in ES_0$, что существует такое

$s \in p$, что $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \nvDash_s \psi$. Тогда $s \in S \setminus [\psi]_{env,C}\,|_S$, следовательно, $see^C(s) \in see^C(S \setminus [\psi]_{env,C}\,|_S)$, а значит $p = see^C(s) \notin P^C \setminus see^C(S \setminus [\psi]_{env,C}\,|_S)$ и $(p, i_{pln}) \notin ES_0$, что противоречит исходному предположению. ■

A.3.2. Обработка формул $\psi_1 \wedge \psi_2$

Утверждение 56 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\psi_1 \wedge \psi_2, C, env, \{sup_{ag}\})$. *Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \vDash_s \psi_1 \wedge \psi_2$.

Доказательство. Так как исходные планы не выходят за рамки $\{sup_{ag}\}$, то и результат, построенный на этом шаге и являющийся комбинацией результатов для подформул, не выходит за рамки $\{sup_{ag}\}$.

Предположим, что существуют такие $(p, i_{pln}) \in ES_0$, и $s \in p$, что $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \nvDash_s \psi_1 \wedge \psi_2$. Тогда $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \nvDash_s \psi_1$ или $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \nvDash_s \psi_2$. Учитывая, что $\{pln_{ag}\} = \{pln^1_{ag}\} = \{pln^2_{ag}\}$ получаем $(M, i^C_{\{pln^1_{ag}\}}), C, env, (s, i_{pln}) \nvDash_s \psi_1$ или $(M, i^C_{\{pln^2_{ag}\}}), C, env, (s, i_{pln}) \nvDash_s \psi_2$. Следовательно, $(p, i_{pln}) \notin ES^1_0$ или $(p, i_{pln}) \notin ES^2_0$, а значит и $(p, i_{pln}) \notin ES_0$, что противоречит исходному предположению. ■

A.3.3. Обработка формул $\psi_1 \vee \psi_2$

Утверждение 57 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\psi_1 \vee \psi_2, C, env, \{sup_{ag}\})$. *Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \vDash_s \psi_1 \vee \psi_2$.

Доказательство. Так как исходные планы не выходят за рамки $\{sup_{ag}\}$, то и результат, построенный на этом шаге и являющийся комбинацией результатов для подформул, не выходит за рамки $\{sup_{ag}\}$.

Предположим, что существует такая пара $(p, i_{pln}) \in ES_0$ и такое $s \in p$, что $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \nvDash_s \psi_1 \vee \psi_2$. Тогда $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \nvDash_s \psi_1$

и $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \not\models_s \psi_2$. Для (p, i_{pln}) есть три возможности: $i_{pln} \in C(\psi_1) \cap C(\neg\psi_2)$ (это значит, что $(p, i_{pln}) \in ES^1_0 \cap (P^C \times C(\psi_1) \cap C(\neg\psi_2)))$, $i_{pln} \in C(\neg\psi_1) \cap C(\psi_2)$ (это значит, что $(p, i_{pln}) \in ES^2_0 \cap (P^C \times C(\neg\psi_1) \cap C(\psi_2)))$ или $i_{pln} \in C(\ \psi_1) \cap C(\psi_2)$ (это значит, что $(p, i_{pln}) \in ES^{12}_0 \cap (P^C \times C(\psi_1) \cap C(\psi_2)))$. Если $(p, i_{pln}) \in ES^1_0 \cap (P^C \times C(\neg\psi_2))$, то $(p, i_{pln}) \in ES^1_0$ и $(M, i^C_{\{pln^1_{ag}\}}), C, env$, $(s, i_{pln}) \models_s \psi_1$. Так как маркеры $\neg\psi_2$ и ψ_2 устанавливаются только на этом этапе алгоритма, они не могут повлиять на реализацию ψ_1. Следовательно, $(M, i^C_{\{pln^1_{ag} \cap P^C \times C(\neg\psi_2) \times C(\neg\psi_2) \times A_{ag}\}}), C, env, (s, i_{pln}) \models_s \psi_1$ и, так как $\{pln_{ag} \cap P^C \times C(\neg\psi_2) \times C(\neg\psi_2) \times A_{ag}\} = \{pln^1_{ag} \cap P^C \times C(\neg\psi_1) \times C(\neg\psi_2) \times A_{ag}\}$, следовательно, $(M, i^C_{\{pln_{ag} \cap P^C \times C(\neg\psi_2) \times C(\neg\psi_2) \times A_{ag}\}}), C, env, (s, i_{pln}) \models_s \psi_1$. Так как $i_{pln} \in C(\neg\psi_2)$, начав действовать в этом состоянии, план может применять только те правила переходов, которые помечены маркером $\neg\psi_2$, следовательно, из этого состояния множества правил $\{pln_{ag} \cap P^C \times C(\psi_1) \cap C(\neg\psi_2) \times C(\psi_1) \cap C(\neg\psi_2) \times A_{ag}\}$ и $\{pln_{ag}\}$ ведут к одинаковым результатам. Следовательно, $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \psi_1$, чего не может быть. Значит, $i_{pln} \in C(\neg\psi_1) \cap C(\psi_2)$ или $i_{pln} \in C(\psi_1) \cap C(\psi_2)$. Однако, аналогичными рассуждениями можно показать, что $i_{pln} \notin C(\neg\psi_1) \cap C(\psi_2)$, так как в этом случае $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \psi_2$, и что $i_{pln} \notin C(\psi_1) \cap C(\psi_2)$, так как в этом случае $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \psi_1 \wedge \psi_2$. Следовательно, исходное предположение неверно. ■

A.3.4. Обработка формул $\langle\langle D\rangle\rangle\psi_1$

Утверждение 58 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\langle\langle D\rangle\rangle\psi_1, C, env, \{sup_{ag}\})$. *Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \langle\langle D\rangle\rangle\psi_1$.

Доказательство. Так как все множества правил, объединяемые в результат $\{pln_{ag}\}$, не выходят за рамки $\{sup_{ag}\}$ и $\{sup'_{ag}\} \subseteq \{sup_{ag}\}$, то и результат $\{pln_{ag}\}$ не выходит за рамки $\{sup_{ag}\}$.

Предположим, что существуют такие пара $(p, i_{pln}) \in ES_0$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \not\models_s \langle\langle D\rangle\rangle\psi_1$. Следовательно, $(M, i^D_{\{pln_{ag}\}}), D, env|_D$, $(s, i_{pln}) \not\models_s \psi_1$, а значит для любого $p^{C\setminus D} \in P^{C\setminus D}$ выполнено $(see^D(s) \oplus p^{C\setminus D}$, $i_{pln}) \notin ES_0$ (так как $(ES_0, \{pln_{ag}\}) = find(\psi_1, D, env|_D, \{sup'_{ag}\})$), следовательно, $(p, i_{pln}) = (see^C(s), i_{pln}) \notin ES_0$, что противоречит исходному предположению. ■

A.3.5. Обработка формул A $\bigcirc \psi_1$, E $\bigcirc \psi_1$ и ¬E $\bigcirc$ *true*

Утверждение 59 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\mathbf{A} \bigcirc \psi_1, C, env, \{sup_{ag}\})$. *Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \mathbf{A} \bigcirc \psi_1$.

Доказательство. Так как все множества правил, объединяемые в результат $\{pln_{ag}\}$, не выходят за рамки $\{sup_{ag}\}$, то и результат $\{pln_{ag}\}$ не выходит за рамки $\{sup_{ag}\}$.

Предположим, что существует такие пара $(p, i_{pln}) \in ES_0$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{A} \bigcirc \psi_1$. Следовательно, для пары $(see^C(s), i_{pln})$ не определено ни одного правила в множестве $\{pln_{ag}\}$ или существует такой набор правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag}\}$ для каждого агента коалиции $ag \in C$, действие системы $acs \in ACS$ и состояние внешней среды $s' \in S$, что $(s, acs, s') \in env$, $(M, i^C_{\{pln_{ag}\}}), C, env, (s', i'_{pln}) \not\models_s \psi_1$ и для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$.

Если для для пары $(see^C(s), i_{pln})$ не определено ни одного правила в множестве $\{pln_{ag}\}$, то, очевидно не существует таких i'_{pln} и $acs \in ACS$, что $(see^C(s), i_{pln}$, $i'_{pln}, acs) \in Pre_{\mathbf{A}}(ES^1_0, env^C, \{sup_{ag}\})$, а значит $(see^C(s), i_{pln}) \notin ES_0$.

Если существует такой набор правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag}\}$ для каждого агента коалиции $ag \in C$, действие системы $acs \in ACS$ и состояние внешней среды $s' \in S$, что $(s, acs, s') \in env$, $(M, i^C_{\{pln_{ag}\}}), C, env, (s', i'_{pln}) \not\models_s \psi_1$

и для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$, то $(see^C(s'), i'_{pln}) \notin ES_0^1$ (так как после попадания в одно из состояний из ES_0^1 результаты построенного множества правил эквивалентны результатам множества правил $\{pln_{ag}^1\}$) и, следовательно, $(see^C(s), i_{pln}, i'_{pln}, acs) \in Pre_{\mathbf{E}}(P^C \times I_{pln} \setminus ES_0^1, env, \{sup_{ag}\})$, а значит $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \notin \{pln_{ag}\}$, что противоречит исходному утверждению. ■

Утверждение 60 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\mathbf{E} \bigcirc \psi_1, C, env, \{sup_{ag}\})$. *Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \mathbf{E} \bigcirc \psi_1$.

Доказательство. Так как все множества правил, объединяемые в результат $\{pln_{ag}\}$, не выходят за рамки $\{sup_{ag}\}$, то и результат $\{pln_{ag}\}$ не выходит за рамки $\{sup_{ag}\}$.

Предположим, что существует такие пара $(p, i_{pln}) \in ES_0$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{E} \bigcirc \psi_1$. Следовательно, для пары $(see^C(s), i_{pln})$ не определено ни одного правила в множестве $\{pln_{ag}\}$ или для любых набора правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag}\}$, действия системы $acs \in ACS$ такого, что для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$, и состояния внешней среды $s' \in S$ такого, что $(s, acs, s') \in env$, выполнено $(M, i^C_{\{pln_{ag}\}}), C, env, (s', i'_{pln}) \not\models_s \psi_1$. Если для для пары $(see^C(s), i_{pln})$ не определено ни одного правила в множестве $\{pln_{ag}\}$, то $(see^C(s), i_{pln}) \notin D(\{pln_{ag}\})$, а значит $(p, i_{pln}) \notin ES_0$.

