В.А. Лапшин

Система Cyc и ее библиотека онтологий

Аннотация. Работа посвящена описанию библиотеки онтологий, предоставляемой программной системой под названием Cyc. Данная статья представляет собой вторую часть работы, посвященной описанию данной системы. В первой части рассказывалось о мотивах создания системы, ее истории (системе уже 25 лет), а также было дано описание языка CycL, на котором описываются факты, добавляемые в онтологии Cyc. Вторая часть посвящена описанию систем логического вывода, использующихся в Cyc, а также описанию структуры онтологий системы Cyc, в основном редко изменяемой части этой структуры — онтологии верхнего уровня системы Cyc.

Ключевые слова: базы знаний, онтологии, онтологии верхнего уровня, Cyc, OpenCyc.

1. Система логического вывода

Система логического вывода Cyc (Cyc Inference Engine) поддерживает выводы на основе модус поненс и модус толленс[1], использование кванторов всеобщности и существования, а также математические преобразования. Логический вывод производится в контексте микротеорий, позволяя таким образом ограничивать рассуждение суждениями, относящимися только к данной микротеории и ко всем ее родительским микротеориям.

Система Cyc позволяет расширять встроенные процедуры логического вывода новыми модулями. Таких модулей достаточно много. Например, в системе Cyc используются процедуры вывода языка Prolog, а также система вывода, основанная на фреймовом подходе. Имеются также модули, которые непосредственно не осуществляют логический вывод, но часто используются как вспомогательные инструменты в процедурах логического вывода. Например, имеется модуль, который находит все экземпляры данного класса, или модуль, который осуществляет подстановку термов на основе аксиом эквивалентности.

Внутреннее представление суждений. В предыдущих версиях Cyc формулы хранились в том же виде, в каком они вводились пользователями системы. Например:

```
(implies
  (and
    (isa ?afp AdultFemalePerson)
    (residesInRegion ?afp Guam))
  (and
    (acquaintedWith Zippy ?afp)
    (likesAsFriend Zippy ?afp)))
```

В Cyc-10 формулы сохраняются в конъюнктивной нормальной форме[2]. Когда формула добавляется в онтологию Cyc, она автоматически преобразуется в конъюнктивную нормальную форму. За это ответственен специальный мо-

[1] Модус поненс или правило отделения позволяет осуществлять вывод следующим образом: если в онтологии имеются формулы A и $A \Rightarrow B$, то в онтологию также можно добавить и формулу B. Модус толленс представляет собой обращение правила модус поненс: если в онтологии имеются формулы $\neg B$ (отрицание B) и $A \Rightarrow B$, то можно также добавить и формулу $\neg A$.

[2] Говорят, что формула F находится в конъюнктивной нормальной форме, если она имеет вид $F = F_1 \wedge F_2 \wedge \ldots \wedge F_n$, где каждая из формул $F_1, F_2, \ldots F_n$ представляет собой дизъюнкцию литералов [5]. Под литералом подразумевается формула, состоящая из единственного предиката, аргументами которого являются несоставные термы или их отрицания. Фактически, литералы – это базовые атомарные формулы, только расширенные формулами, где в качестве аргументов могут выступать отрицания термов. Известно, что любое высказывание можно автоматически преобразовать в конъюнктивную нормальную форму.

дуль – *канонизатор* (canonicalizer). Вот как будет канонизирована формула из примера выше:

```
(and
  (or
    (not (isa ?afp AdultFemalePerson))
    (not (residesInRegion ?afp Guam))
    (acquaintedWith Zippy ?afp))
  (or
    (not (isa ?afp AdultFemalePerson))
    (not (residesInRegion ?afp Guam))
    (likesAsFriend Zippy ?afp)))
```

Универсальные кванторы в Cyc-10 обрабатываются тривиальным образом: переменные, связанные с универсальным квантором, считаются свободными. Как уже говорилось выше, формулы, содержащие кванторы существования, подвергаются сколемизации. Например, формула

```
(forAll ?A
  (implies
    (isa ?A Animal)
    (thereExists ?M
      (and
        (mother ?A ?M)
        (isa ?M FemaleAnimal)))))
```

сначала будет приведена к виду

```
(implies
  (isa ?A Animal)
  (and
    (mother ?A (SkolemFunctionFn (?A) ?M))
    (isa (SkolemFunctionFn (?A) ?M)
FemaleAnimal)))
```

после чего подвергнется сколемизации. В результате, получим формулу:

```
(and
  (or
    (not (isa ?A Animal))
    (mother ?A (SkolemFunctionFn (?A) ?M)))
  (or
    (not (isa ?A Animal))
    (isa (SkolemFunctionFn (?A) ?M)
FemaleAnimal))).
```

Модуль канонизации автоматически генерирует имя для каждой сколемовской функции, которую ему необходимо создать во время канонизации. Это имя позже может быть заменено пользователем на более осмысленное. Кроме того, канонизатор также описывает свойства сколемовской функции: местность, типы аргументов – совершенно так же, как в случае обычной функции. Вот, например, как после применения алгоритма сколемизации (канонизации) будет выглядеть в действительности формула из примера выше:

```
(and
  (or
    (isa SKF-8675309 SkolemFunction))
  (or
    (arity SKF-8675309 1))
  (or
    (arg1Isa SKF-8675309 Animal))
  (or
    (resultIsa SKF-8675309 FemaleAnimal))
  (or
    (not (isa ?U Animal))
    (mother ?U (SKF-8675309 ?U)))
  (or
    (not (isa ?U Animal))
    (isa (SKF-8675309 ?U) FemaleAnimal)))
```

Модуль канонизации также запоминает порядок литералов в добавляемой формуле. Это необходимо для того, чтобы быстро сравнивать формулы друг с другом во время логического поиска. Если порядок литералов зафиксирован, то во время поиска можно сосредоточиться только на структурном сравнении формул.

