Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Казанский (Приволжский) федеральный университет»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ
И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА СИСТЕМНОГО АНАЛИЗА И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Михайлов В.Ю.

ДЕСКРИПТИВНАЯ ЛОГИКА КАК ОСНОВА СОЗДАНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

КАЗАНЬ - 2025

УДК 51-74

Учебное пособие публикуется по решению
учебно-методической комиссии Института вычислительной математики
и информационных технологий КФУ
Протокол № 7 от 21 марта 2025 г.

заседания кафедры системного анализа
и информационных технологий
Протокол № 6 от 14 марта 2025 г.

Автор-составитель
к.ф.-м.н. Михайлов В.Ю.

Рецензент
к.ф.-м.н. Пшеничный П.В.

Дескриптивная логика как основа создания интеллектуальных систем**:** Учебное пособие / Михайлов В.Ю.– Казань: Казанский университет, 2025. – 50 с.

Учебное пособие предназначено для студентов, изучающих курсы "Математическая логика и теория алгоритмов" и "Интеллектуальные системы" , а также для преподавателей, ведущих лекционные и практические занятия по данным курсам.

СОДЕРЖАНИЕ

Введение

Дескрипционные или описательные логики (Description Logics, DL) относятся к языкам представления знаний, позволяющим описывать понятия предметной области в формализованном виде. Дескрипционные логики служат для:

- формального определения понятий (классов);
- определения отношений между понятиями;
- организации иерархий (подкласс – суперкласс);
- определения свойств и атрибутов понятий и ограничений на их значения.

Дескрипционные логики нашли широкое применение в различных аналитических и поисковых системах. Достоинство дескрипционных логик заключается в том, что с их помощью эксперты описывают термины и определения понятий предметной области, формируют базу знаний предметной области, а затем специальная программа reasoner, основанная на алгоритмах поиска логического вывода в DL, осуществляет контроль корректности определений и реализацию различных запросов к базе знаний.

Выделяют следующие **этапы развития DL**

Этап 0 (1965–1980): Пред-DL период. В это время были предложены семантические сети и фреймы как специальные подходы к структурированному представлению знаний.

Этап 1 (1980–1990): Разработка систем. Этот этап сосредоточен на реализации систем KL-ONE, K-REP, KRYPTON, BACK и LOOM.

Этап 2 (1990–1995): Табличные алгоритмы. На этом этапе появились табличные алгоритмы, которые работали с полными по булевой логике DL и являлись полными даже для некоторых выразительных логик.

Этап 3 (1995–2000): Поддержка выразительных DL. На этой стадии были разработаны эффективные процедуры вывода для очень выразительных DL, основанные на табличных методах.

Этап 4 (с 2000-х годов — по настоящее время): Индустриальные DL-системы. Результаты предыдущих этапов были использованы для создания промышленных DL- систем, поддерживающих очень выразительные DL и ориентированных на применения в таких важнейших областях:

- Семантическая паутина (Semantic Web)
- Медицинская информатика
- Биоинформатика

1. Дескриптивные логики и базы знаний

Дескриптивные логики являются общим классом логик специально спроектированных для представления знаний о предметных областях. Они определяют формальный язык для определения *понятий* и отношений (называемых *ролями*). На этом языке можно выражать утверждения о свойствах индивидов предметной области и формулировать различные запросы, включая выполнимость (*satisfiability*) понятий и включение (*subsumption*) понятий друг в друга.

Элементами этого языка являются: *концепт* - понятие, которое представляет класс, категорию или сущность; *роль,* которая является бинарным отношением между концептами. *Конструкторы (операции) языка* позволяют определять сложные концепты через более простые и роли.

Системы, построенные с помощью дескриптивной логики, используются для создания *Баз Знаний* (БЗ), которые представляются в виде пары *Tbox* и *Abox.*

Tbox (*terminological knowledge*) это набор утверждений, описывающих набор классов, их свойства и отношения между ними (интенсиональные знания);

Abox (*assertional knowledge*) представляет собой реализацию схемы классов в виде набора экземпляров, содержащих утверждения об экземплярах понятий (экстенсиональные знания).