Если для любых набора правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag}\}$, действия системы $acs \in ACS$ такого, что для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$, и состояния внешней среды $s' \in S$ такого, что $(s, acs, s') \in env$, выполнено $(M, i^C_{\{pln_{ag}\}}), C, env, (s', i'_{pln}) \not\models_s \psi_1$, то для любого возможного результата s' выполнено $(see^C(s'), i'_{pln}) \notin ES_0^1$ (так как после попадания в одно из состояний из ES_0^1 результаты построенного множества правил эквивалентны резуль-

татам множества правил $\{pln^1_{ag}\}$) и, следовательно, $(see^C(s), i_{pln}, i'_{pln}, acs) \notin Pre_{\mathbf{E}}(ES^1_0, env, \{sup_{ag}\})$. Значит $\{(see^C(s), i_{pln}\} \notin D(Pre_{\mathbf{E}}(ES^1_0, env, \{sup_{ag}\})$, следовательно, $(p, i_{pln}) \notin ES_0$, что противоречит исходному утверждению. ■

Утверждение 61 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\neg\mathbf{E} \bigcirc true, C, env, \{sup_{ag}\})$. *Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \neg\mathbf{E} \bigcirc true$.

Доказательство. Множество правил $\{pln_{ag}\} = \{\emptyset_{ag}\}$ очевидно не выходят за рамки $\{sup_{ag}\}$.

Предположим, что существует такие пара $(p, i_{pln}) \in P^C \times I_{pln}$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \not\models_s \neg\mathbf{E} \bigcirc true$. Следовательно, $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \neg\bigcirc true$, а значит существует такой набор правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag}\}$ для каждого агента коалиции $ag \in C$, действие системы $acs \in ACS$ и состояние внешней среды $s' \in S$, что $(s, acs, s') \in env$, $(M, i^C_{\{pln_{ag}\}}), C, env, (s', i'_{pln}) \models_s true$ и для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$, что очевидно противоречит тому, что $\{pln_{ag}\} = \{\emptyset_{ag}\}$. ■

А.3.6. Обработка формул $\mathbf{A}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$ и $\mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$

Утверждение 62 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\mathbf{A}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$. *Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \mathbf{A}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2, C, env, \{sup_{ag}\}$.

Доказательство. Так как исходные планы не выходят за рамки $\{sup_{ag}\}$, то и результат, построенный на этом шаге и являющийся комбинацией результатов для подформул, не выходит за рамки $\{sup_{ag}\}$. Для дальнейшего доказательства воспользуемся индукцией по количеству итераций в поиске внутренней неподвижной точки t.

Если $t = 0$, предположим, что существуют такие пара $(p, i_{pln}) \in ES_{0,0}$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag,0}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{A}\,\psi_1\ \mathcal{U}_{\leq 0}\psi_2$. Тогда $(M, i^C_{\{pln_{ag,0}\}}), C, env, (s', i_{pln}) \not\models_s \psi_2$, следовательно, $(see^C(s), i_{pln}) \notin ES^2_0$ и, так как $ES_{0,0} \subseteq ES^2_0$ (для $t = 0$), $(p, i_{pln}) \notin ES_{0,0}$, что противоречит исходному утверждению.

Если число итераций равно $t + 1$ и для t корректность уже доказана, предположим, что существуют такие пара $(p, i_{pln}) \in ES_{0,t+1}$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{A}\,\psi_1\ \mathcal{U}_{\leq t+1}\psi_2$. Тогда, в соответствии с правилом рекурсивного вычисления 4.52, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_2 \vee (\psi_1 \wedge \mathbf{A} \bigcirc \mathbf{A}\,\psi_1\ \mathcal{U}_{\leq t}\psi_2)$, следовательно, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_2$ и $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s (\psi_1 \wedge \mathbf{A} \bigcirc \mathbf{A}\,\psi_1\ \mathcal{U}_{\leq t}\psi_2)$.

При этом, если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s', i_{pln}) \not\models_s \psi_2$, то $(see^C(s), i_{pln}) = (p, i_{pln}) \notin ES^2_0 \cap (P^C \times C(\psi_2))$.

Если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s (\psi_1 \wedge \mathbf{A} \bigcirc \mathbf{A}\,\psi_1\ \mathcal{U}_{\leq t}\psi_2)$, то $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_1$ или $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{A} \bigcirc \mathbf{A}\,\psi_1\ \mathcal{U}_{\leq t}\psi_2$. Если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_1$, то $(see^C(s), i_{pln}) \notin ES^1_0$ и, следовательно, $(p, i_{pln}) \notin ES^1_0$, а значит, так как $ES_{0,t+1} \subseteq ES^1_0 \cup ES^2_0 \cap (P^C \times C(\psi_2))$ и $(p, i_{pln}) \notin ES^2_0 \cap (P^C \times C(\psi_2))$, $(p, i_{pln}) \notin ES_{0,t+1}$, что противоречит исходному предположению.

Если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{A} \bigcirc \mathbf{A}\,\psi_1\ \mathcal{U}_{\leq t}\psi_2$, то для пары (s, i_{pln}) в множестве правил $\{pln_{ag,t+1}\}$ не задано ни одного правила или существует такой набор правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag,t+1}\}$ для каждого агента коалиции $ag \in C$, действие системы $acs \in ACS$ и состояние внешней среды $s' \in S$, что $(s, acs, s') \in env$, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s', i'_{pln}) \not\models_s \mathbf{A}\,\psi_1\ \mathcal{U}_{\leq t}\psi_2)$ и для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$.

Если для пары (s, i_{pln}) в множестве правил $\{pln_{ag,t+1}\}$ не задано ни одного правила, то $(see^C(s), i_{pln}) \notin D(\{pln_{ag,t+1}\})$, следовательно, так как множество правил на каждом шаге расширяется, $(see^C(s), i_{pln}) \in ES^2_0 \cap (P^C \times C(\psi_2))$, что, однако, противоречит с $(p, i_{pln}) \notin ES^2_0 \cap (P^C \times C(\psi_2))$.

Если существует такой набор правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag,t+1}\}$ для каждого агента коалиции $ag \in C$, действие системы $acs \in ACS$ и состояние внешней среды $s' \in S$, что $(s, acs, s') \in env$, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s', i'_{pln}) \not\models_s \mathbf{A}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$ и для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$, то $(see^C(s), i_{pln}, i'_{pln}, a_{ag}) \in Pre_{\mathbf{E}}(P^C \times I_{pln} \setminus ES_{0,t}, env, \{pln^1_{ag,1}\})$. Следовательно, $(see^C(s), i_{pln}) \notin ES_{0,t+1} \setminus ES_{0,t}$ и, так как $p = see^C(s)$ и $ES_{0,t} \subseteq ES_{0,t+1}$, $(p, i_{pln}) \in ES_{0,t}$. Но, если $(p, i_{pln}) \in ES_{0,t}$, то действия агентов, начинающиеся в этом состоянии подчиняются только правилам из плана $\{pln_{ag,t}\}$ (на каждой итерации внутреннего цикла к плану добавляются только те правила, левые части которых не были рассмотрены раньше) и добиваются цели на шаг быстрее максимального срока $t+1$, следовательно, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \models_s \mathbf{A}\,\psi_1\,\mathcal{U}_{\leq t+1}\psi_2$, что противоречит исходному предположению.

Следовательно, для любого конечного числа итераций поиска внутренней неподвижной точки корректность алгоритма доказана. Так как все рассматриваемые множества конечны и множество ES_0 монотонно растет, следовательно поиск внутренней неподвижной точки завершается за конечное число итераций, что и требовалось доказать. ■

Утверждение 63 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$. *Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2, C, env, \{sup_{ag}\}$.

Доказательство. Так как исходные планы не выходят за рамки $\{sup_{ag}\}$, то и результат, построенный на этом шаге и являющийся комбинацией результатов для подформул, не выходит за рамки $\{sup_{ag}\}$. Для дальнейшего доказательства воспользуемся индукцией по количеству итераций в поиске неподвижной точки t.

Если $t = 0$, предположим, что существуют такие пара $(p, i_{pln}) \in ES_{0,0}$ и состоя-

ние $s \in p$, что $(M, i^C_{\{pln_{ag,0}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq 0}\psi_2$. Тогда $(M, i^C_{\{pln_{ag,0}\}}), C, env, (s', i_{pln}) \not\models_s \psi_2$, следовательно, $(see^C(s), i_{pln}) \notin ES^2_0$ и, так как $ES_{0,0} \subseteq ES^2_0$ (для $t = 0$), $(p, i_{pln}) \notin ES_{0,0}$, что противоречит исходному утверждению.

Если число итераций равно $t + 1$ и для t корректность уже доказана, предположим, что существуют такие пара $(p, i_{pln}) \in ES_{0,t+1}$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t+1}\psi_2$. Тогда, в соответствии с правилом рекурсивного вычисления 4.53, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_2 \vee (\psi_1 \wedge \mathbf{E} \bigcirc \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2)$, следовательно, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_2$ и $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s (\psi_1 \wedge \mathbf{E} \bigcirc \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2)$.

При этом, если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s', i_{pln}) \not\models_s \psi_2$, то $(see^C(s), i_{pln}) = (p, i_{pln}) \notin ES^2_0 \cap (P^C \times C(\psi_2))$.

Если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s (\psi_1 \wedge \mathbf{E} \bigcirc \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2)$, то $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_1$ или $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{E} \bigcirc \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$. Если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_1$, то $(see^C(s), i_{pln}) \notin ES^1_0$ и, следовательно, $(p, i_{pln}) \notin ES^1_0$, а значит, так как $ES_{0,t+1} \subseteq ES^1_0 \cup ES^2_0 \cap (P^C \times C(\psi_2))$ и $(p, i_{pln}) \notin ES^2_0 \cap (P^C \times C(\psi_2))$, $(p, i_{pln}) \notin ES_{0,t+1}$, что противоречит исходному предположению.

Если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{E} \bigcirc \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$, то для пары (s, i_{pln}) в множестве правил $\{pln_{ag,t+1}\}$ не задано ни одного правила или для любых набора правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag}\}$, действия системы $acs \in ACS$ такого, что для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$, и состояния внешней среды $s' \in S$ такого, что $(s, acs, s') \in env$, выполнено $(M, i^C_{\{pln_{ag}\}}), C, env, (s', i'_{pln}) \not\models_s \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t}\psi_2$.

Если для пары (s, i_{pln}) в множестве правил $\{pln_{ag,t+1}\}$ не задано ни одного правила, то $(see^C(s), i_{pln}) \notin D(\{pln_{ag,t+1}\})$, следовательно, так как множество правил на каждом шаге расширяется, $(see^C(s), i_{pln}) \in ES^2_0 \cap (P^C \times C(\psi_2))$, что, однако, противоречит с $(p, i_{pln}) \notin ES^2_0 \cap (P^C \times C(\psi_2))$.

