**Федеральное государственное автономное
образовательное учреждение высшего образования
«Российский университет дружбы народов»**

А. И. Панов

Интеллектуальные динамические системы

Учебно-методическое пособие

**Москва
Российский университет дружбы народов
2015**

В пособии рассмотрены основные методы, применяющиеся при построении интеллектуальных динамических систем (ИДС). Одним из основных свойств ИДС является свойство иерархичности, уровневости организации всех процессов, связанных с ИДС, начиная от управления такими системами и заканчивая процессами самоорганизации в их базе знаний.

Оглавление

Введение

Динамические интеллектуальные системы — результат интеграции интеллектуальных систем с динамическими системами. В общем случае это двухуровневые динамические модели, где один из уровней отвечает за стратегию поведения системы (или, как иногда говорят, носит делиберативный характер), а другой уровень отвечает за реализацию конкретной (в том числе, математической) модели.

К таким системам относятся сложные естественные системы, такие как экологические, социальные и политические системы, а также такие динамические системы, в которых зависимости настолько сложны, что не допускают своего обычного аналитического представления. Сложность задач управления, в которых существенная роль принадлежит экспертным суждениям и знаниям человека, заставляет в дополнение к количественным методам или вместо них применять такие подходы, в которых в качестве значений переменных допускаются не только числа, но и слова или предложения искусственного или естественного языка.

Потребность в моделях такого рода назрела в связи с развитием, например, беспилотных средств транспортного и иного назначения. В частности, в беспилотных автономных самолетах и вертолётах одним из уровней управления должен являться делиберативный уровень управления, решающий задачи, например, планирования полёта или выбора траектории или выбора цели. Другой уровень управления — назовем его активным — реализует требуемые действия. Например, на делиберативном уровне управления беспилотным вертолётом принимается решение о зависании над целью, тогда на активном уровне начинает работать математическая модель зависания, вырабатывающая требуемые управления для исполнительных механизмов.

Глава 1

Представление статических знаний

1.1 Логика предикатов первого порядка

Одним из наиболее изученных формальных языков является язык исчисления предикатов первого порядка. Существуют работы, где язык исчисления предикатов рассматривается как язык представления знаний, однако, это не главное его назначение и мы будем использовать его, главным образом, в качестве средства описания элементов конструкций других языков, более ориентированных на представление знаний.

Опишем вначале основные конструкции языка исчисления предикатов первого порядка и их интерпретацию в духе [3; 4].

1.1.1 Описание языка

Основные конструкции языка L – языка исчисления предикатов первого порядка называются формулами. Введем вначале *алфавит* языка L. Алфавит включает:

1. Счетное множество букв: $z, y, x, \ldots$, которое будем называть множеством символов для обозначения переменных языка.

2. Счетное множество букв $a, b, c, \ldots$, которое будем называть множеством символов для обозначения констант языка.

3. Счетное множество прописных букв $P, Q, \ldots$ для обозначения предикатных символов языка.

4. Счетное множество строчных букв $f, g, \ldots$ для обозначения функциональных символов.

5. Символы для логических связок $\rightarrow$ (влечет), $\neg$ (не).

6. Символ для квантора $\forall$ (для любого);

7. $(,)$ — скобки.

Предикатные буквы $P, Q, \ldots$ и функциональные буквы $f, g, \ldots$ могут быть n-местными или, как еще говорят, n-арными. Иначе говоря, с каждым предикатным или функциональным символом будем связывать некоторое натуральное число, равное числу его аргументов.

Определим теперь понятие формулы или правильно построенного выражения языка исчисления предикатов первого порядка. *Формулы* языка определяются индуктивным образом. Начнем с определения *терма* языка:

1. Переменная есть терм.

2. Константа есть терм.

3. Если $t_1, t_2, \ldots, t_m, \ldots, t_n$ — термы, а f и g — функциональные символы арности m и n, соответственно, то $f(t_1, t_2, \ldots, t_m)$ и $g(t_1, t_2, \ldots, t_n)$ также термы.

4. Если $t_1, t_2, \ldots, t_m, \ldots, t_n$ — термы, а P и Q — предикатные символы арности m и n, соответственно, то $P(t_1, t_2, \ldots, t_m)$ и $Q(t_1, t_2, \ldots, t_n)$ — атомарные формулы.

5. Атомарная формула есть формула.

6. Если A, B — формулы, то $(A \to B)$, $\neg A$, $\neg B$ — формулы.

7. Если A — формула, то $\forall x A$ — формула.

8. Всякое слово в алфавите языка является формулой тогда и только тогда, когда это можно показать с помощью конечного числа применений п.п. 1-7.

Таким образом, мы завершили одно из возможных определений языка исчисления предикатов первого порядка. Существуют и другие определения, однако, язык, определенный нами, является полным, т. е. в нем выразимо все то, что выразимо в языках (исчисления предикатов первого порядка), определенных любым иным способом.

Можно, например, определить логические связки $\wedge, \vee$(читается *и* и *или*), выразив их через связки $\to$ и $\neg$:

- $A \wedge B = \neg(A \to \neg B)$,
- $A \vee B = \neg A \to B$.

Квантор существования — $\exists$ (существует) также выражается через квантор всеобщности и отрицание: $\exists x A(x) = \neg \forall x \neg A(x)$.

Разумеется, $\wedge$, $\vee$ и $\exists$ с тем же успехом можно было бы включить в язык в качестве трех дополнительных символов. Есть, однако, некоторые преимущества в том, чтобы сохранить список символов как можно более коротким. Например, индуктивные определения и доказательства по индукции оказываются в этом случае короче.

В дальнейшем нам придется использовать понятия *свободного* и *связанного* вхождения переменной в формулу. Вхождение переменной x в формулу A называется связанным, если эта переменная следует за квантором существования или всеобщности, предшествующими формуле A. В противном случае, вхождение переменной называется свободным. Если в формуле A отсутствуют свободно входящие в нее переменные (т. е. либо все переменные связаны, либо просто отсутствуют), то формула называется *замкнутой формулой* или *предложением*. Атомарную замкнутую формулу будем называть *фактом*. В том случае, если язык состоит только лишь из предложений, то он называется пропозициональным языком, а буквы $A, B, \ldots$, входящие в формулы этого языка — пропозициональными переменными.

1.1.2 Основные понятия исчисления

Рассмотрим вкратце основные понятия исчисления предикатов первого порядка.

Введем вначале аксиомы исчисления предикатов:

1. $A \to (B \to A)$,
2. $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$,
3. $(\neg A \to \neg B) \to (B \to A)$.

А затем правила вывода:

Правило отделения: если выводимо A и выводимо $A \to B$, то выводимо B.

Правило подстановки: в любую аксиому на место любой пропозициональной переменной можно подставить любое предложение, предварительно переименовав пропозициональные переменные подставляемого предложения так, чтобы они не совпадали с пропозициональными переменными аксиомы.