По существу *Tbox* является моделью того, что *должно* быть истинным, а *Abox* является моделью того, что в *настоящее время* является истинным.

В DL любой запрос сводится к определению выполнимости (*satisfiability*) БЗ (т. е. ее логической согласованности). То есть для ответа на запрос *Q*, обычно нужно доказать, что «not *Q*» является невыполнимым (т. е. текущая БЗ не может предполагать not *Q, противоречит* not *Q*) , что в этом случае приводит к заключению, что *Q* должно быть истинным. Понятно, что этот метод может быть менее эффективен, чем более простые. Преимуществом использования этого универсального метода вывода является то, что он может использоваться даже в тех случаях, когда БЗ
является не полностью определенной.

2. Логика ALC

ALC (Attributive Language with Complement) является одной из базовых DL, на основе которой строятся многие другие DL для работы с базами знаний и онтологиями в различных предметных областях. Собственно ALC – это не одна логика, а язык и правила построения логик, с помощью которых создано целое семейство логик, где каждая отличается набором множеств атомарных концептов и ролей.

2.1. Синтаксис и семантика логики ALC

Синтаксис любой дескрипционной логики задает правила в виде допустимых конструкций составных концептов.

Чтобы сформулировать синтаксис какой-либо DL, необходимо задать непустые (и обычно конечные) множества символов, так называемых атомарных концептов и атомарных ролей, из которых будут строиться выражения языка данной логики. DL характеризуется набором конструкторов и индуктивным правилом, с помощью которого составные концепты данной логики строятся из атомарных концептов и атомарных ролей, используя эти конструкты.

Типичными конструктами для построения составных концептов являются:

• пересечение (или конъюнкция) концептов, обозначается как $C \sqcap D$;

• объединение (или дизъюнкция) концептов, обозначается как $C \sqcup D$;
• дополнение (или отрицание) концепта, обозначается как $\neg C$;
• ограничение на значения роли (или ограничение квантором всеобщности), обозначается как $\forall R.C$);
• экзистенциальное ограничение (или ограничение квантором существования), обозначается как $\exists R.C$;
• численные ограничения на значения роли, например: $\leq n\ R$, $\geq n\ R.C$, задающие количество R–последователей, удовлетворяющих отношению R, и другие.
Существуют дескрипционные логики, в которых имеются также составные роли, строящиеся из простых ролей с помощью операций: инверсии, пересечения, объединения, дополнения, композиции ролей, транзитивного замыкания и других.

Синтаксис *ALC*. Множество концептов логики *ALC* задается следующим индуктивным правилом:
• всякий атомарный концепт является концептом;
• выражения $\top$ и $\bot$ являются концептами (истина и ложь);
• если C есть концепт, то его дополнение $\neg$ C является концептом;
• если C и D есть концепты, то их пересечение $C \sqcap D$ и объединение $C \sqcup D$ являются концептами;
• если C есть концепт, а R есть роль, то выражения $\forall R.C$ и $\exists R.C$ являются концептами, задающими общее ограничение на значения.

Семантика логики *ALC* – это интерпретация $I = (\Delta^I, \cdot^I)$, где Δ^I — непустое множество (область интерпретации), а интерпретирующая функция $\cdot^I$ ставит в соответствие каждому концепту все удовлетворяющие ему экземпляры (индивиды) по следующим правилам:
- $A^I \subseteq \Delta^I$, для каждого атомарного концепта A;
- $r^I \subseteq \Delta^I \times \Delta^I$, для каждой роли r; если r(a,b) справедливо, то b называют *r-последователем* a;
- $\top^I = \Delta^I$;
- $\bot$ интерпретируется как пустое множество: $\bot^I = \varnothing$
• дополнение концепта интерпретируется как дополнение множества: $(\neg C)^I = \Delta^I \setminus C^I$;

- пересечение концептов интерпретируется как пересечение множеств: $(C \sqcap D)^I = C^I \cap D^I$;
- объединение концептов интерпретируется как объединение множеств: $(C \sqcup D)^I = C^I \cup D^I$;

• выражение $\forall\, r.C$ интерпретируется как множество тех индивидов, у которых все *r*-последователи принадлежат интерпретации концепта *C*. Формально:

$(\forall r.C)^I = \{x \in \Delta^I \mid$ для всех $y \in \Delta^I$, если $(x, y) \in r^I$, то $y \in C^I\}$

• выражение $\exists\, r.C$ интерпретируется как множество тех индивидов, у которых имеется *r*-последователь, принадлежащий интерпретации концепта *C*. Формально:

$(\exists r.C)^I = \{x \in \Delta^I \mid$ существует $y \in \Delta^I$, такое что $(x, y) \in r^I$ и $y \in C^I\}$

Упражнение (для самопроверки).