1.1. Логический вывод: введение

Обратный логический вывод (backward inferencing) в Cyc – это тип логического вывода, инициирующийся операцией ASK[3] Обратный вывод можно представлять как поиск по дереву, где каждый узел дерева – это формула, для которой проверяется связывание с переданными константами, а ребра между узлами представляют собой трансформацию суждения – объекта проверки.

Поясним сказанное на примере. Предположим, проводится поиск объектов, на которых истинна формула `(likesObject ?x ?y)`. Эта формула будет корнем дерева в описанной выше процедуре обратного логического вывода. Для ответа на запрос необходимо найти суждения, которые помогут связать переменные `?x` и `?y` с конкретными объектами. В онтологии может быть формула вида `(likesObject Keith BillM)`. В этом случае будет создан дочерний узел корня дерева – это будет просто константа `#$True`, так как предикат `likesObject` вы-

[3] Операция ASK (запрос) в Cyc представляет собой одну из двух базовых операций с базой знаний. Целью операции ASK является опрос базы знаний на предмет того, является или нет некоторая формула истинной. Вторая базовая операция, осуществляемая над онтологией Cyc, – это ASSERT (добавление). Операция ASSERT обычно используется для добавления в базу знаний новых суждений. При добавлении используется прямой логический вывод, в отличие от обратного вывода, используемого в операции ASK.

полняется на константах Keith и BillM. Предположим также, что в онтологии имеется формула вида:

```
(implies
  (possesses ?x ?y)
  (likesObject ?x ?y))
```

Тогда, у корня дерева поиска появится второй узел – формула (possesses ?x ?y). Процедур обратного вывода рекурсивно запустится на этом узле. Предположим, что в онтологии имеется формула

```
(implies
  (objectFoundInLocation ?x KeithsHouse)
  (possesses Keith ?x))
```

заключение которой совпадает с обрабатываемым узлом дерева. Тогда у узла формулы (possesses ?x ?y) появится дочерний узел с формулой (objectFoundInLocation ?x KeithsHouse), у которой второй параметр связан константой KeithsHouse, т.е. у узла имеется только одна свободная переменная. Это можно интерпретировать следующим образом: мы ищем вещи в доме Кейта, потому что, если какая-то вещь находится в доме Кейта, то Кейт будет владельцем этой вещи, а значит, она ему будет нравиться. Вопрос же о том, что нравится Кейту, есть частный случай проблемы, поставленной вопросом о выполнимости формулы (likesObject ?x ?y).

Предположим, что в онтологии также имеется правило

```
(implies
  (and
    (isa ?x Agent)
    (owns ?x ?y))
  (possesses ?x ?y)),
```

тогда в дерево будет добавлен новый узел:

```
(and
  (isa ?x Agent)
  (owns ?x ?y))
```

Развертывание дерева, очевидно в любом случае, закончится, когда в узлах не останется свободных переменных, независимо от того, найдутся ли в онтологии суждения, которые могут инициировать создание дочерних узлов данного узла.

Из рассмотренных примеров видно, что алгоритм логического вывода, инициированный операцией ASK, фактически, представляет собой поиск по дереву. Здесь остаются невыясненными следующие вопросы:

- Каким образом производится решение, какие узлы дерева будут созданы как дочерние узлы обрабатываемого узла?
- Как в онтологии находятся суждения, которые используются для создания новых узлов?

1.2. Логический вывод в Cyc

Три наиболее важных характеристики логического вывода:

1. Код системы логического поиска строится на основе модульного принципа. Взаимодействие между модулями должно быть тщательно продумано и отлажено.
2. Текущее состояние логического поиска может быть сохранено. Поэтому, поиск может быть продолжен снова даже в том случае, если в какой-то момент он был прекращен из-за недостатка ресурсов программной системы.
3. Логический поиск полностью параметризован, в качестве параметров могут выступать различные поисковые алгоритмы.

Обсудим чуть подробнее третий пункт. По умолчанию, в Cyc-10 производится эвристический поиск (о котором подробно поговорим ниже), но для некоторых целей необходим и полноценный поиск с построением полного дерева. Такое построение обычно называются *поиском в глубину* (depth search). Поиск в глубину необходим, например, в процедуре прямого логического вывода, для которой необходимо построение полного дерева. Но, обычно, приложения используют преимущества параметризованного поиска, проводя, в основном, эвристический поиск[4], но применяя для некоторых узлов другие методы.

Эвристики для выбора дочернего узла. В системе Cyc-10 используется большое число эвристических правил для выбора дочернего узла дерева. Большинство из них чисто синтаксические.

1. Среди возможных дочерних узлов данного узла дерева поиска выбирается узел с наи-

[4] Под эвристическим поиском здесь понимается применение специальных процедур, позволяющих на основе каких-то свойств формулы выбрать ее среди остальных кандидатов для построения следующего узла дерева вывода. Критерии, на основе которых осуществляется выбор той или иной формулы, называются *эвристиками* или *эвристическими правилами*.

меньшим числом литералов. Эта эвристика позволяет управлять скоростью поиска посредством выбора наименьшего поддерева.

2. Выбирается узел с наименьшим числом свободных переменных. Это также позволяет управлять скоростью поиска, двигаясь каждый раз к дереву с наименьшим количеством ветвей.