Если в аксиомах 1 – 3 все переменные являются пропозициональными, то такое исчисление называется *пропозициональным исчислением* или *исчислением высказываний*.

Рассмотрим пример вывода в исчислении высказываний. Возьмем, например, три закона логики, сформулированные Аристотелем и называемые постулатами Аристотеля. В языке исчисления высказываний они записываются следующим образом: Пусть P — пропозициональная переменная исчисления высказываний.

Постулат 1 (закон тождества): $P \to P$.

Постулат 2 (закон исключения третьего): $P \vee \neg P$.

Постулат 3 (закон противоречия): $\neg(P \wedge \neg P)$.

Докажем один из постулатов, например закон тождества. Используем аксиому 1 и правило подстановки (вместо B подставим $P \to P$) и получим $A \to ((P \to P) \to A)$. Из аксиомы 2:

$$(A \to ((P \to P) \to C)) \to ((A \to (P \to P)) \to (A \to C)).$$

Теперь вместо A и C подставим P:

$$\underbrace{(P \to ((P \to P) \to P))}_{X} \to \underbrace{((P \to (P \to P)) \to (P \to P))}_{Y}.$$

Затем применим правило отделения: та часть последней формулы, которая обозначена через X является аксиомой, т. е. выводима, тогда в силу правила отделения, выводима формула, обозначенная через Y. Теперь применим правило отделения к Y:

$$\underbrace{(P \to (P \to P))}_{X'} \underbrace{\to (P \to P)}_{Y'}.$$

и, рассуждая таким же образом, получим, что Y' — выводимо. Таким образом, закон тождества Аристотеля является *теоремой* исчисления высказываний. Действуя таким же образом, можно доказать, что второй и третий постулаты Аристотеля также являются *теоремами* исчисления высказываний.

Однако, исчисление предикатов первого порядка не исчерпывается приведенными выше тремя аксиомами и правилами вывода. Смысл кванторов устанавливается еще двумя аксиомами и одним правилом вывода.

5. $\forall x((A \to B) \to (A \to \forall x B))$, где x не является свободной переменной в A;

6. $\forall t A(t) \to A(x)$, где t — терм, а x не содержится в t в качестве свободной переменной.

Четвертая аксиома называется аксиомой генерализации, а вторая — аксиомой спецификации.

Правило обобщения: $A \to \forall x A$, где x — свободная переменная в A.

Аксиомы 1–6 исчисления предикатов первого порядка (или математической логики первого порядка) называются *логическими* аксиомами, они описывают логические законы, справедливые всегда, независимо от предметной области. Если же к аксиомам 1–6 добавить еще и аксиомы, описывающие

некоторую предметную область, например, арифметику или теорию групп, то получим *формальную теорию* — формальную арифметику или формальную теорию групп, соответственно. При этом, разумеется, в алфавит языка следует ввести специальные функциональные символы, такие как сложение в арифметике или умножение в теории групп.

Словосочетание «первый порядок» относится исключительно к тому обстоятельству, что кванторы $\forall$ и $\exists$ действуют на некотором универсальном множестве U. Логика второго порядка разрешает одному из кванторов действовать на подмножествах множества U и на функциях из степеней U в U. Логика третьего порядка может использовать кванторы по множествам функций и т. д. Уже из этих разъяснений видно, что в логиках более высоких порядков (как говорят, более сильных логиках) используются и некоторые нелогические понятия, такие как множество.

1.1.3 Формальные системы и интпретация

Будем полагать, что если заданы некоторый алфавит, множество формул, множества аксиом и правил вывода, тотем самым задана некоторая формальная система. Иначе говоря, формальная система F представляет собой совокупность следующих объектов:

$$F = \langle T, P, A, \Pi \rangle,$$

где T — конечное множество символов; P — множество правил грамматики, применение которых к символам из T, позволяет строить правильно построенные формулы; — множество аксиом; Π — множество правил вывода.

Если среди аксиом имеются нелогические аксиомы (аксиомы, описывающие некоторую предметную область), то формальная система называется *формальной теорией.*

Определение 1.1.1 *Выводом (или доказательством) в формальной системе называется конечная последовательность правильно построенных формул*

$$A_1, A_2, \ldots, A_n,$$

таких что каждая из формул последовательности либо является аксиомой либо получена из предыдущих формул последовательности с использованием аксиом и правил вывода.

Формула A_n в этом случае называется *выводимой формулой* (или *теоремой*) формальной системы F.

Введем теперь некоторые алгебраические понятия, которые окажутся полезными в дальнейшем.

Определение 1.1.2 *n-арным отношением R на называется подмножество M^n и записывается как $R \subset M^n$.*

Пусть задано непустое множество — *универсум* M. Дадим следующее определение, следуя [5].

Определение 1.1.3 *Упорядоченную тройку* $S = \langle M, G, R\rangle$, *где универсум называется основным множеством,* G — *множество* n-*местных функций из* M^n *в* M, *а* R — *семейство* n-*арных отношений на* M, *будем называть алгебраической системой.*

Если в определении алгебраической системы S положить $R = \emptyset$, то такая алгебраическая система называется *алгеброй*, а если $G = \emptyset$, то тогда алгебраическая система называется *моделью*.

Иногда удобнее использовать понятие *многосортной* или *многоосновной* алгебраической системы, полагая существование нескольких основных множеств $M_1, M_2, \ldots, M_n$, вместо одного множества . Тогда функции из G и отношения из R становятся многосортными, т. е. с каждым местом в списке аргументов функции или отношения связывается некоторый сорт i, задаваемый, например, индексом одного из множеств из M_i $(i = 1, 2, \ldots, n)$, а сами функции и отношения определяются на декартовых произведениях множеств из числа основных множеств.

Пусть заданы формальная система F с языком L и алгебраическая система S. Определим отображение I, которое всякому константному символу a языка L системы F ставит в соответствие некоторый элемент $m \in M$ из алгебраической системы S, всякому n-местному предикатному символу P — n-местное отношение $R \subset M^n$, а всякому m-местному функциональному символу f — m-местную функцию $G : M^n \to M$. Тогда I называется *интерпретирующим отображением*, а алгебраическая система S — *моделью* или *интерпретацией* формальной системы F. Элементы множества , функции и отношения, соответствующие константным, функциональным и предикатным символам языка L в смысле отображения I, называют иногда интерпретациями константных, функциональных и предикатных символов языка L, соответственно.

Если интересоваться в большей степени собственно языком, нежели фактами, которые на нем записаны, то можно говорить и о моделях языка. Моделью Ω языка L называется пара $\langle M, I\rangle$, где и I уже использовавшиеся только что универсум и интерпретирующее отображение. Заметим, что при данном универсуме для символов языка L существует много различных интерпретаций.

1.1.4 Истинность в модели

Рассмотрим теперь вопрос об истинности или ложности формул языка L в некотором возможном мире, т. е. в модели.