Пусть $I = (\Delta^I,)$, где

$\Delta^I = \{a, b, c, d, e, f\}$;

$Person^I = \{a, b, c, d, f\}$ $Female^I = \{a, b, c, e\}$;

$hasChild^I = \{(a, b), (b, c), (d, e), (f, f)\}$.

Вычислить:

- $(Person \sqcap Female)^I$
- $(Person \sqcap \exists hasChild.Person)^I$
- $(Person \sqcap \exists hasChild.(Person \sqcap Female)^I$
- $(Person \sqcap \exists hasChild.Person \sqcap Female)^I$
- $(Person \sqcap \exists hasChild.\top)^I$
- $(Person \sqcap \exists hasChild.\exists hasChild.\top)^I$

Определения

- Общее включение концептов (GCI) имеет вид $C \sqsubseteq D$, где C и D — концепты ALC.
- **TBox -** это конечное множество GCI, которые будут называться аксиомами;
- Интерпретация I является моделью GCI $C \sqsubseteq D$, если $C^I \subseteq D^I$;
- I является моделью TBox T, если она является моделью каждого GCI в T.
- Обозначение $C \equiv D := C \sqsubseteq D$ и $D \sqsubseteq C$;

- Аксиома вида A ≡ C, где A – имя концепта, называется определением A;
- TBox T называется *определяющим,* если он содержит только определения с дополнительными ограничениями:

(i) T содержит не более одного определения для каждого имени концепта и

(ii) T ацикличен, то есть определения любого концепта A в T не ссылаются (прямо или косвенно) на A самого.

- Утверждающие аксиомы имеют формы:

(i) Аксиома x : C — индивид x принадлежит концепту C;

(ii) Аксиома (x, y) : r — пара индивидов принадлежит роли r;

- ABox — конечное множество утверждающих аксиом;
- Интерпретация I является моделью аксиомы x : C, если $x^I \in C^I$, и моделью аксиомы (x, y) : r, если $(x^I,y^I) \in r^I$;
- I является моделью ABox A, если она является моделью всех аксиом в A.

Еще ряд важных определений

- **База знаний (БЗ)** — это пара (T, A), где T — TBox, а A — ABox.
- Интерпретация I является моделью базы знаний K = (T, A), если I является моделью T и моделью A.
- Выражение I ⊨ K (соответственно, I ⊨ T, I ⊨ A, I ⊨ a), будет обозначить, что I является моделью базы знаний K (соответственно, TBox T, ABox A, аксиомы a).

Пусть дана база знаний K = (T, A), где T — TBox, а A — ABox.Тогда:

- K называется **согласованной**, если у неё есть модель.
- Концепт C называется **выполнимым** относительно K, если существует модель I базы знаний K, такая что $C^I \neq \varnothing$. Такая интерпретация называется **моделью C относительно K**.
- Концепт D **подчиняет** концепт C относительно K (обозначается K ⊨ C ⊑ D), если для всех моделей I базы знаний K выполняется $C^I \subseteq D^I$.
- Концепты C и D **эквивалентны** относительно K (обозначается K ⊨ C ≡ D), если они подчиняют друг друга относительно K.
- Индивид a является **экземпляром** концепта C относительно K (обозначается K ⊨ a :C), если $a^I \in C^I$ для всех моделей I базы знаний K.

• Пара индивидов (a, b) является **экземпляром роли** r относительно K (обозначается $K \models (a, b) : r$), если $(a^I, b^I) \in r^I$ для всех моделей I базы знаний K.