3. Редко выбираются узлы, которые уже присутствуют в дереве где-то наверху. Это позволяет избежать слишком глубокого спуска по дереву с использованием повторяющихся узлов.

4. Узлы без свободных переменных имеют высший приоритет. Если в формуле не осталось свободных переменных, то поиск с большой вероятностью может в недалеком будущем прийти к завершению.

5. Редко выбираются узлы, формулы которых содержат отрицания. База знаний Cyc состоит, в основном, из позитивных (т.е. не содержащих отрицания) суждений, поэтому вероятность найти решение среди позитивных суждений выше.

В поиске используются также семантические эвристические правила:

- отбрасываются узлы, которые могут быть частью цикла при унификации параметров формулы.

- отбрасываются узлы, которые кажутся наименее пригодными для поиска. Мера пригодности рассчитывается для каждого из литералов узла, в которых присутствует предикат или константа.

Весовые эвристики. Здесь рассматриваются правила, которые работают, оценивая каждый узел-кандидат и выдавая численную меру этой оценки. Оценки всех таких правил затем суммируются и для обработки выбирается узел с наименьшей суммой[5].

Эвристики для решения об обработке литерала в узле. Часто формула содержит более одного литерала. В примере выше встречался узел с формулой

```
(and
  (isa ?x Agent)
  (owns ?x ?y))
```

Если узел с этой формулой был выбран для обработки, то необходимо решить, какой литерал надо обрабатывать первым: искать элементы класса `Agent` или пары, удовлетворяющие предикату owns.

В Cyc-10 используется следующий подход. Литерал передается на оценку каждому эвристическому модулю (HL module) системы. Модуль возвращает численную меры стоимости обработки литерала (чем выше трудность обработки, тем выше цена). Эти меры суммируются и выбирается литерал с наименьшей ценой. Эта процедура похожа на рассмотренную выше процедуру выбора узла на основе весовых эвристик, только в данном случае выбирается литерал для обработки, а не дочерний узел.

Параметры поиска в Cyc-10. Состояние логического поиска сохраняется в специальной структуре, поля которой содержат методы, используемые для реализации специфической поисковой методики, и реализованные в виде функций. Перечислим эти функции:

- `too-deep-func : node depth-cut -> [boolean]`. Эта функция используется для оценки того, вышел или нет данный узел за пределы допустимой глубины дерева поиска.
- `add-node-func : node leaves -> leaves`. Функция используется для добавления новых узлов в дерево поиска.
- `no-leaves-p-func : leaves -> [boolean]`. Данная функция используется для проверки, имеет или нет данный узел дочерние узлы. Если таковых нет, то поиск в данной ветке дерева заканчивается.
- `next-node-func : leaves -> node, leaves`. Функция позволяет выбирать узел для обработки из множества возможных дочерних узлов.
- `goal-p-func : node -> [boolean]`. Эта функция используется для определения, удовлетворяет или нет данный узел цели поиска.
- `add-goal-func : node goals -> goals`. Если узел является целевым узлом поиска, то данная функция добавляет его во множество целевых узлов.
- `options-func : node -> [list of options]`. Данная функция предоставляет список опций, использующихся для определения следующего узла для обработки.

[5] В оригинальной документации совокупность всех таких весовых правил сравнивается с хором, участниками которого выступают сами правила. Каждому оцениваемому узлу «поется песня» и, чем громче звучат «голоса» правил при оценке данного узла, тем громче песня. Выбирается узел, «песня» которого была наименее «громкой».

• `expand-func : node option -> [list of nodes]`. Эта функция используется для задания списка дочерних узлов в дереве поиска, т.е. тех, которые будут обрабатываться процедурой поиска далее.

Данную структуру используют все процедуры поиска: поиск в глубину и эвристический поиск. Для каждой конкретной процедуры поиска поля структуры заполняются конкретными функциями и процедура поиска параметризуется.

Узлы поиска. В Cyc-10, каждый узел поиска представлен специальной структурой со следующими полями:

• `search` содержит информацию о том, какая процедура поиска инициировала создание данного узла;

• `parent` содержит ссылку на родительский узел;

• `children` представляет собой список ссылок на дочерние узлы;

• `depth`"— это глубина узла в дереве поиска;

• `options` содержит список способов, которыми из данного узла можно породить дочерние;

• `state` представляет собой структуру данных, которая содержит описание семантики узла.

Данные о семантике узла содержат следующую информацию:

• `formula`: содержит формулу, которая инициировала порождение данного узла;

• `inference supports`: суждения и эвристические модули, которые были использованы для порождения данного узла из родительского;

• `variable bindings`: соответствие между переменными родительского узла данного узла и элементами, с которыми данные переменные были связаны, чтобы породить данный узел.

1.3. ASK и направления

Как уже говорилось выше, процедуры ASK и ASSERT используют направления (directions) для управления доступом к суждениям во время логического вывода. Направления могут иметь два значения: прямое (forward) и обратное (backward). Суждения с прямыми направлениями используются в процедурах ASSERT, а с обратными – в процедурах ASK.

Обратные базовые атомарные формулы не задействуются во время добавления суждений в онтологию Cyc, т.е. в процедурах ASSERT. Прямой вывод использует только правила и базовые атомарные формулы с прямым направлением. Большинство интерфейсов, используемых для добавления суждений в базу знаний Cyc, по умолчанию полагают, что каждое правило (формула импликации) имеет обратное направление, а базовая атомарная формула – прямое.