Если для любых набора правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag}\}$, действия си-

стемы $acs \in ACS$ такого, что для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$, и состояния внешней среды $s' \in S$ такого, что $(s, acs, s') \in env$, выполнено $(M, i^C_{\{pln_{ag}\}}), C, env, (s', i'_{pln}) \nvDash_s \mathbf{E}\,\psi_1\ \mathcal{U}_{\leq t}\psi_2$, то для любого возможного результата выполнено $(see(s'), i'_{pln}) \notin ES_{0,t}$, так как в множестве $ES_{0,t}$ находятся только те состояния, для которых искомый путь уже найден. Следовательно, $(see^C(s), i_{pln}, i'_{pln}, a_{ag}) \notin Pre_{\mathbf{E}}(ES_{0,t}, env, \{pln^1_{ag,1}\})$. Значит, $(see^C(s), i_{pln}) \notin ES_{0,t+1} \setminus ES_{0,t}$ и, так как $p = see^C(s)$ и $ES_{0,t} \subseteq ES_{0,t+1}$, $(p, i_{pln}) \in ES_{0,t}$. Но множество $ES_{0,t}$ содержит только те состояния, из которых существует искомый путь, длинна которого не превосходит t, следовательно, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env$, $(s, i_{pln}) \vDash_s \mathbf{E}\,\psi_1\,\mathcal{U}_{\leq t+1}\psi_2$, что противоречит исходному предположению.

Следовательно, для любого конечного числа итераций поиска неподвижной точки корректность алгоритма доказана. Так как все рассматриваемые множества конечны и множество ES_0 монотонно растет, следовательно поиск внутренней неподвижной точки завершается за конечное число итераций, что и требовалось доказать. ■

A.3.7. Обработка формул $\mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2$ и $\mathbf{E}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2$

Утверждение 64 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$. *Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \vDash_s \mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2, C, env, \{sup_{ag}\}$.

Доказательство. Так как исходные планы не выходят за рамки $\{sup_{ag}\}$, то и результат, построенный на этом шаге и являющийся комбинацией результатов для подформул, не выходит за рамки $\{sup_{ag}\}$. Для дальнейшего доказательства воспользуемся индукцией по количеству итераций в поиске внутренней неподвижной точки t.

Если $t = 0$, предположим, что существуют такие пара $(p, i_{pln}) \in ES_{0,0}$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag,0}\}}), C, env, (s, i_{pln}) \nvDash_s \mathbf{A}\,\psi_1\ \mathcal{W}_{\leq 0}\psi_2$. Тогда $(M, i^C_{\{pln_{ag,0}\}})$,

$C, env, (s, i_{pln}) \not\models_s \psi_1$, следовательно, $(see^C(s), i_{pln}) \notin ES_0^1$ и, так как $ES_{0,0} \subseteq ES_0^1$, $(p, i_{pln}) \notin ES_{0,0}$, что противоречит исходному утверждению.

Если количество итераций равно $t+1$ и для t корректность уже доказана, предположим, что существуют такие пара $(p, i_{pln}) \in ES_{0,t+1}$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t+1}\psi_2$. Тогда, в соответствии с правилом рекурсивного вычисления 4.54, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_1 \wedge (\psi_2 \vee \mathbf{A} \bigcirc \mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2)$, следовательно, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_1$ или $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s (\psi_2 \vee \mathbf{A} \bigcirc \mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2)$.

При этом, если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_1$, то $(see^C(s), i_{pln}) \notin ES_0^1$ и, так как $ES_{0,t+1} \subseteq ES_0^1$, $(p, i_{pln}) \notin ES_{0,t+1}$, что противоречит исходному утверждению.

Если же выполнено $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s (\psi_2 \vee \mathbf{A} \bigcirc \mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2)$, то $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_2$ и $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{A} \bigcirc \mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2$.

Если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{A} \bigcirc \mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2$, то для пары (s, i_{pln}) в множестве правил $\{pln_{ag,t+1}\}$ не задано ни одного правила или существует такой набор правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag,t+1}\}$ для каждого агента коалиции $ag \in C$, действие системы $acs \in ACS$ и состояние внешней среды $s' \in S$, что $(s, acs, s') \in env$, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s', i'_{pln}) \not\models_s \mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2$ и для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$.

Если для пары (s, i_{pln}) в множестве правил $\{pln_{ag,t+1}\}$ не задано ни одного правила, то $(see^C(s), i_{pln}) \notin D(\{pln_{ag,t+1}\}$, следовательно, $(p, i_{pln}) \notin ES_{0,t+1}$.

Если существует такой набор правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag,t+1}\}$ для каждого агента коалиции $ag \in C$, действие системы $acs \in ACS$ и состояние внешней среды $s' \in S$, что $(s, acs, s') \in env$, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s', i'_{pln}) \not\models_s \mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2$ и для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$, то $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in Pre_{\mathbf{E}}(P^C \times I_{pln} \setminus ES_{0,t}, env, \{P^C \times C(\neg\psi_2) \times I_{pln} \times A_{ag}\})$, а значит $(see^C(s), i_{pln}) \in ES_0^1 \cap ES_0^2 \cap P^C \times C(\psi_2)$. Если $(see^C(s), i_{pln}) \in ES_0^1 \cap ES_0^2 \cap P^C \times$

$C(\psi_2)$, то действия коалиции определяются дальше исключительно правилами $\{pln^2_{ag}\}$, следовательно, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \models_s \psi_2$, что противоречит тому, что $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_2$.

Следовательно, для любого конечного числа итераций поиска внутренней неподвижной точки корректность алгоритма доказана. Так как все рассматриваемые множества конечны и множество ES_0 монотонно убывает, следовательно поиск внутренней неподвижной точки завершается за конечное число итераций, что и требовалось доказать. ■

Утверждение 65 *Пусть* $(ES_0, \{pln_{ag}\}) = find(\mathbf{E}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2, C, env, \{sup_{ag}\})$. *Тогда для любого* $ag \in C$ *выполнено* $pln_{ag} \subseteq sup_{ag}$, *а также для любой пары* $(p, i_{pln}) \in ES_0$ *и для любого* $s \in p$ *выполнено* $(M, i^C_{\{pln_{ag}\}}), C, env, (s, i_{pln}) \models_s \mathbf{E}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2, C, env, \{sup_{ag}\}$.

Доказательство. Так как исходные планы не выходят за рамки $\{sup_{ag}\}$, то и результат, построенный на этом шаге и являющийся комбинацией результатов для подформул не выходит за рамки $\{sup_{ag}\}$. Для дальнейшего доказательства воспользуемся индукцией по количеству итераций в поиске неподвижной точки t.

Если $t = 0$, предположим, что существуют такие пара $(p, i_{pln}) \in ES_{0,0}$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag,0}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{E}\,\psi_1\ \mathcal{W}_{\leq 0}\psi_2$. Тогда $(M, i^C_{\{pln_{ag,0}\}})$, $C, env, (s, i_{pln}) \not\models_s \psi_1$, следовательно, $(see^C(s), i_{pln}) \notin ES^1_0$ и, так как $ES_{0,0} \subseteq ES^1_0$, $(p, i_{pln}) \notin ES_{0,0}$, что противоречит исходному утверждению.

Если количество итераций равно $t+1$ и для t корректность уже доказана, предположим, что существуют такие пара $(p, i_{pln}) \in ES_{0,t+1}$ и состояние $s \in p$, что $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \mathbf{E}\,\psi_1\ \mathcal{W}_{\leq t+1}\psi_2$. Тогда, в соответствии с правилом рекурсивного вычисления 4.55, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_1 \wedge (\psi_2 \vee \mathbf{E} \bigcirc \mathbf{E}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2)$, следовательно, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s \psi_1$ или $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \not\models_s (\psi_2 \vee \mathbf{E} \bigcirc \mathbf{E}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2)$.

При этом, если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \nvDash_s \psi_1$, то $(see^C(s), i_{pln}) \notin ES^1_0$ и, так как $ES_{0,t+1} \subseteq ES^1_0$, $(p, i_{pln}) \notin ES_{0,t+1}$, что противоречит исходному утверждению.

Если же выполнено $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \nvDash_s (\psi_2 \vee \mathbf{E} \bigcirc \mathbf{E}\psi_1 \mathcal{W}_{\leq t}\psi_2)$, то $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \nvDash_s \psi_2$ и $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \nvDash_s \mathbf{E} \bigcirc \mathbf{E}\psi_1 \mathcal{W}_{\leq t}\psi_2$.

Если $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \nvDash_s \mathbf{E} \bigcirc \mathbf{E}\psi_1 \mathcal{W}_{\leq t}\psi_2$, то для пары (s, i_{pln}) в множестве правил $\{pln_{ag,t+1}\}$ не задано ни одного правила или для любых набора правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag}\}$, действия системы $acs \in ACS$ такого, что для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$, и состояния внешней среды $s' \in S$ такого, что $(s, acs, s') \in env$, выполнено $(M, i^C_{\{pln_{ag}\}}), C, env, (s', i'_{pln}) \nvDash_s \mathbf{E}\psi_1 \mathcal{U}_{\leq t}\psi_2$.

Если для пары (s, i_{pln}) в множестве правил $\{pln_{ag,t+1}\}$ не задано ни одного правила, то $(see^C(s), i_{pln}) \notin D(\{pln_{ag,t+1}\})$, следовательно, $(p, i_{pln}) \notin ES_{0,t+1}$.

Если для любых набора правил $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \in \{pln_{ag}\}$, действия системы $acs \in ACS$ такого, что для всех агентов $ag \in C$ выполнено $acs[ag] = a_{ag}$, и состояния внешней среды $s' \in S$ такого, что $(s, acs, s') \in env$, выполнено $(M, i^C_{\{pln_{ag}\}}), C, env, (s', i'_{pln}) \nvDash_s \mathbf{E}\psi_1 \mathcal{U}_{\leq t}\psi_2$, то для любого возможного результата $(see^C(s'), i'_{pln}) \notin ES_{0,t}$, так как для состояний из множества $ES_{0,t}$ искомый путь уже найден. Следовательно, $\{(see^C(s), i_{pln}, i'_{pln}, a_{ag})\} \notin Pre_{\mathbf{E}}(ES_{0,t}, env, \{P^C \times C(\neg\psi_2) \times I_{pln} \times A_{ag}\})$, а значит $(see^C(s), i_{pln}) \in ES^1_0 \cap ES^2_0 \cap P^C \times C(\psi_2)$. Если $(see^C(s), i_{pln}) \in ES^1_0 \cap ES^2_0 \cap P^C \times C(\psi_2)$, то действия коалиции определяются дальше исключительно правилами $\{pln^2_{ag}\}$, следовательно, $(M, i^C_{\{pln_{ag,t+1}\}}), C, env, (s, i_{pln}) \vDash_s \psi_2$, что противоречит тому, что $(M, i^C_{\{pln_{ag,t+1}\}})$, $C, env, (s, i_{pln}) \nvDash_s \psi_2$.