Пример 1. Описания в ALC основных понятий предметной области "семья".

Базовые понятия (базовые свойства объектов, атомарные концепты, базовые концепты):

Человек, женский род

Определяемые понятия (определяемые свойства объектов, определяемые концепты, составные концепты):

Женщина, Мужчина, Мать, Отец, Родитель, Бабушка, МногодетнаяМать, МатьБезДочери, Жена

Отношения между объектами (роли):

иметьДетей (x,y), иметьМужа (x,y)

Определения сложных концептов с помощью формул ALC

TBox:

Женщина ≡ Человек ⊓ женскийРод

Мужчина ≡ Человек ⊓ ¬женскийРод

Мать ≡ Женщина ⊓ ∃ иметьДетей.Человек

Отец ≡ Мужчина ⊓ ∃ иметьДетей.Человек

Родитель ≡ Мать ⊔ Отец

Бабушка ≡ Мать ⊓ ∃ иметьДетей.Родитель

МногодетнаяМать ≡ Мать ⊓ ≥3иметьДетей

МатьБезДочери ≡ Мать ⊓ ∀ иметьДетей.¬Женщина

Жена ≡ Женщина ⊓ ∃ иметьМужа.Мужчина

ABox:

МатьБезДочери (МАРИЯ)

Отец (ПЕТР)

иметьДетей (МАРИЯ, ПЕТР)

иметьДетей (ПЕТР, ИВАН)

иметьДетей (МАРИЯ, СЕМЕН)

Пример 2. SNOMED CT

• Терминология медицинских терминов насчитывающая около 400000 определений (400000 имен концептов в 60 имен ролей)

• Принадлежит (и поддерживается) некоммерческой организации IHSTDO (International Health terminology Standards Development Organisation).

• IHSTDO имеет 9 членов (бесплатное членство для 49 развивающихся стран)

• Цель: обеспечение обмена информацией между медиками, исследователями и пациентами по всему миру.

Приведем два коротеньких фрагмента TBox онтологии SNOMED CT

Фрагмент 1 (устройство бедренной кости):

EntireFemur ⊑ StructureOfFemur

FemurPart ⊑ StructureOfFemur ⊓ ∃part_of.EntireFemur

BoneStructureOfDistalFemur ⊑ FemurPart

EntireDistalFemur ⊑ BoneStructureOfDistalFemur

DistalFemurPart ⊑ BoneStructureOfDistalFemur ⊓

∃part_of.EntireDistalFemur

StructureofDistalEpiphysisOfFemur ⊑ DistalFemurPart

EntireDistalEpiphysisOfFemur ⊑ StructureOfDistalEpiphysisOfFemur

Фрагмент 2 (кардиология):

Pericardium ⊑ Tissue ⊓ ∃cont_in.Heart

Pericarditis ⊑ Inflammation ⊓ ∃has_loc.Pericardium

Inflammation ⊑ Disease ⊓ ∃acts_on.Tissue

Disease ⊓ ∃has_loc.∃cont_in.Heart ⊑ Heartdisease ⊓ NeedsTreatment

Мы видим, что в качестве языка представления знаний в SNOMED CT используется очень простая дескриптивная логика, называемая EL, в которой допускаются только два конструкта $\sqcap$ и $\exists r.C$.

Пример 3. Описание произвольной предметной области.

В терминологической части, называемой **TBox**, описываются понятия предметной области, их свойства и взаимосвязи — это аналог схемы в базах данных. Например,

- $\text{HappyMan} \equiv \text{Human} \sqcap \neg\text{Female} \sqcap (\exists \text{married.Doctor}) \sqcap (\forall \text{hasChild.}(\text{Doctor} \sqcup \text{Professor}))$

(Счастливый человек это мужчина, женатый на докторе, и все дети которого — либо доктора, либо профессора);

- $\text{Person} \sqcap \forall \text{hasChild.Male}$

(класс людей, у которых все дети мужского пола);

- $\exists \text{interested\_in.Computer\_Science} \sqcap \neg\exists \text{interested\_in.Philosophy}$

(класс объектов интересующихся информатикой, но с интересом в философии);

- $\text{Living\_being} \sqcap \neg\text{Human\_being}$

(живые существа не являющиеся людьми);

- $\text{Student} \sqcap \neg\exists \text{interested\_in.Mathematics}$

(студенты, не интересующиеся математикой);

- $\text{Student} \sqcap \forall \text{drinks.tea}$

(студенты, которые пьют только чай);

- $\text{Person} \sqcap \forall \text{hasChild.}(\text{Male} \sqcap \exists \text{hasChild.}\top)$

(класс объектов, у которых все дети мужского пола, имеющие собственных детей).