В полной мере этот вопрос оказывается не таким уж простым. Например, не существует регулярного метода, позволяющего по произвольной формуле языка L решать, истинна она или ложна в алгебраической системе $\langle N, +, *, \sigma, 0\rangle$, где N — множество целых чисел, $+$, $*$ и σ — операции сложения, умножения

чисел и получения следующего числа, соответственно; 0 – константа нуль (которую можно рассматривать как нульместную функцию). Заметим попутно, что эта алгебраическая система называется стандартной моделью арифметики.

Чтобы определить понятие «Предложение (формула) A истинно в модели Ω», мы должны вначале разбить A на меньшие части и решить это вопрос для каждой из полученных частей. Если имеет вид $\neg$ или $\wedge$, то ясно, что вначале надо решить вопрос об истинности и . С другой стороны, если имеет вид $\forall xB$, то такое рассуждение не проходит, так как B может и не быть предложением, поэтому бессмысленно спрашивать, истинно в Ω или ложно. На самом деле, предполагается, что значения всякой свободной в формуле B переменной пробегают множество M. Поэтому, имеет смысл вопрос: является ли формула B истинной в модели Ω, если значением свободной переменной является m? Если для каждого m из M ответ на этот вопрос окажется утвердительным, то можно говорить, что A истинно в Ω. Если же в M найдется элемент m, для которого ответ будет отрицательным, то будем говорить, что A ложно в Ω. Однако, если B имеет вид $\forall yC$, придется вновь преодолевать те же трудности. Поэтому, определение истинности формул в модели является индуктивной процедурой, основанной на индуктивном определении формулы языка L из раздела 1.1.1.

Напомним что, интерпретирующее отображение I всякому константному символу a языка L ставит в соответствие некоторый элемент $m \in M$, всякому n-местному предикатному символу P — n-местное отношение $R \subset M^n$, а всякому m-местному функциональному символу f — m-местную функцию $G : M^m \to M$; иначе говоря, $I(a) = m$, $I(P) = R$ и $I(f) = G$, соответственно.

Итак, пусть, P — n-местный предикатный символ, $x_1, x_2, \ldots, x_n$ — свободные переменные. Будем говорить, что:

1. Атомарная формула $P(x_1, x_2, \ldots, x_n)$ выполняется на n-ке $(I(a_1), I(a_2), \ldots, I(a_n))$ в модели Ω, если $(I(a_1), I(a_2), \ldots, I(a_n)) \in I(P)$.

2. Формула $\forall x_1 \forall x_2 \ldots \forall x_n P(x_1, x_2, \ldots, x_n)$ истина в модели Ω, если формула $P(x_1, x_2, \ldots, x_n)$ выполняется на любой n-ке $(m_1, m_2, \ldots, m_n) \in I(P)$. Заметим, что если $a_1, a_2, \ldots, a_n$ — константы, то для атомарной формулы $P(a_1, a_2, \ldots, a_n)$ понятия выполнимости и истинности в модели Ω эквивалентны.

3. Формула $\neg A$ истинна в модели Ω, если A — ложна.

4. Формула $A \to B$ истинна в модели Ω, если A — истинна и B — истинна, либо A — ложна и B — ложна, либо A — ложна, а B — истинна (в модели Ω).

5. Формула может быть истиной только в силу п.п. 1–4.

В случае, когда A не истина в Ω, мы говорим, что A ложна в Ω, или что A не выполняется в Ω, или что Ω — модель предложения $\neg A$. Если дано множество предложений Σ, такое, что Ω является моделью каждого из предложений $\sigma \in \Sigma$, то Ω — модель этого множества. Предложение, истинное в каждой модели языка L, называется истинным.

Рассмотрим простой пример. Пусть в языке L определен трехместный предикатный символ «СЕМЬЯ». Интерпретирующее отображение I таково, что ставит в соответствие этому предикатному символу трехместное отношение «**СЕМЬЯ**». Т.е. I(«СЕМЬЯ»)=«**СЕМЬЯ**». Поскольку это отношение (как и всякая семья) конечно, то можно задать его в виде таблицы, каждая строка которой содержит имена членов одной семьи:

Мать	**Отец**	**Ребенок**
Аня	Петя	Вася
Галя	Женя	Коля

Тогда модель включает универсум M, состоящий из имен членов всех семей и трехместное отношение «**СЕМЬЯ**». Если мы желаем узнать что-либо о выполнимости формулы «СЕМЬЯ»(x, y, z) в построенной таким образом модели, то, рассматривая различные наборы значений (из универсума M) переменных x, y, z, увидим, что в таблице имеется, например, строка (Аня, Петя, Вася). И отсутствует строка (Аня, Петя, Коля). Это означает, что формула «СЕМЬЯ»(x, y, z) выполнима в построенной модели, а именно, выполняется на тройках (Аня, Петя, Вася) и (Галя, Женя, Коля) и не выполняется, например, на тройке (Аня, Петя, Коля). Отсюда следует, в частности, что предложение $\forall x \forall y \forall z$«СЕМЬЯ»$(x, y, z)$ ложно, а, к примеру, предложение $\exists x \exists y \exists z$«СЕМЬЯ»$(x, y, z)$ истинно в построенной модели,

Завершая этот раздел, вспомним о том, что язык L — язык исчисления предикатов первого порядка и на нем записаны аксиомы этого исчисления, а для формул этого языка определены правила вывода и на них распространяется понятие выводимости (раздел 1.1.2). Одно из основных утверждений, устанавливающих тесную связь между понятиями выводимости в исчислении предикатов первого порядка и истинности приведено ниже.

Теорема 1.1 (Гёделя о полноте) *Произвольное предложение языка L выводимо в исчислении предикатов первого порядка тогда и только тогда, когда оно истинно.*

1.2 Атрибутивная логика

1.3 Семантические сети

Глава 2

Представление процедурных знаний

2.1 Системы правил

2.2 Семиотическое представление

Глава 3

Пополнение знаний

3.1 Проблема привязки символов

3.2 Биологически правдоподобные методы

3.3 Выявление причинно-следственных связей

Глава 4

Планирование поведения

4.1 Классические алгоритмы планирования

4.1.1 Планирование как доказательство теорем

Одним из примеров системы доказательства теорем, использовавшейся для решения задачи планирования, является система QA3 [53]. В системе QA3 одно множество утверждений используется для описания начального состояния, а другое — для описания эффектов действий. Чтобы следить за тем, какие факты являются истинными и в каком состоянии, в каждый предикат включаются выделенные переменные состояния. Целевое условие описываются формулой с переменной, связанной квантором существования. Задача системы состоит в том, чтобы доказать существование состояния, в котором истинно целевое условие. В основе доказательства лежит метод резолюций.