Следовательно, для любого конечного числа итераций поиска неподвижной точки корректность алгоритма доказана. Так как все рассматриваемые множества конечны и множество ES_0 монотонно убывает, следовательно поиск внутренней

неподвижной точки завершается за конечное число итераций, что и требовалось доказать. ■

A.3.8. Алгоритмическая сложность

Утверждение 66 *Сложность алгоритма планирования не превосходит полинома от* $2^{|\phi|}$ *и* $|M|$*, где* $|\phi|$ *есть сумма размеров всех входных формул, а* $|M|$ *есть размер модели.*

Доказательство. Алгоритм включает в себя следующие этапы.

- Построение множеств S_0 и env_s. На этом этапе используются алгоритмы верификации и операции над множествами, полиномиальность и тех, и других, доказана.
- Построение множества I_{pln}. На этом этапе происходит спуск по структуре формулы, сложность которого не превосходит полинома от длинны формулы.
- Построение общего плана. Это этап является наиболее сложным и рассмотрен более детально далее.
- Выделение отдельных планов. Этот этап включает только операции над множествами, следовательно, его сложность ограничена полиномом.

Очевидно, что наиболее сложным этапом является этап построения общего плана. На этом этапе происходит рекурсивный спуск по структуре формулы $|\phi|$, при этом в некоторых узлах рекурсивные вызовы осуществляются в цикле. Так как количество итераций всех подобных циклов ограничено $|P^{AG}|\cdot|I_{pln}|^2\cdot|ACS|$, общее количество рекурсивных вызовов не превосходит $|\phi|\cdot|P^{AG}|\cdot|I_{pln}|^2\cdot|ACS|$. При этом на каждом шаге рекурсии могут происходить следующие вычисления.

- Обработка формул без темпоральных операторов. Сложность этого этапа, как показано в утверждении 39, не превосходит $O(|\phi|\cdot i_{\max}\cdot\log(|S|)^2\cdot$

$|S|^{3+2\cdot i_{max}}$). Следовательно, так как все сомножители не превосходят полинома размера модели $|M|$ и размера формулы $|\phi|$, сложность этого алгоритма не превосходит полинома от размера модели.[2]

- Обработка формул вида $\psi_1 \vee \psi_2$. На этом этапе используются только операции с множествами, полиномиальная сложность которых доказана.

- Обработка формул вида $\psi_1 \wedge \psi_2$. На этом этапе в цикле вычисляется несколько операций с множествами, а количество итераций цикла не превосходит $|P^{AG}|\cdot|I_{pln}|^2\cdot|ACS|$, так как множество $\{pln_{ag}\}$ монотонно уменьшается. Следовательно, сложность этого этапа ограничена полиномом.

- Обработка формул вида $\langle\langle D\rangle\rangle\psi_1$. На этом этапе используются только полиномиальные операции над множествами.

- Обработка формул $\mathbf{A}\bigcirc\psi_1$ или $\mathbf{E}\bigcirc\psi_1$. На этом этапе с помощью композиции $OBDD$ и операций над множествами вычисляется прообраз. Следовательно, сложность этого этапа ограничена полиномом от размера модели.

- Обработка формул вида $\neg\mathbf{E}\bigcirc true$. Сложность этого этапа является константной.

- Обработка формул вида $\mathbf{A}\,\psi_1\ \mathcal{U}_{\leq t}\psi_2$ или $\mathbf{A}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2$. На этих этапах вычислений используется двойной цикл. Количество итераций внешнего цикла не превосходит $|P^{AG}|\cdot|I_{pln}|^2\cdot|ACS|$, а внутреннего $|P^{AG}|\cdot|I_{pln}|$. На каждой итерации выполняются только операции над множествами, сложность которых полиномиальна. Следовательно, сложность этих этапов ограничена полиномом от размера модели.

[2]Заметим, что одним из элементов модели M является интерпретация предикатных и функциональных символов, размер которой ограничен $|S|^{i_{\max}}$.

- Обработка формул вида $\mathbf{E}\,\psi_1\ \mathcal{U}_{\leq t}\psi_2$ или $\mathbf{E}\,\psi_1\ \mathcal{W}_{\leq t}\psi_2$. На этих этапах вычислений используется один цикл, количество итераций которого не превосходит $|P^{AG}|\cdot|I_{pln}|$. На каждой итерации выполняются только операции над множествами, сложность которых полиномиальна. Следовательно, сложность этих этапов ограничена полиномом от размера модели.

Следовательно, общая сложность алгоритма не превосходит полинома от размера формулы $|\phi|$, размера модели $|M|$ и размера множества $|I_{pln}|$. При этом размер множества I_{pln} не превосходит $2^{|\phi|}$, следовательно, общая сложность алгоритма не превосходит полинома от $2^{|\phi|}$ и $|M|$, что и требовалось доказать. ■

Приложение Б

Реализация алгоритмов

Для демонстрации и проверки возможностей алгоритмов верификации и планирования реализован экспериментальный прототип, состоящий из следующих частей.

Cudd. Пакет разрешающих диаграмм университета Колорадо [166], адаптированный для использования в операционной системе Windows с помощью набора библиотек и компиляторов MinGW.

ObjectCudd. Объектно-ориентированный интерфейс доступа к функциональности пакета $CUDD$.

ModelCheking. Библиотека, содержащая реализацию алгоритмов верификации.

Planning. Библиотека, содержащая реализацию алгоритмов планирования.

XmlParser. Библиотека, используемая для чтения входных файлов.

MaslSpecification. XSD-схема, описывающая формат входных файлов для прототипа.

Входными данными для библиотек верификации и планирования является XML-файл специального формата, описанного схемой MaslSpecification.xsd. Эта схема также используется для автоматической генерации парсера и для проверки корректности входных файлов. Для чтения входных XML-файлов библиотеки планирования и верификации используют библиотеку XmlParser, часть которой сгенерирована автоматически на основе XSD-схемы. Для того, чтобы уменьшить влияние формата входных файлов на реализацию основных

алгоритмов, библиотека XmlParser экспортирует абстрактный интерфейс, через который она и используется основными библиотеками.

Основная реализация алгоритмов планирования и верификации выполнена на языке C++ с использованием компиляторов для Microsoft Windows, входящих в программную оболочку Microsoft Visual Studio (Academic Edition). В тоже время, анализ входных файлов и построение на их основе дерева объектов реализованы на языке C# с использованием платформы .NET Framework 2.0. Подобное гибридное решение позволяет удачно сочетать быстродействие алгоритмов, реализованных на C++, с удобством чтения и проверки XML, предоставленными платформой .NET Framework.

Б.1. Формат входных файлов

Корневым элементом входных XML-файлов является XML-элемент `Specification` с единственным обязательным атрибутом `Name`. Внутри корневого элемента находятся два обязательных вложенных элемента: `Model` и `Properties`. Элемент `Model` содержит описание *модели* системы для верификации или построения плана, а элемент `Properties` содержит описание свойств, которые нужно проверить или для реализации которых нужно построить план.

Описание модели состоит из следующих основных частей: описания типов `Types`, описания переменных состояний внешней среды `StateVariables`, описания агентов `Agents` и описания поведения внешней среды `Environment`. Типы могут быть двух видов: целочисленные типы и типы-перечисления. Для целочисленных типов необходимо указать количество бит, используемых для представления значений этого типа, а для типов-перечислений — список допустимых значений. При объявлении любой переменной необходимо указать тип, к которому она принадлежит.

Описание каждого агента `Agent` из списка `Agents` состоит из следующих

частей: описание переменных действия `ActionVariables`, описания представлений агента `Beliefs` и описания его текущего плана `Plan`. Синтаксис описания переменных действия агента аналогичен синтаксису описания переменных состояния среды, а синтаксис описания представлений агента аналогичен синтаксису описания поведения внешней среды. В алгоритмах планирования элементы `Beliefs` и `Plan` игнорируются.

Описание плана `Plan` состоит из двух основных частей: описания переменных состояния плана `StateVariables` и множества правил переходов плана `Rules`. Переменные состояния плана агента описываются аналогично переменным состояния внешней среды. Описание правила плана `Rule` состоит из четырех частей.

Предусловие на состояние среды `EnvironmentState`. Выражение, описывающее множество состояний внешней среды, в которых применимо данное правило.

Предусловие на состояние плана `PlanState`. Выражение, описывающее множество состояний плана, в которых применимо данное правило.

Условие на действие `Action`. Выражение, описывающее множество действий, которые можно выполнять согласно данному правилу.

Условие на следующее состояние плана `PlanNextState`. Описание множества возможных следующих состояний плана, согласно данному правилу.

Каждое выражения `EnvironmentState` и `Action` синтаксически подобны описанию свойств системы `Properties`, тогда как в выражения `PlanState` и `PlanNextState` представляют собой набор элементов `Include`, каждый из которых описывает то, с каким значением определенная переменная может входить (или, наоборот, не входить) в состояние плана по данному правилу.

Описание поведения внешней среды `Environment` состоит из двух частей: описания инвариантных свойств `InvariantProperties` и описания правил переходов `Rules`. Инвариантные свойства описываются выражением, которое должно быть выполнено в любом допустимом состоянии внешней среды. Синтаксически эти выражения аналогичны описанию свойств `Properties`. Каждое правило описания поведения внешней среды состоит из трех частей.

Предусловие `Precondition`**.** Выражение, описывающее множество состояний внешней среды, в которых данный переход возможен.

Действие `Action`**.** Выражение, описывающее множество действий, которые могут инициировать данный переход.

Результат `Postcondition`**.** Выражение, описывающее множество возможных следующих состояний внешней среды, согласно данному переходу.

Все выражения в описании правил поведения внешней среды синтаксически аналогичны описанию свойств `Properties`.

Блок описания свойств системы `Properties` состоит из набора элементов `Formula`, рекурсивно вложенных друг в друга. Формулы бывают следующих типов.

Логические операции. Формулы данного типа содержат список вложенных формул (для алгоритмов планирования из не более чем двух элементов) и атрибут `Operation`, указывающий на конкретную логическую операцию (`Not`, `And` или `Or`).

Логические константы. Формулы данного типа используются для обозначения логических константы $true$ и $false$.