В TBox можно указывать общие аксиомы, например:

$\exists \text{hasChild.Human} \sqsubseteq \text{Human}$

«Только человек может иметь детей, которые тоже являются людьми».

Утверждающая часть базы знаний, называемая **ABox**, используется для описания конкретной ситуации путём указания свойств индивидов — это аналог данных в базе данных. Пример ABox:

- HappyMan (Bob)
- hasChild (Bob, Mary)
- $\neg$Doctor (Mary)

3. Запросы к базе знаний и логический вывод

Запросы к базе знаний K = (T, A), где T — TBox, а A — ABox бывают нескольких типов.

Определить, является ли данный индивидуум **a** экземпляром заданного концепта C? Т.е. необходимо проверить, выполнимость
K ⊨ a :C ? Отвечает на этот запрос *алгоритм экземпляров.*

Определить, является ли один концепт C подмножеством другого концепта D ? Т.е. необходимо проверить, выполнимость
K ⊨ C⊑D ? Отвечает на этот запрос *алгоритм подчинения.*

Определить, является ли база знаний K согласованной? Отвечает на этот запрос *алгоритм согласованности.* Запросы этого типа являются самыми общими, к ним сводятся запросы остальных типов.

Все эти алгоритмы опираются на общий алгоритм табличного вывода для логики ALC, изложению которого посвящен следующий параграф.

4.Табличный алгоритм для ALC

Сосредоточимся на задаче **проверки согласованности базы знаний**, поскольку она является наиболее общей задачей, к которой можно свести многие другие. Например, для базы знаний K = (T, A), концепт C подчинён концепту D относительно K (K ⊨ C ⊑ D) тогда и только тогда, когда база знаний

K'=(T, A ∪ {x : (C ⊓ ¬D)}) — несогласованна, где x — новое имя индивидуума (то есть, не встречающееся в K).

4.1.Приведенная нормальная форма

Концепт находится в **приведенной нормальной форме, ПНФ** (Negation Normal Form, NNF) если отрицание находится только перед *именами концептов.*

Каждый ALC-концепт может быть трансформирован в эквивалентный концепт в ПНФ при помощи следующих правил:

¬⊤ ≡ ⊥

¬⊥ ≡ ⊤

¬¬C ≡ C

¬(C ⊓ D) ≡ ¬C ⊔ ¬D (закон де Моргана)

$\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$ (закон де Моргана)
$\neg\forall R.C \equiv \exists R.\neg C$

Пример 4.
Преобразуем концепт
$\neg\exists R.(A \sqcap \neg B) \sqcup \neg\forall R.(\neg A \sqcup \neg B)$
в ПНФ:
$\neg\exists R.(A \sqcap \neg B) \sqcup \neg\forall R.(\neg A \sqcup \neg B) \equiv$ (используя $\neg\exists R.D \equiv \forall R.\neg D$)
$\forall R.\ \neg(A \sqcap \neg B) \sqcup \neg\forall R.(\neg A \sqcup \neg B) \equiv$ (используя $\neg(A \sqcap D) \equiv \neg A \sqcup \neg D$)
$\forall R.(\neg A \sqcup \neg\neg B\ ) \sqcup \neg\forall R.(\neg A \sqcup \neg B) \equiv$ (используя $\neg\neg B \equiv B$)
$\forall R.(\neg A \sqcup B) \sqcup \neg\forall R.(\neg A \sqcup \neg B) \equiv$ (используя $\neg\forall R.D \equiv \exists R.\neg D$)
$\forall R.(\neg A \sqcup B) \sqcup \exists R.\ \neg(\neg A \sqcup \neg B) \equiv$ (используя $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$)
$\forall R.(\neg A \sqcup B) \sqcup \exists R.(\neg\neg A \sqcap \neg\neg B\ ) \equiv$ (используя $\neg\neg C \equiv C$)
$\forall R.(\neg A \sqcup B) \sqcup \exists R.(A \sqcap B)$

Если база знаний К=(T, A) содержит концепты, мы предполагаем, что они записаны в приведенной нормальной форме ПНФ, то есть отрицания применяются только к атомарным концептам.