Эквивалентность. Эквивалентность используется в процедуре унификации. В онтологии Cyc сравнение элементов основано на положении об уникальности их имен: элементы с разными именами считаются различными, а с совпадающими – эквивалентными. Кроме того, эквивалентными считаются те элементы, о равенстве которых явно сказано в онтологии. Равенство элементов задается с помощью предиката `equals`, например, `(equals Fred Joe)` задает равенство констант `Fred` и `Joe`[6]. Во время унификации не производится никаких процедур вывода, относящихся к проверке эквивалентности элементов.

2. Онтология системы Cyc

Как уже говорилось выше, онтология системы Cyc имеет довольно сложную и разветвленную структуру. В этом разделе мы рассмотрим эту структуру более детально.

Авторы системы Cyc предпочитают выводить структуру онтологии Cyc в виде пирамиды, изображенной на Рис. 1.

Если взглянуть упрощенно, то эту пирамиду можно представлять как изображение дерева таксономии классов (коллекций) базы знаний системы Cyc. Конечно, система Cyc позволяет, чтобы у класса было более одного суперкласса, поэтому на самом деле структура онтологии системы это не таксономия (дерево), но направленный ациклический граф, т.е. более сложная структура. Для упрощения изложения мы, следуя авторам системы, изобразившим структуру онтологии в виде пирамиды, будем считать структуру таксономией. Авторы систе-

[6] Таким образом, отношение эквивалентности между именами можно сравнить с отношением синонимии на словах русского языка.

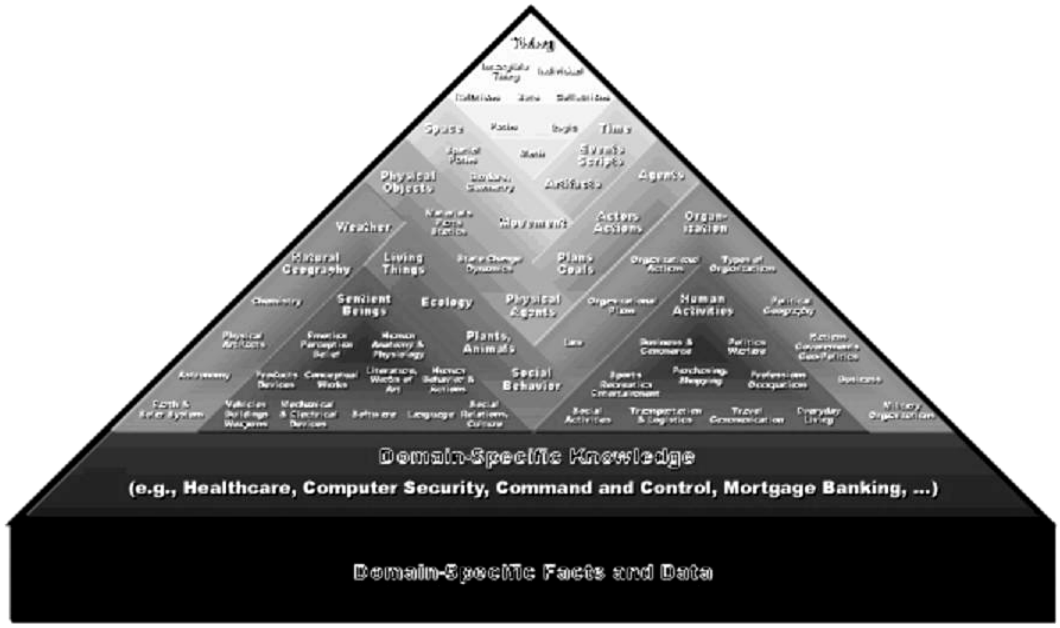

Рис. 1. Онтология системы Cyc в виде пирамиды

мы предоставили специальное приложение, делающее эту пирамиду интерактивной. Приложение доступно на странице [3]. Пользователю достаточно кликнуть левой кнопкой мыши на рисунке пирамиды онтологии и откроется новая страница браузера с интерактивным приложением.

Внимательный читатель уже наверное заметил, что на пирамиде онтологии, изображенной на Рис. 1, названия элементов отделены друг от друга секторами. Если пользователь кликнет на такой сектор левой кнопкой мыши, то в окне пирамиды будет выведена информация об элементе онтологии. Окно приложения разделено на три части. В центральной части находится интерактивный рисунок пирамиды онтологии. В левой части выводится имя элемента онтологии, в нижней - описание этого элемента. Элементы онтологии разделены на уровни, каждый из которых выделен своим цветом[7]. По мере обретения конкретики, т.е. спуска (подъема) по дереву таксономии, цвет уровней меняет от светло-серого к более темным тонам. Специфические для описаний конкретных знаний понятия выделены темно-синим цветом. Ниже мы рассмотрим большую часть пирамиды подробно.

Более привычный взгляд на онтологию дается на Рис. 2, где онтология изображена в виде направленного (хотя направления ребер не указаны на рисунке, подразумевается, что у каждого ребра имеется направление, от суперкласса к дочернему классу) ациклического графа.

2.1. Светло-серый уровень

На этом уровне располагаются самые абстрактные понятия, которые еще не имеют пространственной и временной протяженности. Рассмотрим их.

Понятие *Объект* (Thing) представляет собой корень дерева иерархии онтологии системы Cyc. Каждый элемент онтологии имеет класс «Объект» в качестве суперкласса. Поэтому, можно сказать, что каждый элемент онтологии системы Cyc есть Объект. С точки зрения реализации, понятие Объект представляет собой структуру, содержащую информацию, позволяющую:

- сравнивать элементы дерева иерархии друг с другом;
- обращаться к элементам онтологии непосредственно используя программный интерфейс.