Предикаты. Формулы данного типа содержат два терма-аргумента `Left` и `Right` и атрибут `Predicate`, указывающий на конкретный предикат

(`Equals`, `NotEquals`, `Less`, `NotLess`, `Greater`, `NotGreater`). При этом термы могут быть трех типов: константы, предметные переменные и двухмерные функциональные символы `Plus` и `Minus`.

Предикат неравенства нулю. Единственный одноместный предикат, применимый только к предметным переменный. Используется для проверки на неравенство значения переменной нулю.

Оператор результата действия. Формулы данного типа состоят из двух частей `Action` и `Result` и соответствуют оператору $[\alpha]\phi$.

Темпоральное свойство. Формулы этого типа соответствуют квантору пути `All` или `Some`. При этом вложенной формулой для темпорального свойства может быть одна из двух формул пути: "в следующий момент" или "когда-нибудь". Формулы первого типа содержат список свойств, которые должны быть выполнены в следующий момент, а формулы второго типа состоят из двух частей: настоящего `Present` и будущего `Future`. Кроме того, для формул пути второго типа можно указать флаг "слабины" и ограничение на максимальное количество шагов.

Квантор коалиции. Формулы этого типа состоят из двух частей: списка имен агентов `Agents` и списка вложенных свойств `Formulas`.

Оператор представлений. Формулы этого типа используются для формулирования свойств, связанных с представлениями агентов[1].

Б.2. Алгоритмы верификации

Точкой входа в библиотеку верификации является статический класс фабрики `Factory`, используемый для создания других объектов библиотеки на основе

[1]Свойства, связанные с желаниями и намерениями в экспериментальной реализации не поддерживаются.

соответствующих XML-описаний. Помимо фабрики, библиотека верификации экспортирует три основных класса: класс модели `Model`, класс верификатора `ModelChecker` и абстрактный класс формулы `Formula`.

Класс модели `Model` содержит всю необходимую информацию о текущей модели, включая следующие части.

- Список всех известных типов переменных, представленных объектами класса `TypeDescriptor`. Каждый такой объект ответственен за инициализацию переменных разрешающих диаграмм и за построение символического представления предикатных и функциональных символов.

- Список всех известных переменных, представленных объектами класса `Variable`. Каждый такой объект отвечает за хранение информации о соответствующих переменных разрешающих диаграмм и за построение проекций.

- Список всех известных агентов, представленных объектами класса `Agent`. Каждый такой объект хранит всю информацию об объектах данного типа.

- Описание внешней среды, представленное объектом класса `Environment`. Этот объект отвечает за построение символического представления отношения переходов внешней среды.

В свою очередь, каждый объект класса `Agent` содержит информацию о переменных действия для данного агента, представления агента, заданные объектом класса `Environment`, и план агента, заданный объектом класса `Plan`. Объект класса `Plan` ответственен за построение символического представления плана агента, а также за вычисление различных вариантов прообраза.

Класс `ModelChecker` является основным классом библиотеки и отвечает за верификацию формул, относительно модели. При этом для проведения верификации он использует дерево объектов классов-наследников абстрактного класса

`Formula`, реализующих алгоритм верификации для формул определенного типа.

Результатом работы алгоритмов верификации является множество состояний внешней среды и состояний планов агентов, в которых заданное свойство выполнено. Это множество описывается с помощью двоичной функции относительно битов в представлении переменных состояния.

Б.3. Алгоритмы планирования

Точкой входа в библиотеку планирования является класс фабрики `Factory`, позволяющий создавать объекты класса `Planner` на основе описания задачи из XML-файла. Этот класс, используя класс `Model` из библиотеки верификации и абстрактный класс `PlanBuilder`, осуществляет построение плана, соответствующего требуемым свойствам.

Для построения плана сначала строится дерево объектов классов-наследников абстрактного класса `PlanBuilder`, каждый из которых ответственен за построение планов для определенных подформул исходной формулы. На каждом этапе построения плана результат автоматически проверяется на корректность с использованием класса `Formula` из библиотеки верификации.

Результатом работы класса `Planner` являются множество состояний внешней среды и плана, а также отношение переходов плана, для которого выполнено искомое свойство. Множество состояний и отношение переходов представлены как двоичная функция относительно битов в представлении переменных состояния и действия.

Б.4. Тестовые приложения

Для проверки работоспособности реализованных алгоритмов разработаны два тестовых приложения: TestModelChecking и TestPlanning. Оба приложения являются консольными приложениями и используют классы соответствующих библиотек для решения задачи верификации или планирования, описанной в XML-файле, путь к которому задан как параметр командной строки.

Кроме того, разработан ряд тестовых файлов, включающих две модели и различные варианты свойств. Внешняя среда для первой модели описывается тремя двоичными флагами и контролируется двумя агентами, первый из которых может переключать значения первых двух переменных, а второй — включать или выключать значение третьей переменной. На этой модели тестируются большинство свойств, включая следующие свойства:

- Верификация и планирование для простых логических формул.
- Верификация и планирование для квантора коалиций.
- Верификация и планирование для темпоральных свойств, выполненных на любом пути.
- Верификация и планирование для свойств безопасности.
- Верификация и планирование для свойств живучести.
- Верификация и планирование для различных комбинаций темпоральных свойств.
- Верификация для оператора $\mathsf{Bel}\ \phi$.

Вторая модель представляет внешнюю среду, не контролируемую агентом целиком. Состояние среды описывается двумя двоичными флагами, при этом

единственный представленный агент может попытаться включить или выключить значение первого флага, но результат его действий зависит от текущего состояния первого флага. Вторая модель используется для тестирования верификации и планирования для темпоральных свойств, выполненных хотя бы на одном пути.

Литература

[1] *Бугайченко, Д. Ю.* Математическая модель интеллектуального агента. // Сборник трудов международной конференции «Процессы управления и устойчивость». — Санкт-Петербург: Издательство СПбГУ, 2006. — С. 9–19.

[2] *Бугайченко, Д. Ю.* Математическая модель и спецификация интеллектуальных агентных систем. / Д. Ю. Бугайченко // *Системное программирование.* — 2006. — № 2. — С. 94–115.

[3] *Бугайченко, Д. Ю.* Верификация распределенных систем реального времени по спецификации $MASL$ / Д. Ю. Бугайченко // *Вестник СПбГУ, Серия 1.* — 2007. — № 3. — С. 65–74.

[4] *Бугайченко, Д. Ю.* Символическое планирование в ограничениях CTL. // Сборник трудов международной конференции «Процессы управления и устойчивость». — Санкт-Петербург: Издательство СПбГУ, 2007. — С. 335–341.

[5] *Бугайченко, Д. Ю.* Разработка методики проектирования интеллектуальных систем реального времени с использованием интеллектуальных агентов. // Сборник трудов конференции «Технологии Microsoft в теории и практике». — Санкт-Петербург: Издательство СПбГПУ, 2004. — С. 77–78.

[6] *Бугайченко, Д. Ю.* Абстрактная архитектура интеллектуального агента и методы ее реализации. / Д. Ю. Бугайченко, И. П. Соловьев // *Системное программирование.* — 2005. — № 1. — С. 36–67.

[7] *Бугайченко, Д. Ю.* Архитектура изолированного интеллектуального агента. // Сборник трудов международной конференции «Современные проблемы информатизации». — Воронеж: Издательство «Научная книга», 2006. — С. 220–222.

[8] *Бугайченко, Д. Ю.* Методы решения некоторых инфраструктурных задач, возникающих при разработке мультиагентных систем. // Сборник трудов конференции «Технологии Microsoft в теории и практике». — Санкт-Петербург: Издательство СПбГПУ, 2006. — С. 177–178.

[9] *Бугайченко, Д. Ю.* Формально-логическая спецификация мультиагентных систем реального времени. / Д. Ю. Бугайченко, И. П. Соловьев // *Вестник СПбГУ, Серия 1.* — 2007. — № 3. — С. 49–57.

[10] *Гаранина, Н. О.* Верификация распределенных систем с использованием аффинного представления данных, логик знаний и действий: Дис. . . канд. физ-мат. наук: 28.12.04 / Российская академия наук, Сибирское отделение, Институт систем информатики им. А.П.Ершова. — Новосибирск, 2004.

[11] *Марков, А. А.* Элементы математической логики / А. А. Марков. — Москва: Издательство МГУ, 1984.

[12] *Мендельсон, Э.* Введение в математическую логику / Э. Мендельсон. — Москва: Наука, 1984.

[13] Agent programming in 3APL / K. V. Hindriks, F. S. D. Boer, W. V. der Hoek, J.-J. C. Meyer // *Autonomous Agents and Multi-Agent Systems.* — 1999. — Vol. 2, no. 4. — Pp. 357–401.

[14] *Agha, G.* Actors: a model of concurrent computation in distributed systems / G. Agha. — Cambridge, MA, USA: MIT Press, 1986. — 190 pp.

[15] *Agotnes, T.* On the logic of coalitional games // Proc. Fifth International Joint Conference on Autonomous Agents and Multiagent Systems / Ed. by P. Stone, G. Weiss. — New York (USA): ACM Press, 2006. — P. 153–160.

[16] *Agotnes, T.* Temporal qualitative coalitional games // Proc. Fifth International Joint Conference on Autonomous Agents and Multiagent Systems / Ed. by P. Stone, G. Weiss. — New York (USA): ACM Press, 2006. — Pp. 177–184.

[17] *Akers, S.* Binary decision diagrams / S. Akers // *IEEE Transactions on Computers.* — 1978. — June. — Vol. C-27, no. 6. — Pp. 509–516.

[18] Algebraic Decision Diagrams and their applications // Proc. IEEE/ACM International Conference on CAD. — Santa Clara (California): IEEE Computer Society Press, 1993. — Pp. 188–191.

[19] *Alur, R.* Timed automata // Proc. 11th International Conference on Computer Aided Verification. — London (UK): Springer-Verlag, 1999. — Pp. 8–22.

[20] *Alur, R.* A theory of timed automata / R. Alur, D. L. Dill // *Theoretical Computer Science.* — 1994. — Vol. 126, no. 2. — Pp. 183–235.

[21] *Alur, R.* The benefits of relaxing punctuality / R. Alur, T. Feder, T. A. Henzinger // *J. ACM.* — 1996. — Vol. 43, no. 1. — Pp. 116–146.

[22] *Alur, R.* Logics and models of real time: A survey // Proc. Real-Time: Theory in Practice, REX Workshop. — London (UK): Springer-Verlag, 1992. — Pp. 74–106.

[23] *Alur, R.* Real-time logics: complexity and expressiveness / R. Alur, T. A. Henzinger // *Inf. Comput.* — 1993. — Vol. 104, no. 1. — Pp. 35–77.