4.2. Предварительные соображения

Пусть мы рассматриваем задачу проверки выполнимости вложения концептов: для концептов C, D проверить, верно ли что $C \sqsubseteq D$? Вместо того, чтобы непосредственно проверять вложения описанных концептов, табличный алгоритм использует отрицание для сведения вложения концептов к (не) выполнимости некоторого другого концепта: $C \sqsubseteq D$ iff $C \sqcap \neg D$ невыполним.
Прежде чем более глубоко и более формально описать табличный алгоритм выполнимости для ALC, мы проиллюстрируем основные идеи простым примером.
Пример 5. Пусть A, B - имена концептов, а R - имя роли.
В качестве примера предположим, что мы хотим знать, включается ли концепт $(\exists R.A) \sqcap (\exists R.B)$ в концепт $\exists R.(A \sqcap Б)$.
Это означает, что мы должны проверить, является ли концепт
$C = (\exists R.A) \sqcap (\exists R.B) \sqcap \neg(\exists R.(A \sqcap B))$ невыполнимым.
Сначала мы продвинем все отрицания вглубь описания концепта,

используя правила де Моргана и правила для кванторов. В результате мы получим описание

$C_0 = (\exists R.A) \sqcap (\exists R.B) \sqcap \forall R.(\neg A \sqcup \neg B)$,

которое находится в предваренной нормальной форме, где все отрицания стоят только непосредственно перед именами концептов. Затем мы постараемся построить конечную интерпретацию I такую, что $C_0^I \neq \varnothing$.

Это значит, что должен существовать объект в Δ^I который являлся бы элементом C_0^I. Алгоритм генерирует такой объект, назовем его b, и налагает на b ограничение $b \in C_0^I$. Так как C_0 есть конъюнкция описаний концептов, то b должен удовлетворять следующим трем ограничениям:

$b \in (\exists R.A)^I$, $b \in (\exists R.B)^I$, and $b \in (\forall R.(\neg A \sqcup \neg B))^I$.

Из $b \in (\exists R.A)^I$ мы заключаем, что должен существовать объект c, такой что $(b,c) \in R^I$ и $c \in A^I$. Аналогично, $b \in (\exists R.B)^I$ влечет существование объекта d с ограничениями $(b, d) \in R^I$ и $d \in B^I$. В этой ситуации мы не можем считать объекты c и d одинаковыми, так как это было бы дополнительным и не обоснованным ограничением на сгенерированный объект b.

Таким образом для любого экзистенциального ограничения алгоритм вводит новый объект в качестве последователя роли, и этот объект должен удовлетворять требованиям, данного ограничения.

Так как b должен удовлетворять ограничению $\forall R.(\neg A \sqcup \neg B)$, и объекты c, d были введены как R-последователи b, мы получим дополнительные ограничения

$c \in (\neg A \sqcup \neg B)^I$ и $d \in (\neg A \sqcup \neg B)^I$.

Таким образом алгоритм порождает новые ограничения на объекты, полученные в результате анализа рассмотренных ранее ограничений ролей.

Теперь $c \in (\neg A \sqcup \neg B)^I$ означает, что $c \in (\neg A)^I$ или $c \in (\neg B)^I$, и мы должны выбрать одну из этих возможностей. Если мы предположим

$c \in (\neg A)^I$, то это будет противоречить другому ограничению $c \in A^I$. Поэтому мы должны выбрать $c \in (\neg B)^I$. Аналогично мы должны выбрать $d \in (\neg A)^I$ чтобы удовлетворить ограничение $d \in (\neg A \sqcup \neg B)^I$ и избежать противоречия с ограничением $d \in B^I$.