Эксплуатация QA3 продемонстрировала её низкую вычислительную эффективность. Кроме того, для неё не существовало сколько-нибудь приемлемого решения *проблемы фрейма*. Суть этой проблемы состоит в том, что действие может иметь нелокальный эффект, иначе говоря, неясно какие формулы, описывающие состояние системы, изменяются при применении действия. Это, приводит к тому, что в описание действия включаются утверждения об изменении (не изменении) каждого факта, представленного в состоянии. Очевидно, что в сложных предметных областях описание эффектов действий значительно усложняется.

4.1.2 Планирование в пространстве состояний

Первым планировщиком, осуществляющим планирование в пространстве состояний, является STRIPS (STanford Research Institute Problem Solver).STRIPS изначально разрабатывался для решения задачи формирования плана поведения робота, перемещающего предметы через множество помещений.

Собственно, идея алгоритма STRIPS заимствована из системы GPS [55]. Метод, использованный в GPS, назывался анализ средств и целей (means-ends

analysis). Он подразумевает рассмотрение в текущем состоянии тех действий, которые имеют отношение к цели. Однако, при таком подходе возникает следующая проблема: применять ли действия, связанные с целью, немедленно, как только они найдены или же приостановить применение действия до тех пор, пока не будут найдены все действия имеющие отношение к цели? STRIPS применяет действия не откладывая, достигая каждой цели по отдельности.

МакДермот показал, что эффективность планирования с использованием метода анализа средств и целей может быть намного повышена задержкой применения действия до тех пор, пока не будут найдены все релевантные (относящиеся к цели действия) и повторением поиска релевантных действий заново, после каждого применения действия. Для решения проблемы фрейма STRIPS предлагает в состоянии, к которому применяется правило (действие), изменять выполнимость лишь тех формул, которые описаны в эффекте действия, а выполнимость всех остальных оставлять неизменной.

Рассмотрим постановку задачи планирования при классических допущениях в терминах STRIPS. Как и выше, фактом будем называть замкнутую атомарную формулу языка исчисления предикатов 1-го порядка (ИПП), а состоянием — некоторое множество фактов. Неформально, состояние представляет модель среды, в которой действует интеллектуальный агент. Приведём пример описания среды в терминах STRIPS:

$$s = \{ATR(a), AT(B, b), AT(C, c), \forall u \forall x \forall y((AT(u, x) \wedge (x \neq y)) \rightarrow \neg AT(u, y))\}.$$

Здесь, $ATR(a)$ означает, что «робот находится в комнате a», $AT(B, b)$ — «ящик B находится в комнате b», $AT(C, c)$ — «ящик C находится в комнате c». Имена конкретных объектов из этого множества: a, b, c — соответственно «комната a», «комната b», «комната c»; A, B, C — соответственно, «ящик A», «ящик B», «ящик C».

Действия агента будем описывать с помощью правил, при этом, для упрощения таких описаний, примем некоторые соглашения. При описании STRIPS-задачи планирования как в множествах добавляемых, так и в множествах удаляемых фактов будем использовать лишь атомарные формулы без функциональных символов. Пример правила:

Имя правила:	$Push(x, y, z)$
Условие:	$C(R) = \{ATR(y), AT(x, y)\}$
Список добавлений:	$A(R) = \{ATR(y), AT(x, y)\}$
Список удалений:	$D(R) = \{ATR(z), AT(x, z)\}$

В приведённом примере STRIPS-правило $Push(x, y, z)$ описывает действие робота по перемещению ящика x из комнаты y в комнату z. Здесь, x, y, z — переменные. Выполнение агентом действия сводится к применению правила. Применение правила модифицирует состояние s. Определение применимости

правила было приведено в разд. 2.1. Применение правила R преобразует состояние s в s' следующим образом:

$$s' = (s \setminus (D(R)\theta)) \cup (A(R)\theta)).$$

Это преобразование обозначается так: $s \overset{R,\theta}{\Rightarrow} s'$, где через θ обозначена подстановка элементов предметной области вместо переменных.

Определение 4.1.1 (STRIPS-допущение) *При применении некоторого правила R к состоянию s выполнимость факта $f \in s$ изменяется, только если факт f описан либо в списке удалений $D(R)$, либо в списке добавлений $A(R)$.*

Технически, при проверке применимости некоторого правила R, STRIPS выполняет полную подстановку на места всех переменных индивидов предметной области. Возможны различные варианты подстановок. Некоторые варианты подстановки могут давать примеры правил, применимых (или же неприменимых) в состоянии s. Однако, в алгоритм STRIPS можно внести незначительные модификации для применения не полностью означенных правил. В этом случае, в состоянии s появились бы факты с переменными в описании. Как будет видно далее, неполная подстановка активно используется планировщиками в пространстве планов. Соответствующее свойство этих планировщиков получило название *малого связывания* (least commitment).

Приведем постановку задачи STRIPS-планирования.

Определение 4.1.2 *Будем называть доменом планирования $P = \langle s_0, \Sigma R \rangle$, где s_0 — начальное состояние, ΣR — конечное множество правил.*

Определение 4.1.3 *Будем называть задачей планирования $T = \angle P, G \rangle$, где G — описание целевого факта агента, или просто цель.*

Решение задачи планирования T заключается в нахождении плана, который достигает цели G.

Определение 4.1.4 *План $Plan$ — это последовательность состояний $s_0, \ldots, s_n$, последовательность правил $R_1, \ldots, R_n$ и последовательность подстановок $\theta_1, \ldots, \theta_n$, такая что, G выполнима в s_n. Длина плана $Plan$ равна n:*

$$Plan : s_0 \overset{R_1,\theta_1}{\Rightarrow} s_1 \overset{R_2,\theta_2}{\Rightarrow} s_2 \ldots \overset{R_n,\theta_n}{\Rightarrow} s_n.$$

На вход алгоритма STRIPS подаётся множество правил ΣR, начальное состояние s_0, цель G.

Алгоритм 1 Алгоритм STRIPS

Вход: ΣR, s, G
Выход: $Plan$

1: $s = s_0$
2: **пока** G не выполнимо в s
3: выбрать компоненту g из G
4: выбрать правило $R \in \Sigma R$ такое, что $g \in A(R)$
5: STRIPS(ΣR,s,$C(R)$)
6: применить R к s
7: добавить R в $Plan$
вернуть $Plan$

Будем полагать, что в множестве ΣR все правила полностью конкретизированы. Вначале в стек целей помещается главная цель G. Если цель не является простой, т. е. содержит конъюнкцию литералов, то система STRIPS добавляет в стек в некотором порядке каждый из литералов составной цели (шаг 3). Когда верхняя цель стека является однолитеральной, система ищет действие (шаг 4), которое содержит в списке добавлений литерал, сопоставимый с этой целью. Если такое действие не применимо к текущему состоянию, тогда его предусловие помещается в стек целей, иначе действие применяется к текущему состоянию (шаг 6) и помещается в план $Plan$. Если верхняя цель в стеке соответствует текущему состоянию, то она удаляется из стека. Алгоритм STRIPS завершается, когда стек пуст.