[7] К сожалению, здесь предоставлено изображение пирамиды онтологии Cyc в оттенках серого цвета, поэтому описания цветов ниже выглядят достаточно условными. Заинтересованный читатель может загрузить приложение для работы с пирамидой онтологии со страницы по адресу [3].

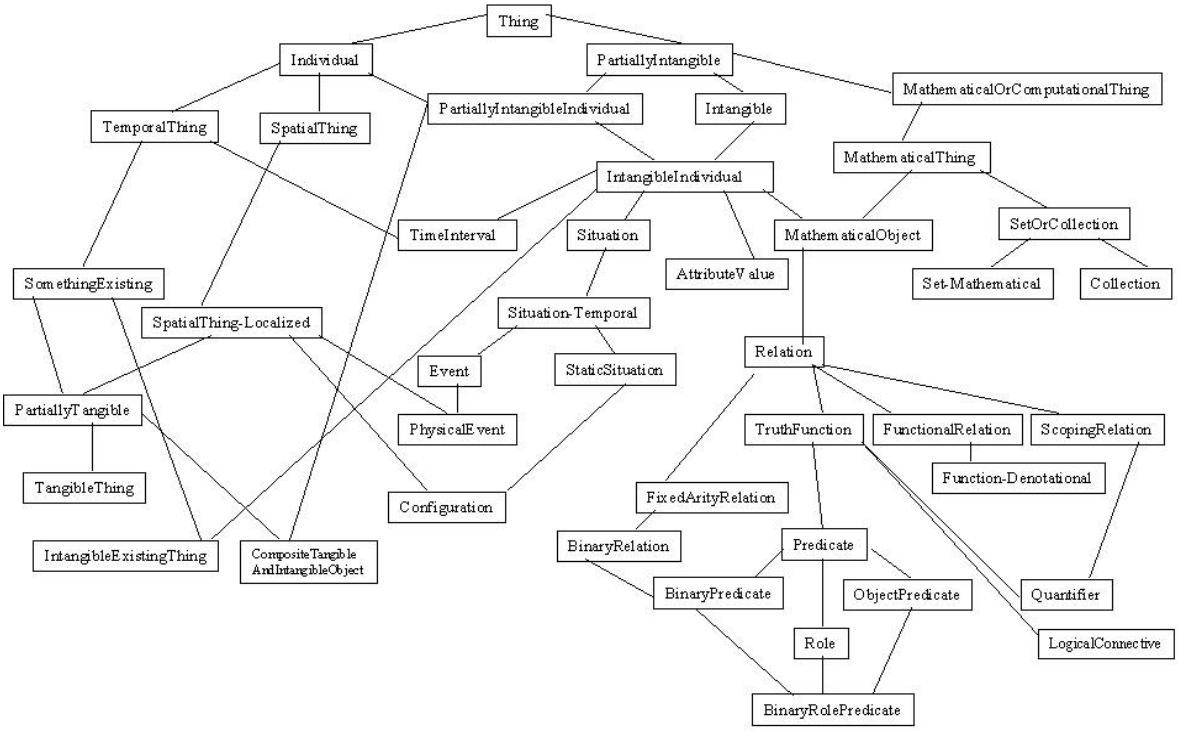

Рис. 2. Онтология системы Cyc в виде ациклического графа

Класс *Нематериальный Объект* (Intangible Thing) является прямым потомком класса Объект. Нематериальные объекты не имеют материального воплощения, они не сделаны, не закодированы и т.п. Нематериальные объекты – это события, такие как поход в горы или поедание завтрака. Идеи также являются нематериальными объектами. С точки зрения реализации весьма полезно отличать нематериальные объекты от остальных. Это позволяет избежать различных ошибок, например, задания вопроса пользователю о том, какого цвета будет собрание в следующий четверг.

Другой непосредственный потомок класса Объект – это класс *Индивид* (Individual). В Индивидах главное, что они отличаются от Множеств и Коллекций. Индивидами являются события, физические объекты, числа и отношения. Конкретный автомобиль Тойота Королла – это Индивид, но понятие «четырехдверный седан» – уже коллекция. Эта коллекция, в отличие от конкретного автомобиля, абстрактна. Она не имеет протяжения, массы и максимальной скорости, но имеет элементы, подколлекции и надколлекции.

Класс *Отношение* (Relation) является потомком класса Нематериальный Объект. Отношения подобны клею, который соединяет разрозненные понятия в суждения. Например, понятия «Иосиф Кобзон» и «певец» можно связать предложением «Иосиф Кобзон это певец». Связанные отношениями понятия представляют собой *суждения*. Также отношения можно использовать при работе с внешними источниками знаний, такими как базы данных компаний или Web-страницы. Как читатель могу убедиться выше, система Cyc построена на принципе: «чем больше суждений, тем лучше». Создатель системы Дуглас Ленат известен своим высказыванием: «интеллект – это десять миллионов суждений». Язык CycL представляет собой язык исчисления логики предикатов первого порядка, но содержит также и много возможностей, относящихся к языку второго порядка. В частности, отношения также можно связывать друг с другом другими отношениями, что, помимо всего прочего, позволяет системе автоматически обучаться. Когда в систему добавляется факт о том, что автомобиль X имеет подушку безопасности, то система Cyc уже знает, что автомобиль X содержит подушку безопасности, а также все ее составные элементы внутри автомобиля. Это происходит в результате уже имеющихся связей между отношениями *Содержать*, *Быть Частью* и *Внутри*.

Класс *Множество* (Set) представляет собой прямой аналог понятия математического множества. В отличие от *Коллекции*, элементы множества не обязательно связаны между собой какими-то общими свойствами. Единственное общее свойство таких элементов – это то, что они принадлежат данному множеству.