[24] *Alur, R.* A really temporal logic / R. Alur, T. A. Henzinger // *J. ACM.* — 1994. — Vol. 41, no. 1. — Pp. 181–203.

[25] *Alur, R.* Alternating-time temporal logic / R. Alur, T. A. Henzinger, O. Kupferman // *J. ACM.* — 2002. — Vol. 49, no. 5. — Pp. 672–713.

[26] *Alur, R.* Reactive modules / R. Alur, T. Henzinger // *Formal Methods in System Design.* — 1999. — July. — Vol. 15, no. 11. — Pp. 7–48.

[27] *Andreka, H.* Handbook of Philosophical Logic / H. Andreka, I. Nemeti, I. Sain // Handbook of Philosophical Logic / Ed. by D. Gabbay. — Kluwer, 1997.

[28] *Antoniotti, M.* Synthesis and verification of discrete controllers for robotics and manufacturing devices with temporal logic and the Control-D system: Ph.D. thesis / New York University. — New York, NY, USA: New York University, 1995.

[29] *Austin, J. L.* How to do things with words / J. L. Austin. — Oxford, England: Oxford University Press, 1962.

[30] *Aziz, A.* BDD variable ordering for interacting finite state machines // Proc. 31st annual conference on Design automation. — San Diego (USA): ACM Press, 1994. — Pp. 283–288.

[31] *Bacchus, F.* Planning for temporally extended goals / F. Bacchus, F. Kabanza // *Annals of Mathematics and Artificial Intelligence.* — 1998. — Vol. 22, no. 1-2. — Pp. 5–27.

[32] *Baeten, J. C. M.* A brief history of process algebra / J. C. M. Baeten // *Theor. Comput. Sci.* — 2005. — Vol. 335, no. 2-3. — Pp. 131–146.

[33] *Baier, C.* Approximate symbolic model checking of continuous-time markov chains // Proc. 10th International Conference on Concurrency Theory. — London (UK): Springer-Verlag, 1999. — Pp. 146–161.

[34] *Biere, A.* ABCD: a compact BDD library (http://www.inf.ethz.ch/personal/biere/projects/abcd/).

[35] *Bonet, B.* Heuristic search planner 2.0 / B. Bonet, H. Geffner // *The AI Magazine.* — 2001. — Vol. 22, no. 1. — Pp. 77–80.

[36] *Bonet, B.* Planning as heuristic search / B. Bonet, H. Geffner // *Artificial Intelligence.* — 2001. — Vol. 129, no. 1-2. — Pp. 5–33.

[37] *Borgo, S.* Coalitions in action logic // Proc. Twentieth International Joint Conference on Artificial Intelligence. — Hyderabad (India): 2007. — January. — Pp. 1822–1827.

[38] *Brand, D.* On communicating finite-state machines / D. Brand, P. Zafiropulo // *J. ACM.* — 1983. — Vol. 30, no. 2. — Pp. 323–342.

[39] *Bratman, M. E.* Intention, plans, and practical reason / M. E. Bratman. — Cambridge, MA: Harvard University Press, 1987.

[40] *Bratman, M. E.* Plans and resource-bounded practical reasoning / M. E. Bratman, D. Israel, M. Pollack // Philosophy and AI: Essays at the Interface / Ed. by R. Cummins, J. L. Pollock. — Cambridge, Massachusetts: The MIT Press, 1991. — Pp. 1–22.

[41] *Bryant, R. E.* Graph-based algorithms for boolean function manipulation / R. E. Bryant // *IEEE Transactions on Computers.* — 1986. — Vol. C-35, no. 8. — Pp. 677–691.

[42] *Bryant, R. E.* Symbolic boolean manipulation with ordered binary-decision diagrams / R. E. Bryant // *ACM Computing Surveys.* — 1992. — Vol. 24, no. 3. — Pp. 293–318.

[43] *Bugaychenko, D. Y. MASL*: A logic for the specification of multiagent real-time systems. // Proc. 5^{th} International Central and Eastern European Conference on Multi-Agent Systems. — Leipzig (Germany): Springer-Verlag, 2007. — Pp. 183–192.

[44] *Castro, J.* Towards requirements-driven information systems engineering: the Tropos project / J. Castro, M. Kolp, J. Mylopoulos // *Information Systems.* — 2002. — Vol. 27, no. 6. — Pp. 365–389.

[45] *Chellas, B. F.* Modal logic: an introduction / B. F. Chellas. — Cambridge University Press, 1980. — Vol. 316.

[46] *Chen, Y. X.* Temporal planning using subgoal partitioning and resolution in SGPlan / Y. X. Chen, B. W. Wah, C. W. Hsu // *Journal of Artificial Intelligence Research.* — 2006.

[47] *Cimatti, A.* Automatic OBDD-based generation of universal plans in non-deterministic domains // AAAI/IAAI. — 1998. — Pp. 875–881.

[48] *Clarke, E.* Model checking / E. Clarke. — MIT Press, 1999. — P. 330.

[49] *Cohen, P. R.* Intention is choice with commitment / P. R. Cohen, H. J. Levesque // *Artificial Intelligence.* — 1990. — Vol. 42, no. 2-3. — Pp. 213–261.

[50] Combining test case generation and runtime verification / C. Arthoa, H. Barringerb, A. Goldbergc et al. // *Theoretical Computer Science.* — 2005. — Vol. 336, no. 2-3. — Pp. 209–234.

[51] *Dean, T.* A model for reasoning about persistence and causation / T. Dean, K. Kanazawa // *Computational Intelligence.* — 1989. — Vol. 5, no. 3. — Pp. 142–150.

[52] DESIRE: Modelling multi-agent systems in a compositional formal framework / F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, J. Treur // *Int Journal of Cooperative Information Systems.* — 1997. — Vol. 6, no. 1. — Pp. 67–94.

[53] *Drusinky, D.* Real-time, on-line, low impact, temporal pattern matching // Proc World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003). — 2003.

[54] *Drusinky, D.* Verification of timing properties in rapid system prototyping // Proc. 14th IEEE International Workshop on Rapid System Prototyping (RSP'03). — Washington DC (USA): IEEE Computer Society, 2003. — P. 47.

[55] *Duff, I.* Direct methods for sparse matrices / I. Duff, A. Erisman, J. Reid // *ACM Transactions on Methematical Software.* — 1986. — Vol. 15. — Pp. 1–14.

[56] *Emerson, E. A.* Temporal and modal logic. / E. A. Emerson // Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B) / Ed. by J. van Leeuwen. — North-Holland Pub. Co., 1990. — Pp. 995–1072.

[57] Estelle: Formal protocol specification and testing, `http://www.eecis.udel.edu/~amer/PEL/estelle/`.

[58] *FIPA*. Fipa agent communication specifications (`http://www.fipa.org/repository/aclspecs.html`).

[59] Foundation for Intelligent Physical Agent (`http://www.fipa.org`).

[60] *Fujita, M.* Multi-terminal binary decision diagrams: An efficient datastructure for matrix representation / M. Fujita, P. C. McGeer, J. C.-Y. Yang // *Form. Methods Syst. Des.* — 1997. — Vol. 10, no. 2-3. — Pp. 149–169.

[61] *Gasser, L.* MACE: A flexible testbed for distributed AI research. / L. Gasser, C. Braganza, N. Hermann. // *Distributed Artificial Intelligence.* — 1987. — Pp. 119–152.

[62] *Gill, A.* Introduction to the theory of finite-state machines / A. Gill. — McGraw-Hill, 1962.

[63] *Goldblatt, R.* Logics of time and computation / R. Goldblatt. — Stanford, CA, USA: Center for the Study of Language and Information, 1987.

[64] *Goranko, V.* Coalition games and alternating temporal logics // Proc. Eighth Conference on Theoretical Aspects of Rationality and Knowledge (TARKVI-II) / Ed. by J. van Benthem. — 2001. — P. 259–272.

[65] *Guestrin, C.* Planning under uncertainty in complex structured environments: Ph.D. thesis / Computer Science Department, Stanford University. — 2003.

[66] *Haigh, K. Z.* High-level planning and low-level execution: towards a complete robotic agent // Proc. first international conference on Autonomous agents. — Marina del Rey (USA): ACM Press, 1997. — Pp. 363–370.

[67] *Halpern, J. Y.* A guide to completeness and complexity for modal logics of knowledge and belief / J. Y. Halpern, Y. Moses // *Artificial Intelligence.* — 1992. — Vol. 54, no. 3. — Pp. 319–379.

[68] *Harel, D.* Dynamic logic / D. Harel, D. Kozen, J. Tiuryn // *SIGACT News.* — 2001. — Vol. 32, no. 1. — Pp. 66–69.

[69] The Hearsay-I speech understanding system: An example of the recognition process / D. Reddy, L. Erman, R. Fennell, R. Neely // *Transactions on Computers.* — 1976. — April. — no. 4. — Pp. 422–431.

[70] *Hennessy, M.* A process algebra for timed systems / M. Hennessy, T. Regan // *Inf. Comput.* — 1995. — Vol. 117, no. 2. — Pp. 221–239.

[71] *Henzinger, T. A.* It's about time: Real-time logics reviewed // Proc. 9th International Conference on Concurrency Theory. — London (UK): Springer-Verlag, 1998. — Pp. 439–454.

[72] *Henzinger, T. A.* HYTECH: A model checker for hybrid systems // Proc. 9th International Conference on Computer Aided Verification. — London (UK): Springer-Verlag, 1997. — Pp. 460–463.

[73] *Henzinger, T. A.* Timed alternating-time temporal logic // Proc. 4th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 06) / Ed. by E. Asarin, P. Bouyer. — Vol. 4202. — Paris (France): Springer, 2006. — September. — Pp. 1–17.

[74] *Hett, A.* MORE: an alternative implementation of BDD packages by multi--operand synthesis // Proc. conference on European design automation. — Geneva (Switzerland): IEEE Computer Society Press, 1996. — Pp. 164–169.

[75] *Hewitt, C.* A universal modular ACTOR formalism for AI // Proc. Third International Joint Conference on Artificial Intelligence (IJCAI-73). — 1973. — Pp. 235–245.

[76] *Hintikka, J.* Impossible possible worlds vindicated / J. Hintikka // *Journal of Philosophical Logic.* — 1975. — August. — Vol. 4, no. 3. — Pp. 475–484.

[77] *Hoare, C. A. R.* An axiomatic basis for computer programming / C. A. R. Hoare // *Communications of the ACM.* — 1969. — Vol. 12, no. 10. — Pp. 576–580.