Существуют задачи, для которых STRIPS либо не может построить план, либо находит не минимальный план. Причина этого кроется в том, что STRIPS удовлетворяет каждую компоненту составной цели по отдельности, без учёта их взаимосвязи. Особенность предметной области, где цели взаимосвязаны (взаимодействуют) получила название взаимосвязи целей.

4.2 Планирование с удовлетворением ограничений

В этом разделе рассмотрим алгоритм планирования — Graphplan, который использует технику прямого распространения ограничений [1]. В свое время (1997) Graphplan продемонстрировал хорошие результаты для ряда тестовых задач классического планирования. Создатели Graphplan'a объясняют этот успех способностью Graphplan'a анализировать множество планов одновременно.

Graphplan оказал сильное влияние на последующие работы в области планирования. Он принимает на вход стандартное STRIPS-описание задачи планирования и переводит это описание в компактную структуру, которая называется графом планирования, из которой впоследствии извлекает частично-упорядоченный план. Важно отметить, что граф планирования это не граф состояний, который получается при работе планировщика в пространстве со-

стояний. Graphplan сочетает в себе свойства как планировщика в пространстве состояний, так и планировщика в пространстве планов. Иначе говоря, он не обладает свойством малого связывания. Приведем определения основных понятий планировщика.

Определение 4.2.1 *Факты F — множество элементарных формул без переменных из домена планирования P.*

Перед основной стадией работы Graphplan создаёт множество действий, осуществляя для каждого правила $R \in \Sigma R$ всевозможные варианты подстановки индивидов на места всех переменных. Имеется также специальный вид действия «no-op» — «ничего не делать».

Определение 4.2.2 *Действия $Acts$ — множество полностью конкретизированных правил из ΣR, а также действие «no-op».*

Действие «no-op» имеет условие $C(\text{no-op}) = f$, список добавлений $A(\text{no-op}) = f$, и пустой список удалений $D(\text{no-op}) = \emptyset$, где f — произвольный факт из F.

Определение 4.2.3 *Граф планирования PG — ориентированный ярусный граф с двумя типами узлов и с тремя типами рёбер. Два типа узлов в PG таковы:*

- *FN — множество узлов, ассоциированных с фактами F, и*
- *AN — множество узлов, ассоциированных с действиями $Acts$.*

Ассоциацию некоторого факта $f \in F$ с узлом $fn \in PG$, будем обозначать как $fn \to f$. Ассоциацию некоторого действия $act \in Acts$ с узлом $an \in AN \subset PG$, будем обозначать как $an \to act$.

Множество узлов PG разбито на не пересекающиеся подмножества

$$\langle FL_0, AL_0, FL_1, AL_1, \ldots, AL_{n-1}, FL_n \rangle,$$

где FL — ярус, содержащий узлы-факты, AL — ярус, содержащий узлы-действия, FL_0 содержит узлы-факты, соответствующие фактам начальной ситуации S_0.

Конфликт — это отношение взаимоисключения между двумя узлами на одном ярусе. Существуют конфликты между действиями и между фактами.

Определение 4.2.4 *Конфликты MXF — отношения взаимоисключения между узлами-фактами (fn_1, fn_2), где fn_1, fn_2 — узлы-факты, находящиеся на одном ярусе, такие, что: либо все действия на предыдущем ярусе, добавляющие факт fn_1, удаляют факт fn_2; либо все действия на предыдущем ярусе добавляющие факт fn_2, удаляют факт fn_1.*

Определение 4.2.5 *Конфликты MXA — отношения взаимоисключения между узлами-действиями (an_1, an_2), где an_1, an_2 — узлы-действия, находящиеся на одном ярусе, такие, что действие an_1 удаляет условие или же эффект действия an_2, либо предусловие действия an_1 и предусловие действия an_2 состоят в конфликте $mxf \in MXF$.*

Заметим, что конфликт между парой узлов n_1 и n_2, может иметь место на некотором ярусе L и не иметь места на некотором последующем ярусе L'. С другой стороны, если между парой узлов n_1 и n_2 на некотором ярусе L не существует конфликта, то и на последующих ярусах после L, пара узлов n_1 и n_2 не будет конфликтовать.

Конфликты превращают граф планирования в граф ограничений в смысле CSP-задачи. Метод, который используется для построения графа планирования, называется прямым распространением ограничений.

Любой ярус $AL_i \in PG$ содержит узлы-действия $an \rightarrow act$, такие что $Nodes(C(act \leftarrow an)) \in FL_i$ и не существует $fn_1, fn_2 \in Nodes(C(act \leftarrow an))$ и $(fn_1, fn_2) \in MXF$, где $Nodes(C(act \rightarrow an))$ — узлы на ярусе FL_i, ассоциированные, с фактами из предусловия $C(act)$. Любой ярус фактов $FL_i \in PG$ $(i > 0)$ содержит узлы-факты $fn \rightarrow f$, такие что, для любого $an \in AL_{i-1} \in PG$ справедливо $(f \in D(act \leftarrow an)$ или $f \in A(act \leftarrow an))$.

Рёбра устанавливаются между узлами, расположенными на ярусах. Три типа рёбер PG таковы:

- ребро-предусловие — устанавливается между узлом-фактом $fn \rightarrow f$ на некотором ярусе FL_i и узлом-действием $an \rightarrow act$ на ярусе AL_i, если факт $f \in C(act)$;

- ребро-добавление — устанавливается между узлом-действием $an \rightarrow act$ на некотором ярусе AL_i и узлом-фактом $fn \rightarrow f$ на ярусе FL_{i+1}, если $f \in A(act)$;

- ребро-удаление — устанавливается между узлом-действием $an \rightarrow act$ на некотором ярусе AL_i и узлом-фактом $fn \rightarrow f$ на ярусе FL_{i+1}, если $f \in D(act)$.

Из определения видно, что ярусы PG чередуются так: ярус фактов | ярус действий и т. д. Первый ярус графа содержит факты, характеризующие начальное состояние. Ярусы в PG от самого первого до последнего содержат: факты, истинные в момент времени 1, действия, возможные в момент времени 1, факты, истинные в момент времени 2, действия, возможные в момент времени 2 и т.д. По сути, граф планирования PG позволяет представлять пространство состояний без разделения. Точнее, множество состояний хранящиеся совместно, например, на ярусе FL_{j+1}, получаются в результате всевозможных альтернативных вариантов применения действий, расположенных в ярусах AL_i по AL_j $(i < j)$, к некоторому состоянию s, представленному на ярусе фактов FL_i. Однако, ясно, что альтернативная перестановка k действий

может привести к тому, что одно из действий может удалять эффект, либо условие другого действия. Для обработки подобных ситуаций используются так называемые конфликты между действиями и конфликты между фактами. Это позволяет при необходимости, например, на этапе извлечения плана, выделить из графа PG альтернативные компоненты пространства состояний. Планировщики, использующие такой способ представления пространства состояний, получили название *дизъюнктивных планировщиков.*

Определение 4.2.6 *Пара ярусов фактов FL_i и FL_j — идентична, если FL_i и FL_j содержат одинаковые факты и одинаковые конфликты.*

Определение 4.2.7 *Граф Планирования PG является стабилизированным, если существуют пара смежных ярусов фактов FL_i и FL_{i+1} и FL_i идентичен FL_{i+1}.*

Справедливо следующее утверждение, которое мы приведем без доказательства.