Класс *Коллекция* (Collection) представляет собой прямой аналог рассматривавшегося нами выше понятия типа. Все элементы коллекции имеют нечто общее и это общее выражается посредством одних и тех же атрибутов. Как уже говорилось выше, главным отличием Коллекций и Множеств от Индивидов является то, что последние не могут иметь элементов. Коллекции отличаются от Множеств еще и тем, что могут иметь несколько «воплощений» одновременно, т.е. могут иметься два экземпляра класса Коллекция, имеющих одни и те же элементы, но они все равно будут рассматриваться, как различные. Множества считаются идентичными, если они содержат одни и те же элементы. Вообще, хорошей коллекцией считается такая, которую трудно или вообще невозможно определить простым перечислением элементов. Например, коллекция всех белых автомобилей не очень хороший пример для создания, так как это понятие легко определить посредством других элементов системы Cyc и, следовательно, само понятие не очень информативно. С другой стороны, коллекция всех «белых воротничков»[8] очень трудно определима простым перечислением элементов, поэтому об этом понятии можно многое сказать.

Класс *Путь* (Path) представляет собой непересекающийся путь или цикл. Пути могут представлять пространственно-временные абстракции (ребра между узлами в графах), нематериальные абстракции расположений (линии географической широты) или вполне конкретные линии (разделительная полоса на автомобильной трассе). Иначе говоря, путь может быть всем, что используется как путь в данной системе. Например, апельсин, вообще говоря, это не путь. Но он может стать путем для муравья в системе управления путями муравьев посредством апельсинов. Заметим также, что Путь не обязательно является индивидом, так как ничто не мешает использовать множества или коллекции в качестве путей в специфических системах. Основное разделение путей – это так называемый Простой Путь (Simple Path), представляющий незамкнутые кривые, и Циклический Путь (Cyclic Path), представляющий все замкнутые контуры.

Хотя понятие *Логика* (Logic) и не является классом (collection) в онтологии Cyc, это понятие является в ней фундаментальным. На логических заключениях строятся элемента базы знаний Cyc – так называемые суждения. Вероятно поэтому авторы поместили описание связанных с логикой понятий на светло-серый уровень. Эти понятия мы уже подробно рассматривали выше, поэтому нет необходимости повторять описание здесь. Скажем кратко: в Cyc имеет две логические константы `Истина` и `Ложь`. Суждения строятся из констант и других суждений с помощью логических связок – операторов конъюнкции, дизъюнкции, отрицания и импликации. Если суждение выполняется на каком-то наборе своих констант, то на нем это суждение считается истинным, в противном случае – ложным. Кроме того, в Cyc также реализована процедура логического вывода, в деталях описанная выше.

Математические понятия (Mathematical or computational things), не связанные с понятиями нижних уровней (пространственными, временными или материальными), также располагаются на этом уровне. Среди таких понятий находятся числа, множества, алгоритмы, абстрактные символьные строки и т.п.

2.2. Умеренно-серый уровень

На этом уровне располагаются понятия, связанные с пространственными и временными соотношениями, а также с понятием материи. Некоторые понятия заданы в онтологии Cyc в виде классов (collections), другие представляют собой целые группы понятий, объединенных сходными признаками. Рассмотрим их.

Класс *Пространственный Объект* (Spatial Thing) содержит в себе все понятия, связанные с протяженностью и местоположением. Когда вводятся понятия этого класса, необходимо обращать внимание на две вещи. Во-первых, объект Пространственный Класс может быть частично материальным (например, Ленинградская область) или вовсе нематериальным (окруж-

[8] Так называют сословие офисных работников.

ность на плоскости). Во-вторых, совсем необязательно определять местоположение этого объекта относительно других пространственных объектов, хотя это и можно делать. Дело в том, что не всякий пространственный объект имеет местоположение. Например, отрезок на плоскости как геометрическое понятие не имеет местоположения.

Класс *Пространственный Путь* (Spatial Path) представляет пути в пространстве, связывающие пространственные объекты. В качестве примера пространственных путей можно привести дорогу, коридор, проволоку, кровяные сосуды и нервы. Следует иметь ввиду, что абстрактные пути, такие как генеалогическое древо, не являются Пространственными Путями.

Класс *Рамка* (Border) содержит такие понятия, как линии, отрезки, плоскости или пространственные области, которые служат границами (формируют рамки) между двумя Пространственными объектами.

Класс *Перемещение* (Movement) представляет собой событие, заключающееся в пространственном перемещении. Перемещение может происходить вдоль некоторого пути, т.е. объект, в этом случае, преодолевает некоторую дистанцию, или быть поворотом объекта вдоль некоторой оси. Перемещение может быть непрерывным или периодическим. В качестве примеров перемещений можно привести полет на самолете (с промежуточными посадками), поедание пищи или электрический ток.

Класс *Артефакт* (Artifact) представляет собой неодушевленные вещи, произведенные агентами специально для каких-то целей. Для создания артефакта не обязательно создавать для него новый материал, агент может создать артефакт просто переделав другие вещи. В качестве примеров артефактов можно привести флейту, сделанную из ветвей дерева, или монету, сделанную из металла путем тиснения. Вдобавок к человеческим артефактам (инструментам, зданиями, документами, линиям электропередач) к таковым можно отнести и артефакты, сделанные животными (гнезда, термитники, бобровые плотины).

В системе Cyc используется более 37 тысяч различных типов *Событий* (Events). События используются для описания того, что случилось в мире. Общие события включают в себя пение, делание покупок, мышление (рассматриваемое как событие вида «мыслить что-то») и т.п. Индивидуальным событием может быть, например, «полет Гагарина в космос» или «начало Великой Отечественной Войны».