[78] *Hoare, C. A. R.* Communicating sequential processes / C. A. R. Hoare // *Commun. ACM.* — 1978. — Vol. 21, no. 8. — Pp. 666–677.

[79] *Hoffmann, J.* The FF planning system: Fast plan generation through heuristic search / J. Hoffmann, B. Nebel // *Journal of Artificial Intelligence Research.* — 2001. — Vol. 14. — Pp. 253–302.

[80] *Holzmann, G.* The SPIN model checker / G. Holzmann // *IEEE Transactions on Software Engineering.* — 1997. — May. — Vol. 23, no. 5. — Pp. 279–295.

[81] *Hsiung, P.-A.* A state graph manipulator tool for real-time system specification and verification // Proc. 5th International Conference on Real-Time Computing Systems and Applications. — Washington DC (USA): IEEE Computer Society, 1998. — P. 181.

[82] *Huhns, M. N.* Multiagent Systems / M. N. Huhns, L. M. Stephen // Multiagent Systems / Ed. by G. Weiss. — MIT Press, 2001. — Pp. 79–121.

[83] *ichi Minato, S.* Shared binary decision diagram with attributed edges for efficient boolean function manipulation // Proc. 27th ACM/IEEE conference on Design automation. — Orlando (USA): ACM Press, 1990. — Pp. 52–57.

[84] Implicit state enumeration of finite state machines using BDD's // Proc. IEEE International Conference on Computer-Aided Design. — Santa Clara (USA): 1990. — November. — Pp. 130–133.

[85] International conference on automated planning and scheduling (`http://www.icaps-conference.org/`) // International Conference on Automated Planning and Scheduling. — 2007.

[86] *Ishiura, N.* Minimization of binary decision diagrams based on exchanges of variables // Proc. IEEE International Conference on Computer-Aided Design. — Santa Clara (USA): 1991. — November. — Pp. 472–475.

[87] *ITU-T.* Specification and description language (SDL): Tech. rep. / ITU-T: International Telecommunication Union ,Standardization Sector, 1999.

[88] *Jamroga, W.* Do agents make model checking explode (computationally)? / W. Jamroga, J. Dix // *Multi-Agent Systemsand Applications.* — 2005. — no. IV.

[89] *Janssen, G.* Design of a pointerless BDD package // Proc. IEEE 10th International Workshop on Logic & Synthesis. — Tahoe City (USA): 2001. — June.

[90] *Janssen, G.* A consumer report on BDD packages // Proc. 16th symposium on Integrated circuits and systems design. — Washington DC (USA): IEEE Computer Society, 2003. — P. 217.

[91] *Jenings, N. R.* Applications of intelligent agents. — 2000.

[92] *Jenings, N. R.* Multiagent Systems / N. R. Jenings, M. J. Wooldridge // Multiagent Systems / Ed. by G. Weiss. — MIT Press, 2001. — Pp. 377–421.

[93] *Jennings, N. R.* On being responsible // Proc. Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW 91) / Ed. by E. Werner, Y. Demazeau. — Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992. — Pp. 93–102.

[94] *Jensen, R. M.* A comparison study between the CUDD and BuDDy OBDD package applied to AI-planning problems: Tech. rep. / R. M. Jensen: Computer Science Department, Carnegie Mellon University, 2002. — CMU-CS-02-173.

[95] *Jensen, R. M.* Efficient BDD-based planning for non-deterministic, fault-tolerant, and adversarial domains: Ph.D. thesis / Carnegie Mellon University. — 2003. — June.

[96] *Jensen, R. M.* OBDD-based universal planning: Specifying and solving planning problems for synchronized agents in non-deterministic domains / R. M. Jensen, M. M. Veloso // *Lecture Notes in Computer Science.* — 1999. — Vol. 1600. — Pp. 213–223.

[97] *Jensen, R. M.* ASET: a multi-agent planning language with non-deterministic durative tasks for BDD-based fault tolerant planning // Proc. 15th International Conference on Automated Planning and Scheduling (ICAPS-05). — 2005.

[98] *Jensen, R. M.* Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning // Proc. 14th International Conference on Automated Planning and Scheduling ICAPS-04. — 2004.

[99] *Jensen, R.* OBDD-based deterministic planning using the UMOP planning framework // Proc. Workshop on Model-Theoretic Approaches to Planning (AIPS 00). — 2000. — Pp. 26–31.

[100] *Jiang, S.* Supervisory control of discrete event systems with CTL* temporal logic specifications / S. Jiang, R. Kumar // *SIAM J. Control Optim.* — 2006. — Vol. 44, no. 6. — Pp. 2079–2103.

[101] *Josephs, M. B.* Receptive process theory / M. B. Josephs // *Acta Inf.* — 1992. — Vol. 29, no. 1. — Pp. 17–31.

[102] *Kabanza, F.* Search control in planning for temporally extended goals // Proc. 15th International Conference on Automated Planning and Scheduling (ICAPS-05). — 2005. — Pp. 130–139.

[103] *Konolige, K.* A deduction model of belief / K. Konolige. — London UK, San Mateo, CA: Pitman Publishing, Morgan Kaufmann, 1986.

[104] *Korf, R. E.* Real-time heuristic search / R. E. Korf // *Artif. Intell.* — 1990. — Vol. 42, no. 2-3. — Pp. 189–211.

[105] *Kozen, D.* Handbook of Theoretical Computer Science / D. Kozen, J. Tiuryn // Handbook of Theoretical Computer Science / Ed. by J. van Leeuwen. — Amsterdam: North-Holland, 1990. — Vol. B. — Pp. 789–840.

[106] *Kripke, S.* Semantical analysis of modal logic I: Normal modal propositional calculi / S. Kripke // *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik.* — 1963. — no. 9. — P. 67–96.

[107] *Kwiatkowska, M.* Probabilistic symbolic model checking with PRISM: a hybrid approach / M. Kwiatkowska, G. Norman, D. Parker // *Int. J. Softw. Tools Technol. Transf.* — 2004. — Vol. 6, no. 2. — Pp. 128–142.

[108] *Laroussinie, F.* Model checking timed ATL for durational concurrent game structures // Proc. 4th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 06) / Ed. by E. Asarin, P. Bouyer. — Vol. 4202. — Paris (France): Springer, 2006. — September. — Pp. 245–259.

[109] *Laroussinie, F.* On the expressivity and complexity of quantitative branching-time temporal logics / F. Laroussinie, P. Schnoebelen, M. Turuani // *Theor. Comput. Sci.* — 2003. — Vol. 297, no. 1-3. — Pp. 297–315.

[110] *Lee, C.* Representation of switching circuits by binary-decision programs / C. Lee // *Bell System Technical Journal.* — 1959. — July. — Vol. 38. — Pp. 985–999.

[111] *Lenat, D.* BEINGs: knowledge as interacting experts // Proc. International Joint Conference on Artificial Intelligence. — 1975. — Pp. 126–133.

[112] *Lenat, D.* AM: an artificial intelligence approach to discovery in mathematics as heuristic search.: Ph.D. thesis / Stanford University. — 1976.

[113] *Levesque, H. J.* A logic of implicit and explicit belief // Proc. Fourth National Conference on Artificial Intelligence. — Austin, TX: 1984. — P. 198–202.

[114] *Levesque, H. J.* On acting together // Proc. Eighth National Conference on Artificial Intelligence. — Boston, MA: 1990. — Pp. 94–99.

[115] *Lhoták, O.* Jedd: a BDD-based relational extension of Java / O. Lhoták, L. Hendren // *SIGPLAN Not.* — 2004. — Vol. 39, no. 6. — Pp. 158–169.

[116] *Lind-Nielsen, J.* BuDDy — a Binary Decision Diagram package (http://sourceforge.net/projects/buddy).

[117] *Ljunberg, M.* The OASIS air trafic management system // Proc. Second Pacific Rim International Conference on AI. — 1992.

[118] *Majercik, S. M.* MAXPLAN: A new approach to probabilistic planning // Artificial Intelligence Planning Systems. — 1998. — Pp. 86–93.

[119] *Mayfield, J.* Evaluating KQML as an agent communication language / J. Mayfield, Y. Labrou, T. Finin // *Intelligent Agents II.* — 1996. — Vol. 1037. — P. 347–360.

[120] MBP: a model based planner // Proc. Workshop on Planning under Uncertainty and Incomplete Information (IJCAI 01). — 2001.

[121] *McDermott, D.* Using regression-match graphs to control search in planning / D. McDermott // *Artif. Intell.* — 1999. — Vol. 109, no. 1-2. — Pp. 111–159.

[122] *McMillan, K. L.* Symbolic model checking / K. L. McMillan. — Dordrecht,The Netherlands: Kluwer Academic Publishers, 1993.

[123] *Meyer, J.-J. C.* A logical approach to the dynamics of commitments / J.-J. C. Meyer, W. van der Hoek, B. van Linder // *Artif. Intell.* — 1999. — Vol. 113, no. 1-2. — Pp. 1–40.

[124] *Milner, R.* A calculus of communicating systems / R. Milner. — Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1982.

[125] *Milvang-Jensen, K.* BDDNOW: A parallel BDD package // Proc. Second International Conference on Formal Methods in Computer-Aided Design. — London (UK): Springer-Verlag, 1998. — Pp. 501–507.

[126] *Miraftabi, R.* Agents on the loose: An overview of agent technologies. — University of Joensuu. — 2000.

[127] MOCHA: Modularity in model checking / R. Alur, T. Henzinger, F. Mang et al. // *Lecture Notes in Computer Sciense.* — 1998. — Vol. 1427. — P. 521–525.

[128] Model checking for multiagent systems: The MABLE language and its applications / M. J. Wooldridge, M.-P. Huget, M. Fisher, S. Parsons // *International Journal on Artificial Intelligence Tools.* — 2006. — Vol. 15, no. 2. — Pp. 195–225.

[129] Model checking multi-agent systems with MABLE // Proc. first international joint conference on Autonomous agents and multiagent systems. — Bologna (Italy): ACM Press, 2002. — Pp. 952–959.

[130] *Moore, R. C.* Reasoning about knowledge and action: Tech. rep. / R. C. Moore. — Menlo Park, CA (USA): SRI International, Artificial Intelligence Center, 1980. — October.

[131] *M.Ryan.* Agents and roles: refinement in alternating-time temporal logic // Proc. Eighth International Workshop on Agent Theories, Architectures, and Languages / Ed. by J.-J. C. Meyer, M.Tambe. — Vol. 2333. — 2001. — P. 100–114.

[132] NUSMV: A new symbolic model verifier // Proc. 11th International Conference on Computer Aided Verification. — London (UK): Springer-Verlag, 1999. — Pp. 495–499.