Утверждение 4.1 *Пусть граф PG стабилизирован, и имеется пара идентичных ярусов-фактов $FL_i, FL_{i+1} \in PG$. Тогда, ярус фактов $FL_k \in PG$ идентичен ярусу фактов $FL_i \in PG$, где $k > i \in \mathbb{N}$.*

Цель G является разрешимой (достижимой) в следующих случаях:

- если она удовлетворяется тривиальным образом, т. е. компоненты цели G присутствуют в начальном ярусе фактов,
- если в графе PG существует подграф $Plan$, который состоит из множества путей, идущих от начального яруса фактов к ярусу фактов, содержащему G, и в этом множестве путей нет ни одной пары конфликтующих узлов.

Имеют место следующие утверждения:

- план длиной n можно извлечь из графа планирования PG, содержащего n ярусов-действий;
- алгоритм GraphPlan возвращает «план не существует», только если цель G не достижима;
- алгоритм GraphPlan обладает полнотой.

В начале Graphplan формирует первичный ярус фактов FL_0. Выполнение алгоритма включает ряд стадий (переменная t в алгоритме). На каждой стадии выполняется расширение графа планирования PG и поиск плана в графе PG.

Алгоритм 2 Алгоритм GraphPlan

Вход: $Acts, S_0, G$
Выход: $Plan$

1: $t = 0$ // *номер начальной стадии*
2: $GS = \emptyset$ // *множество для хранения разрешимых/неразрешимых целей*
3: $PG.FL_0 \leftarrow S_0$
4: **пока** PG не стабилизирован или цель G не разрешима
5: SpreadGraph(PG)
6: SearchPlan(PG)
7: $t = t + 1$ // *переход к следующей стадии*
8: **если** «цель G разрешима» **то**
9: **вернуть** $Plan$
10: **иначе**
11: **вернуть** $Plan = \emptyset$

На этапе расширения графа PG на основе текущего яруса фактов создаётся новый ярус действий, а затем, на основе нового яруса действий, формируется новый ярус фактов. Во вновь сформированных ярусах выявляются конфликты MXF и MXA (процедура SpreadGraph).

Graphplan строит частично-упорядоченный план. Извлечение плана осуществляется с помощью техники обратного хода от текущего яруса к начальному ярусу. Эта техника позволяет более эффективно использовать информацию о конфликтах между действиями и фактами в графе PG. Опишем эту технику.

Перед поиском плана Graphplan проверяет следующее условие: «справедливо ли, что $GN \subseteq FL_{\text{тек}}$ и для каждой пары узлов $gn_1, gn_2 \in GN$ и $gn_1, gn_2 \notin mxf$, где GN — множество целевых фактов, ассоциированных с узлами на ярусе $FL_{\text{тек}}$». Если это так, тогда возможно план существует, и Graphplan приступает к поиску. Суть поиска плана сводится к тому, чтобы от целевых фактов в текущем ярусе $GN \subseteq FL_{\text{тек}}$, выделить путь, ведущий к ярусу FL_0. В пути не должны содержаться конфликтные действия. На основе выделенного пути формируется план. Более точно происходит следующее: формируется множество GS — хранилище (под)целевых наборов GS_i^t. GS_i^t — множество целей, выбираемые из яруса фактов с номером i, при поиске плана на стадии t.

Начиная с текущего яруса FL_i (вначале $i = t$) в GS_i^t заносятся целевые факты $GN \subseteq FL_i$. Далее на ярусе действий с номером $i - 1$ выделяются всевозможные комбинации действий (A_{comb}), доставляющие GS_i^t (множество $Comb$). Устанавливается подцель GS_{i-1}^t, в которую помещаются предусловия выделенных действий, расположенные на ярусе фактов FL_{i-1}. Для каждой из комбинаций действий $A_{comb} \subseteq Comb$ процесс продолжается рекурсивно, до тех пор, пока GS_{i-1}^t окажется тривиально разрешимой, либо не найдётся комбинации действий, доставляющей GS_{i-1}^t, т. е. $Comb = \emptyset$.

Алгоритм 3 Функция SpreadGraph

функция SpreadGraph(FL_t)
  // *создание следующего яруса действий* AL_{t+1}
  для всех действий $act \in ACTS$, применимых в FL_t
    если для каждой пары фактов $f_1, f_2 \in C(act)$ не существуют узлы $fn_1 \to f_1$, $fn_2 \to f_2$, такие что $fn_1, fn_2 \in FL_t$ и $fn_1, fn_2 \in mxf$ **то**
      создать в AL_{t+1} узел-действие $an \to act$
      создать рёбра-предусловия между an и соответствующими узлами $fn \in FL_t$
  выявить конфликты mxa на ярусе AL_{t+1}
  // *создание следующего яруса фактов* FL_{t+1}
  для всех узлов $an \in AL_{t+1}$
    добавить в FL_{t+1} узлы-факты, ассоциированные с фактами из $A(act \leftarrow an)$ и $D(act \leftarrow an)$
    соединить соответствующими рёбрами добавления или рёбрами удаления узлы FL_{t+1} и узлы AL_{t+1}
  выявить конфликты mxf на ярусе FL_{t+1}

Если подцель GS^t_{i-1} оказывается разрешимой, то при возврате из рекурсии в план $Plan$ помещаются тройки $\langle GS^t_{i-1}, A_{comb}, GS^t_i \rangle$, в которой для каждого действия из A_{comb}, известно какие (под)цели из GS^t_i достигает действие, и какие цели из GS^t_{i-1} необходимо достичь, прежде чем выполнить действие.

Для получения линейного плана необходимо выполнить топологическую сортировку нелинейного плана $Plan$, с учётом целевых ограничений GS^t_i. При реализации алгоритма для повышения эффективности обратной техники извлечения плана, используется хеширование. В хеш-таблице на каждой стадии t запоминаются целевые наборы GS^t_i, которые оказались не разрешимыми в ярусе фактов i. На каждой стадии при поиске плана проверяется наличие в хэш-таблице разрешаемой подцели GS^t_{i-1}. Если подцель GS^t_{i-1} в хеш-таблице, то поиск плана немедленно прекращается, и исходная цель GS^t_i, также помещается в хеш, как неразрешимая.