Сценарий (Script) представляет собой описание сложного события, состоящего из подсобытий, связанных между собой временными отношениями. Приложения могут использовать сценарии для распознавания того, что данное событие является частью более сложного события, описываемого данным сценарием. Сценарии также можно использовать для планирования и для чтения.

Класс *Временный Объект* (Temporal Thing) представляет объекты, имеющие временную протяженность или расположение во времени. К объектам этого типа применим вопрос «когда?». Таким образом, к временным объектам можно отнести множество вещей, например, события, физические объекты, соглашения, временные интервалы и т.д. Объекты, не имеющие свойства временности, такие как математические множества, атрибуты или числа, очевидно, не являются Временными Объектами.

2.3. Темно-серый уровень

На этом уровне расположены объекты, либо имеющие материальное воплощение, либо взаимодействующие с такими объектами. Рассмотрим элементы этого уровня подробнее.

Класс *Физический Объект* (Physical Object) представляет все объекты, которые состоят из материи и существуют во времени (т.е. к таким объектам можно применять временные категории: когда появился, как изменился и т.п.). Такие объекты могут содержать, а могут и не содержать, нематериальные элементы. Например, конкретный экземпляр книги представляет собой физический объект, так как сделан из материи и имеет пространственную и временную протяженность (размеры и год издания). Но, этот объект также имеет содержание, т.е. смысл текста, напечатанного на страницах этой книги. Содержание, очевидно, является нематериальным объектом.

Материалы (Materials) используются для выделения из неодушевленного объекта другого неодушевленного объекта, который был использован для создания первого объекта. Материал можно рассматривать, как связь между

индивидуальным объектом, который частично состоит из другого объекта, и этим объектом, который, в свою очередь, более или менее равномерно распределен в первом объекте. Например, ложка сахара, растворенная в стакане чая, становится материалом для этого напитка.

Части (Parts) рассматриваются как связи индивидами и их составляющими. Составляющие здесь понимаются в широком контексте. Это могут быть, например, пространственные составляющие, временные составляющие, понятийные составляющие, члены группы и т.п. Части используются, чтобы сказать, что некоторый объект является, в некотором смысле, частью целого. В онтологии Сус наиболее часто используются такие части как физические составляющие, подсобытия, временные отрезки и члены групп.

Класс *Статическая Ситуация* (Static Situation) является подклассом класса *Временная Ситуация* (Temporal Situation). Каждая статическая ситуация представляет собой состояние некоторой связи между двумя и более объектами, статично существующими на некотором временном промежутке. Статические ситуации всегда имеют временную протяженность и часто имеют пространственную протяженность и материальное воплощение. Например, рассмотрим ситуацию: «Президент Дмитрий Анатольевич Медведев сидит на стуле 22 августа 2009 года». Здесь есть два объекта: «Дмитрий Анатольевич Медведев» и «стул», а также множество связей между этими объектами: «спинка стула поддерживает спину Президента», «сидение стула реагирует на вес Президента» и т.д.

Изменения Физических Состояний (Physical State Changes) и *Трансформирующие События* (Metamorphosis Events) и другие события подобного рода можно рассматривать, как изменения, выражающиеся в том, что нечто перестает существовать, а другое нечто начинает.

Класс *Физический Агент* (Physical Agent) представляет объекты, которые существуют и, по меньшей мере, имеют материальное воплощение. Каждый Физический Агент имеет желания и намерения, а также возможности эти желания и намерения осуществить. Агенты могут быть индивидуальными или состоять из нескольких агентов, действующих вместе.

План (plan) – это последовательность шагов, которая, будучи исполнена в должном временном порядке, приведет к достижению некоторой цели или к осуществлению некоторой задачи.

Цели (goals) в системе Сус представляют собой характеристики отношений между агентами и формулами, описывающими состояния вещей, для достижения которых эти агенты намереваются предпринять определенные шаги. Цель также можно рассматривать как атрибут отношения между агентом и статической ситуацией, описываемой некоторой Сус-формулой.

Актеры (Actors) представляют собой вид отношения между событием и любым существующим объектом, который каким-либо образом вовлечен в это событие. Иначе говоря, когда говорят, что кто-то или что-то является действующим лицом (актером) данного события, подразумевается, что этот кто-то или что-то как-то вовлекается в данное событие в течении его протекания. При этом следует заметить, что актер не всегда играет активную роль в событии. Примером может служить «чеховское ружье», которое, если уж висит на стене на протяжении действия, то обязательно должно выстрелить в конце спектакля.

Действия (actions) представляют собой события, которые осуществляются некоторым действующим лицом. Действия – это события, в которых один или более актеров производят некоторые изменения в состоянии мира, обычно выражающиеся в некоторой трате усилий. При этом необязательно, чтобы в результате произошли какие-то материальные изменения (перемещение объектов или их разрушение). Например, действие может заключаться в изменении счета клиента в банке. Также, актер не обязательно должен быть одушевленным объектом. Например, авария газопровода на Озерной улице в Москве в 2009 году, когда огнем горящего газа было повреждено множество автомобилей (здесь актером является газовый факел).

Класс *Агент* (Agent) представляет объекты, которые имеют желания и намерения, а также возможности эти желания и намерения осуществить. Агенты могут быть индивидуальными или состоять из нескольких агентов, действующих вместе.