[133] *Omicini, A.* SODA: societies and infrastructures in the analysis and design of agent-based systems // Proc. First international workshop on Agent-orient-

ed software engineering. — Secaucus (USA): Springer-Verlag New York, Inc., 2001. — Pp. 185–193.

[134] On the use of MTBDDs for performability analysis and verification of stochastic systems / H. Hermannsa, M. Kwiatkowskab, G. Normanb et al. // *Journal of Logic and Algebraic Programming.* — 2003. — May-August. — Vol. 56, no. 1-2. — Pp. 23–67.

[135] *Parker, D.* Implementation of symbolic model checking for probabilistic system: Ph.D. thesis / University of Birmingham. — 2002.

[136] *Parunak, H. V. D.* Foundations of Distributed Artificial Intelligence / H. V. D. Parunak // Foundations of Distributed Artificial Intelligence. — Wiley Inter-Science, 1994.

[137] *Parunak, H. V. D.* Multiagent Systems / H. V. D. Parunak // Multiagent Systems / Ed. by G. Weiss. — MIT Press, 2001. — Pp. 377–421.

[138] *Pauly, M.* Logical for social software: Ph.D. thesis / University of Amsterdam. — 2001.

[139] *Pauly, M.* A modal logic for coalitional power in games / M. Pauly // *Journal of Logic for Computation.* — 2002. — no. 12. — Pp. 146–166.

[140] *Pauly, M.* Logic for mechanism design — a manifesto // Proc. Workshop on Game Theory and Decision Theory in Agent Systems. — 2003.

[141] *Pearl, J.* Heuristics: intelligent search strategies for computer problem solving / J. Pearl. — Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1984.

[142] *Perlis, D.* Languages with self-reference I: foundations (or: we can have every-

thing in first-order logic) / D. Perlis // *Artificial Intelligence.* — 1985. — Vol. 25, no. 3. — Pp. 301–322.

[143] *Perlis, D.* Languages with self-reference II: knowledge, belief and modality / D. Perlis // *Artificial Intelligence.* — 1988. — Vol. 34, no. 2. — Pp. 179–212.

[144] *Perlis, D.* Meta in logic / D. Perlis // *Meta-Level Architectures and Reflection.* — 1988. — Pp. 37–49.

[145] *Pistore, M.* Symbolic techniques for planning with extended goals in non-deterministic domains // Proc. 6th European Conference on Planning (ECP 01). — 2001.

[146] *Pistore, M.* The planning spectrum - one, two, three, infinity // Proc. 18th Annual IEEE Symposium on Logic in Computer Science. — Washington DC (USA): IEEE Computer Society, 2003. — P. 234.

[147] *Pratt, V. R.* A near-optimal method for reasoning about actions / V. R. Pratt // *Computer Systems Sciense.* — 1980. — Vol. 2, no. 20. — Pp. 231–254.

[148] Program monitoring with LTL in EAGLE // Proc. 18th International Parallel and Distributed Processing Symposium. — Manchester (UK): 2004. — April. — Pp. 264–281.

[149] *Puterman, M. L.* Markov decision processes: Discrete stochastic dynamic programming / M. L. Puterman // *Operational Research Society.* — 1995. — Vol. 46, no. 6. — P. 792.

[150] Quantitative temporal reasoning / E. A. Emerson, A. K. Mok, A. P. Sistla, J. Srinivasan // *Real-Time Systems.* — 1992. — Vol. 4, no. 4. — Pp. 331–352.

[151] *Rao, A. S.* Asymmetry thesis and side-effect problems in linear-time and branching-time intention logics // Proc. 12th International Joint Conference on Artificial Intelligence (IJCAI 91) / Ed. by J. Myopoulos, R. Reiter. — Sydney (Australia): Morgan Kaufmann Publishers, 1991. — Pp. 498–505.

[152] *Rao, A. S.* Modeling rational agents within a BDI-architecture // Proc. Proceedings of Knowledge Representation and Reasoning (KR&R 91) / Ed. by R. Fikes, E. Sandewall. — San Mateo (USA): Morgan Kaufmann Publishers, 1991. — April. — Pp. 473–484.

[153] *Rao, A. S.* BDI agents: from theory to practice // Proc. First International Conference on Multi-Agent Systems. — San Francisco (USA): 1995. — June. — Pp. 312–319.

[154] *Rao, A. S.* Decision procedures for BDI logics / A. S. Rao, M. P. Georgeff // *Journal of Logic and Computation.* — 1998. — Vol. 8, no. 3. — Pp. 293–343.

[155] *Rao, A. S.* Modeling rational agents with a BDI-architecture / A. S. Rao, M. P. Georgeff // *Readings in agents.* — 1998. — Pp. 317–328.

[156] Reasoning about action and cooperation // Proc. Fifth International Joint Conference on Autonomous Agents and Multiagent Systems / Ed. by P. Stone, G. Weiss. — New York (USA): ACM Press, 2006. — Pp. 185–192.

[157] Reasoning about agents in the KARO framework // Proc. Eighth International Symposium on Temporal Representation and Reasoning (TIME'01). — Washington DC (USA): IEEE Computer Society, 2001. — P. 206.

[158] Rule-based runtime verification // Proc. Fifth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 04). — 2004. — January.

[159] *Sanner, S.* Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference // Proc. International Joint Conference On Artificial Intelligence. — 2005. — Pp. 1384–1391.

[160] SatPlan: Planning as satisfiability // Abstracts of the 5th International Planning Competition. — 2006.

[161] *Searle, J. R.* Speech acts: an essay in the philosophy of language / J. R. Searle. — Cambridge University Press, 1969. — P. 203.

[162] *Shannon, C.* A symbolic analysis of relay and switching circuits / C. Shannon // *Transactions of the AIEE.* — 1938. — Vol. 57. — Pp. 713–723.

[163] SHOP2: An HTN planning system / D. S. Nau, T. C. Au, O. Ilghami et al. // *Journal of Artificial Intelligence Research.* — 2003. — december. — Vol. 20. — Pp. 379–404.

[164] *Simon, G. A.* An extended finite state machine approach to protocol specification // Proc. Second International Workshop on Protocol Specification, Testing and Verification (IFIP WG6.1). — Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1982. — Pp. 113–133.

[165] *Singh, M. P.* A critical examination of the cohen-levesque theory of intentions // Proc. 10th European conference on Artificial intelligence. — Vienna (Austria): John Wiley & Sons, Inc., 1992. — Pp. 364–368.

[166] *Somenzi, F.* CUDD: Colorado University Decision Diagram package (http://vlsi.colorado.edu/ fabio/CUDD/).

[167] STeP: The stanford temporal prover: Tech. rep. / Z. Manna, A. Anuchitanukul, N. Bjorner et al. — Stanford, CA, USA: Stanford University, 1994.

[168] *Thati, P.* Monitoring algorithms for metric temporal logic specications // Proc. 4th Intl Workshop on Runtime Verication / Ed. by K. Havelund, G. Rou. — 2004.

[169] The tool KRONOS // Proc. Workshop on Hybrid systems III : verification and control (DIMACS/SYCON). — New Brunswick (USA): Springer-Verlag New York, Inc., 1996. — Pp. 208–219.

[170] UPPAAL — a tool suite for automatic verification of real-time systems // Proc. Workshop on Hybrid systems III : verification and control (DIMACS/SYCON). — New Brunswick (USA): Springer-Verlag New York, Inc., 1996. — Pp. 232–243.

[171] Using ARCHON to develop real-world DAI applications for electricity transportation management and particle acceleration control / N. R. Jennings, J. M. Corera, I. Laresgoiti et al. // *EEE Expert Special Issue on Real World Applications of DAI systems.* — 1996.

[172] *van der Hoek, W.* On the complexity of practical ATL model checking // Proc. Fifth International Joint Conference on Autonomous Agents and Multiagent Systems / Ed. by P. Stone, G. Weiss. — New York (USA): ACM Press, 2006. — Pp. 201–208.

[173] *van der Hoek, W.* Tractable multiagent planning for epistemic goals // Proc. First International Joint Conference on Autonomous Agents and Multiagent Systems. — Bologna (Italy): ACM Press, 2002. — P. 1167–1174.

[174] *van der Hoek, W.* Model checking cooperation, knowledge, and time — a case study. — 2003.

[175] *van der Hoek, W.* Time, knowledge, and cooperation: Alternating-time temporal epistemic logic and it's applications. — 2003.

[176] *van der Hoek, W.* Towards a logic of rational agency / W. van der Hoek, M. J. Wooldridge // *Logic Journal of IGPL.* — 2003. — Vol. 11, no. 2. — Pp. 135–159.

[177] *van der Hoek, W.* On the logic of cooperation and propositional control / W. van der Hoek, M. J. Wooldridge // *Artif. Intell.* — 2005. — Vol. 164, no. 1-2. — Pp. 81–119.

[178] *van Linder, B.* Formalizing abilities and opportunities of agents / B. van Linder, W. van der Hoek, J.-C. Meyer // *Fundameta Informaticae.* — 1998. — Vol. 34, no. 1,2. — Pp. 53–101.

[179] *van Riemsdijk, B.* Agent programming in Dribble: from beliefs to goals using plans // Proc. Second International Joint Conference on Autonomous Agents and Multiagent Systems. — Melbourne (Australia): ACM Press, 2003. — Pp. 393–400.

[180] *Wagner, F.* Modeling software with finite state machines: a practical approach / F. Wagner. — Auerbach Publications, 2006.

[181] *Wooldridge, M. J.* The logical modeling of computational multi-agent systems: Ph.D. thesis / University of Manchester. — 1992. — September. — P. 153.

[182] *Wooldridge, M. J.* Reasoning about rational agents / M. J. Wooldridge. — Cambridge, MA: The M. I. T. Presss, 2000.

[183] *Wooldridge, M. J.* Intelligent agents: Theory and practise / M. J. Wooldridge, N. R. Jennings // *The Knowledge Engineering Review.* — 1995.

[184] *Wooldridge, M. J.* The Gaia methodology for agent-oriented analysis and design / M. J. Wooldridge, N. R. Jennings, D. Kinny // *Autonomous Agents and Multi-Agent Systems.* — 2000. — Vol. 3, no. 3. — Pp. 285–312.

[185] *Yin, G. G.* Continuous-time markov chains and applications: a singular perturbation approach / G. G. Yin. — Springer, 1998.

[186] *Yin, G. G.* Discrete-time markov chains: two-time-scale methods and applications / G. G. Yin, Z. Qing. — Springer, 2005. — P. 347.

[187] *Yu, E.* Modelling strategic relationships for process reengineering: Ph.D. thesis / University of Toronto, Department of Computer Science. — 1995.