4.3 Нелинейное планирование

Составление частично-упорядоченных планов рассмотрим на примере планировщика SNLP (Systematic Nonlinear Planning) [2].

Учитывая, что план это последовательность операторов, очевидно, что два плана можно рассматривать как эквивалентные, если один из них может быть выведен из другого посредством переупорядочивания невзаимодействующих шагов. Например, рассмотрим робота, который должен выполнить набор задач в комнате A и набор задач в комнате B. Каждая задача формально ассоциируется с конкретным целевым высказыванием. Мы хотим достичь (сде-

Алгоритм 4 Функция SearchPlan

функция SearchPlan
// результат: «цель G разрешима»/«цель G пока не разрешима»
    если $GN \subseteq FL_t$ **то**
        добавить GN в GS_i^t, где i — номер текущего яруса фактов, t — номер текущей стадии, GN — множество узлов-фактов, ассоциированных с целевыми компонентами
    если если CheckGoal(GS_i^t) **то**
        вернуть «цель G разрешима»
    иначе
        вернуть «цель G пока не разрешима»

Алгоритм 5 Функция CheckGoal

функция CheckGoal(GS_i^t)
// выход: «GS_i^t разрешимо»/«GS_i^t не разрешимо»
// GS_i^t — множество (под)целей на i-ом ярусе фактов
    $Comb$ — множество всех комбинаций A_{comb} действий яруса AL_{i-1} таких, что любая пара действий в A_{comb} не состоит в конфликте $mxa \in MXA$, и при этом A_{comb} доставляет GS_i^t
    для всех $A_{comb} \subseteq Comb$
        создать подцель GS_{i-1}^t, состоящую из предусловий действий в A_{comb}
    если цели GS_{i-1}^t разрешимы **то**
        добавить тройку $\langle GS_{i-1}^t, A_{comb}, GS_i^t \rangle$ в $Plan$
        вернуть «GS_i^t разрешимо»
    иначе если цели в GS_i^t не разрешимы **то**
        вернуть «GS_i^t не разрешимо»

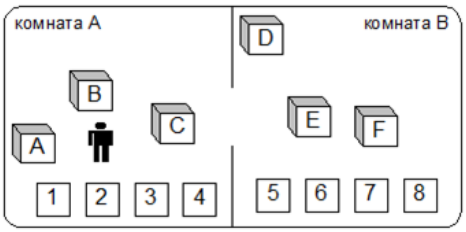
лать истинными) высказывания $P_1, \ldots, P_n, Q_1, \ldots, Q_m$. У нас есть операторы $A_1, \ldots, A_n, B_1, \ldots, B_m$, где A_i действие делает истинным P_i высказывание и при этом должно выполняться в комнате A, а B_i действие делает истинным Q_i высказывание и должно выполняться в комнате B. Более формально, каждое A_i имеет одно предусловие $IN(A)$, добавляет высказывание P_i и не удаляет никаких высказываний. Каждое B_i определяется подобным образом с предусловием $IN(B)$.

Пусть у нас также есть оператор $GO(A)$ без предусловий со списком добавления $IN(A)$ и списком удаления $IN(B)$. У нас также есть и аналогичный оператор $GO(B)$. Множество целей $\{P_1, \ldots, P_n, Q_1, \ldots, Q_m\}$ может быть достигнуто (независимо от начального состояния) посредством плана

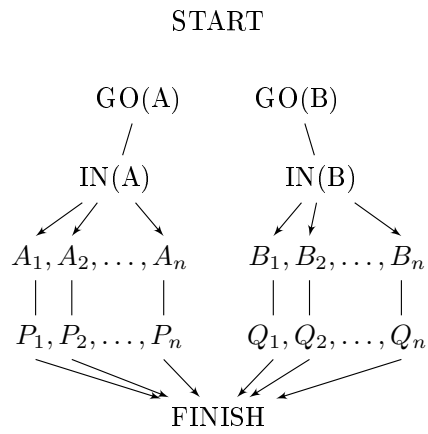
$$GO(A), A_1, \ldots, A_n, GO(B), B_1, \ldots, B_m.$$

Очевидно, что порядок A_i шагов и порядок B_i шагов не имеет значения, при условии, что все A_i шаги будут выполнены в комнате A, а все B_i ша-

ги в комнате B. Этот план следует рассматривать как эквивалентный плану $GO(A), A_n, \ldots, A_1, GO(B), B_m, \ldots, B_1$, в котором шаги A_i и B_i выполняются в обратном порядке. Каждый (линейный) план, который является решением данной проблемы планирования, может быть обобщен «нелинейным планом», который поддерживает только частичный порядок на шагах плана. Два линейных плана рассматриваются как эквивалентные, если они являются разными представлениями одного и того же нелинейного (частично-упорядоченного) плана (см. рис. 4.1a).



(a) Пример частично-упорядоченного плана



(b) Причинные связи в линейном плане

Рис. 4.1: Пример SNLP планирования

Чтобы определить нелинейный план, связанный с данным линейным планом, мы сначала должны преодолеть небольшую техническую проблему. В линейном плане мы можем именовать отдельные шаги плана, ссылаясь на них, как на первый шаг, второй шаг и т. д. Однако в нелинейном плане может не быть четко определенного второго шага. Мы также не можем именовать шаги именами операторов, используемых в этих шагах, т. к. несколько шагов могут использовать один и тот же оператор. Для именования шагов в нелинейном плане мы будем полагать, что с каждым шагом ассоциировано отдельное символьное имя, которое называется именем шага.

Определение 4.3.1 *Таблица символов это отображение из конечного множества имен шагов в операторы. Каждая таблица символов должна содержать два особенных имени шага, называемых START и FINISH. START отображается на некий фиктивный оператор, который не имеет предусловий и списка удалений, но добавляет множество «начальных высказываний». Аналогично, FINISH отображается на оператор, который имеет множество предусловий, называемых «целевыми формулами», но имеет пустые списки добавления и удаления.*

Определение 4.3.2 *Причинная связь — это тройка* $\langle S, f, W\rangle$*, где* f *— некоторый факт,* W *— шаг, имеющий в предусловии факт* f*,* S *— шаг, имеющий факт* f *в списке добавлений.*

Определение 4.3.3 *Угроза* V *для причинной связи* $\langle S, f, W\rangle$ *— это шаг, который либо добавляет, либо удаляет факт* f*, и при этом не является ни шагом* S*, ни шагом* W*.*

Определение 4.3.4 *Защитное ограничение — это отношение порядка* $<$*, заданное на шагах плана, при этом* $S < W$ *означает, что шаг* S *должен быть выполнен до шага* W*, и наоборот,* $S > W$ *означает, что шаг* S *должен быть выполнен после шага* W*.*

Определение 4.3.5 *Нелинейный план* $Plan = \langle ST, CL, SC\rangle$*, где* ST *— множество шагов,* CL *— множество причинных связей,* SC *— множество защитных ограничений.*

Заметим, что нелинейный план $Plan$ обладает полнотой, если:

- каждый шаг плана участвует либо в причинной связи, либо в защитном ограничении;
- любой шаг W с предусловием f состоит в некоторой причинной связи $\langle S, f, W\rangle$;
- имеется шаг-угроза V для причинной связи $\langle S, f, W\rangle$ и нелинейный план содержит либо защитное ограничение $V < S$, либо защитное ограничение $V > W$.