2.4. Фиолетовый уровень

На этом уровне находятся знания, которые у человека обычно рассматриваются как его знания об окружающем мире, т.е. о различных его аспектах. Среди таких знаний есть знания о солнечной системе, ее устройстве, количестве планет и т.д. Есть знания о дальнем космосе, т.е. о звездах, галактиках и других подобных объектах, эти знания объединены в раздел «астрономия». Человек имеет понятие об окружающих его предметах, поэтому в Cyc есть специальный класс *Физический Артефакт* (Physical Artifact), которому принадлежит великое множество объектов нашего мира. Знания о структуре веществ объединены в раздел «Химия». В этом разделе есть подразделы вида «Аналитическая Химия», «Вычислительная Химия», «Электрохимия» и т.д. Знания о континентах, морях и океанах объединены в раздел под названием «Естественная География», а знания о геополитическом устройстве мира – в раздел «Политичесая География». Элементы политической карты мира – это нации, правительства и геополитические объекты (группы лиц, контролирующих тот или иной регион). Имеются также знания о различного рода организациях, их типах, а также о коммерческих и военных организациях. Система Cyc также знает о живых организмах в общем (так называемые Living Things), а также о конкретных растениях и животных. Имеются знания об экологии, социальном поведении и законе.

2.5. Синий, умеренно-синий и темно-синий уровни

На синем уровне также как и на фиолетовом, находятся знания о различных сторонах окружающего мира, но уже более конкретные. Например, один из разделов посвящен описанию автомобилей, зданий и оружия, т.е. содержит описание знаний о конкретной стороне человеческой жизни. Другой подобный раздел – это знания о промышленных продуктах и о различных механизмах. На этом уровне также находятся знания о языках, литературе и других областях искусства. На синем уровне также находятся знания, связанные с различными областями общественной деятельности. Это знания о структуре общества, а также конкретные знания об элементах этой структуры. Например, имеются знания о спорте, продаже и торговле, путешествиях, профессиях, политике, войнах и т.п.

На умеренно-синем уровне находятся знания о специфических областях человеческой деятельности. К таким знаниям можно отнести здравоохранение, компьютерную безопасность, знания о некоторых типах химических реакций и т.п.

На темно-синем уровне находятся конкретные факты и знания. Это, обычно, те знания, которые записываются в таблицах баз данных. Но, в отличие от баз данных, конкретные знания в Cyc соединены с понятиями верхнего уровня. Например, в системе Cyc имеются знания о нескольких сотнях известных музыкантов. Система Cyc знает примерно 85 вещей из тех, которые знают музыканты. Поэтому, например, «Владимир Спиваков» это не просто запись в таблице, Cyc знает, что Спиваков играет на скрипке, руководит ансамблем «Виртуозы Москвы» и т.д. Вообще, в системе Cyc имеется возможность присоединять существующие базы данных к системе. Идентифицируя столбцы таблицы с понятиями онтологии Cyc, можно встроить базу данных в базу знаний системы Cyc. Тогда записи таблиц будут интерпретироваться системой Cyc как суждения и будет иметься возможность задавать к базе данных вопросы, делать выводы на основе фактов, расположенных в этих базах данных и т.п.

Заключение

В работе рассмотрены различные аспекты функционирования программной системы Cyc, которая предоставляет возможности для построения библиотек онтологий, а также функциональность по осуществлению логического вывода на основе знаний, описанных в библиотеках онтологий системы. Ввиду того, что на момент написания статьи, литература на русском языке по данной системе практически отсутствовала, автор посчитал полезным познакомить русскоязычного читателя с содержанием данной программной системы. Таким образом, основная цель данной работы состоит в том, чтобы предоставить обзор системы Cyc на русском языке.

Как читатель мог убедиться в процессе чтения статьи, система Cyc представляет собой доста-

точно объемный проект, на построение которого были затрачены большие человеческие ресурсы. Описание только основных свойств системы занимает значительный объем текста. По этой причине, некоторые вопросы, которые требовали более обстоятельного обсуждения, остались в данной работе не покрытыми.

Так, практически не было уделено внимания системе обработки естественных языков под названием Cyc-NL. Эта система состоит из трех компонентов: словаря, синтаксического анализатора и семантического интерпретатора. На данный момент система Cyc-NL в состоянии обрабатывать синтаксически сложные и неоднозначные конструкции, включающие в себя отрицания, модальные и вложенные кванторы. Разработчики системы Cyc-NL разрабатывают программный интерфейс, который позволит получить доступ к системе Cyc-NL из других систем. Для получения детальной информации о системе Cyc-NL необходимо обратиться к [4].

Также не было уделено внимания описанию проекта OpenCyc [2], представляющего собой открытую систему, доющую возможности для осуществления логического вывода на основе фактов, содержащихся в онтологиях системы Cyc, а также удобные инструменты для добавления новых знаний в систему.

Автор надеется, что данная работа послужит читателю хорошим начальным пособием по системе Cyc.

Литература

1. Сайт компании Cycorp. // http://www.cyc.com.
2. Сайт проекта OpenCyc. // http://opencyc.org.
3. Страница описания онтологии Cyc. // http://www.cyc.com/cyc/technology/whatiscyc_dir /whatdoescycknow.
4. Страница Cyc-NL. // http://www.cyc.com/cyc/technology/whatiscyc_dir/ cycrandd/ areasofrandd_dir/nlu.
5. Чень Ч, Ли Р. Математическая логика и автоматическое доказательство теорем. // М.: Наука, 1983.

Лапшин Владимир Анатольевич. Ведущий разработчик ООО «Инфовотч». Окончил Киргизский Государственный национальный университет в 1996 году. Кандидат физико-математических наук. 15 печатных работ. Область научных интересов: теория формальных языков, онтологии и их применение. E-mail: mefrill@yandex.ru.