Определение 4.3.6 *Топологическая сортировка нелинейного плана* $Plan$ *— это линейная последовательность всех шагов, которая удовлетворяет следующим условиям:*

- *первый шаг в последовательности —* $START$*;*
- *последний шаг в последовательности —* $FINISH$*;*
- *для каждой причинной связи* $\langle S, f, W\rangle$ *шаг* S *в последовательности предшествует шагу* W*;*
- *для каждого защитного ограничения* $U < V$ *шаг* U *в последовательности предшествует шагу* V*.*

Топологическая сортировка нелинейного плана является решением, если применение последовательности действий шагов между шагами START и FINISH из начального состояния, которое задаётся списком добавлений шага START, приводит в состояние, в котором содержатся все предусловия шага FINISH.

Теорема 4.1 *Любая топологическая сортировка нелинейного плана, обладающего полнотой, является решением задачи планирования T.*

Определение 4.3.7 *Нелинейный план является противоречивым, если на нём невозможно осуществить топологическую сортировку.*

Таким образом, противоречивый нелинейный план не является решением задачи планирования.

Теорема 4.2 *Алгоритм SNLP систематичен, т. е. в процессе поиска, осуществляемого в пространстве частично-упорядоченных планов, один и тот же план или эквивалентные планы никогда не рассматриваются дважды.*

Алгоритм 6 Алгоритм SNLP

Вход: $\sum R, Plan$

Выход: $Plan$, «план построен»/«план не построен»

1: **если** $Plan$ противоречив **то**

2: **вернуть** $Plan = \emptyset$, «план не построен»

3: **если** $Plan$ обладает полнотой **то**

4: **вернуть** $Plan$, «план построен»

5: **если** в $Plan$ имеется шаг-угроза V для причинной связи $\langle S, f, W\rangle$ и $Plan$ не содержит защитного ограничения $V < S$ или $V > W$ **то**

6: недетерминированно выполнить
(a) либо SNLP($Plan$+$(V < S)$),
(b) либо SNLP($Plan$+$(V > W)$)

7: **если** существует некоторый шаг W с предусловием f, для которого не существует причинной связи $\langle S, f, W\rangle$ **то**

8: недетермированно выполнить либо
(a) выбрать недетерминированно некоторый шаг S, добавляющий f и выполнить SNLP($Plan$+$\langle S, f, W\rangle$),
(b) либо выбрать недетерминированно правило $R \in \sum R$, добавляющее f, и создать новый шаг S, ассоциированный с выбранным правилом R, и выполнить SNLP($Plan$+$\langle S, f, W\rangle$)

На вход процедуры подаётся множество правил $\sum R$, а также, нелинейный план $Plan$, не обладающий полнотой, который содержит шаги START и FINISH. Далее $Plan$ уточняется путём добавления причинных связей и защитных ограничений, до тех пор, пока не обнаружится такое уточнение, что план либо противоречив, либо обладает полнотой.

4.4 Графические системы планирования

Глава 5

Системы, основанные на правилах

5.1 Состояния и траектории

5.2 Синтез управления

5.3 Синтез обратной связи

5.4 Основы теории управляемости

Глава 6

Практические задания в системе Jadex

6.1 Задачи с международного соревнования планировщиков

Ниже представлен список задач с одного из треков одной из самых главных конференций по планированию — ICAPS за 2014г. Представленный трек из программы International Planning Competition 2014 включает в себя задачи по детерминированному планированию.

6.1.1 Бармен

Автор — Sergio Jiménez Celorrio.

Представим себе робота-бармена, который орудует дозаторами, стаканами и шейкером. Цель планировщика — построить план действий робота по приготовлению необходимого количества коктейлей. Необходимо учесть, что манипуляторы робота могут брать только один предмет за раз, а стаканы должны быть пустыми и чистыми, прежде чем их начинать заполнять.

6.1.2 Дайвинг в пещерах

Авторы: Nathan Robinson,Christian Muise, and Charles Gretton.

Представим себе группу дайверов, каждый из которых может переносить по 4 баллона с воздухом. Необходимо нанять этих дайверов для спуска в подводную пещеру. У них стоит задача фотосъемки либо задача доставки полных баллонов воздуха для подготовки спуска других дайверов. Пещера слишком узкая, чтобы пропустить более одного дайвера за раз.

Пещера является разветвленной и может быть представлена в виде ненаправленного ациклического графа. У всех дайверов единственная точка входа. Определенные конечные точки ответвлений пещеры являются целями для

фотографирования. Как задача фотосъемки, так и обычного плавания расходуют воздух из баллонов. В конце дайверы должны покинуть пещеру и подняться на поверхность. Следовательно, они могут сделать только один спуск в пещеру.

Некоторые дайверы не уверены в других и будут отказываться работать, если кто-то из них не работал прежде со своим коллегой. Стоимость оплаты труда дайвера обратно пропорциональна количеству времени, которое они тратят на работу.

6.1.3 Детская закуска

Авторы: Raquel Fuentetaja, Tomás de la Rosa Turbides.

Задача состоит в том, чтобы приготовить и подать бутерброды группе детей, у некоторых из которых аллергия на глютен. Есть два действия по приготовлению бутербродов из их ингредиентов. Первое из них готовит один бутерброд, а второй делает то же, но с учетом того, что все ингредиенты должны быть без глютена. Есть также действия положить один бутерброд и подать несколько бутербродов.

В начальных условиях даны ингредиенты для приготовления бутербродов. Цели заключаются в обслуживании детей бутербродами, к которым у них нет аллергии.

6.2 Внешняя среда и типы агентов

6.3 Задание состояний

6.4 Задание правил и стратегий

6.5 Планирование поведения

6.6 Задачи по планированию

Заключение

Немного о итогах курса

Список литературы

1. *Blum A. L.*, *Frust M. L.* Fast planning through planning graph analysis // Artificial Intelligence. — 1997. — Vol. 90, 1-2. — Pp. 281–300.
2. *McAllester D.*, *Rosenblitt D.* Systematic Nonlinear Planning // Proceedings of the Ninth National Conference on Artificial Intelligence AAAI-91. — 1991. — Vol. 2. — Pp. 634–639.
3. *Кейслер Г.*, *Чен Ч. Ч.* Теория моделей. — М. : Мир, 1977.
4. *Клини С.* Математическая логика. — М. : Мир, 1973.
5. *Мальцев А. И.* Алгебраические системы. — М. : Наука, 1970.