Таким образом для дизъюнктных ограничений алгоритм должен рассмотреть обе возможности удовлетворить эти ограничения. Если одна из возможностей приводит к противоречию, он должен рассмотреть и другую возможность.

Мы теперь выполнили все ограничения, не сталкиваясь с очевидными противоречиями. Это показывает, что C_0 является выполнимым, и, следовательно, концепт $(\exists R.A) \sqcap (\exists R.B)$ не включается в концепт $\exists R.(A \sqcap B)$. Алгоритм сгенерировал интерпретацию I как обоснование этого факта:

$\Delta^I = \{b,c,d\}$; $R^I = \{(b,c), (b,d)\}$; $A^I = \{c\}$ и $B^I = \{d\}$.

Для этой интерпретации $b \in C_0{}^I$. Это означает, что $b \in ((\exists R.A) \sqcap (\exists R.B))^I$, но $b \notin (\exists R.(A \sqcap B))^I$.

4.3. Общая схема табличного алгоритма для ALC

Мы попытаемся доказать согласованность базы знаний K = (T, A), строя (представление) модели этой базы знаний.

Начинаем с конкретной ситуации, описанной в A, и явно добавляем дополнительные ограничения, вытекающие из концептов и аксиом T.

Чтобы алгоритм был решающим, он должен строить конечные деревья, представляющие (возможно бесконечные) деревья в модели (при условии, что модель существует).

Для построения такого конечного представления алгоритм использует структуру данных**,** называемую лесом завершения (completion forest). Она представляет собой направленный граф с метками, каждая вершина которого является корнем дерева завершения. Каждая вершина x помечается множеством концептов L(x), а каждое ребро $\langle x,y \rangle$ помечается множеством имён ролей $L(\langle x, y \rangle)$.

Если $\langle x, y \rangle$ — ребро в лесу завершения, то мы говорим, что x - предок y (и y - потомок x).

Если метка ребра содержит имя роли r, то x — r-предок y, а y — r-потомок x.

Когда алгоритм запускается с базой знаний (T, A), он инициализирует лес завершения FA следующим образом:

- Для каждого имени индивида a, встречающегося в A, создаётся корневая вершина x_a с меткой $L(x_a) = \{C \mid a : C \in A\}$
- Для каждой пары (a, b) таких, что $(a, b) : r \in A$, создаётся ребро $\langle x_a, x_b \rangle$ с меткой: $L(\langle x_a, x_b \rangle) = \{r\}$.

Алгоритм применяет правила расширения (expansion rules), которые синтаксически декомпозируют концепты, указанные в метках вершин, тем самым:

-либо добавляя новые ограничения для вершины,

-либо расширяя дерево согласно этим ограничениям.

4.4. Правила табличного алгоритма для ALC

(⊓)-Правило (конъюнкция)

Если концепт $C_1 \sqcap C_2$ содержится в L(x), и ни C_1, ни C_2 не содержатся в L(x), то добавить C_1 и C_2 в L(x).

(⊔)-Правило (дизъюнкция)

Если концепт $C_1 \sqcup C_2$ содержится в L(x), и ни C_1, ни C_2 не содержатся в L(x), то недетерминированно выбрать одно из:

- добавить C_1 в L(x),
- добавить C_2 в L(x).

(∀)-Правило (универсальное ограничение)

Если концепт $\forall r.C$ содержится в L(x), и существует r-потомок y такой, что $C \notin L(y)$, то добавить C в L(y).

(∃)-Правило (экзистенциальное ограничение)

Если концепт $\exists r.C$ содержится в L(x), и x не имеет r-потомка y с $C \in L(y)$, то создать новую вершину y с $L(y) = \{C\}$ и добавить ребро $\langle x, y \rangle$ с меткой $\{r\}$.

Алгоритм продолжает применять эти правила, пока:

- ни одно правило больше не применимо

- возникает конфликт.

Конфликт возникает в вершине x, если:

- $\exists$ концепт A, такой что $A \in L(x)$ и $\neg A \in L(x)$,
- $\bot \in L(x)$.

В случае конфликта дерево считается несогласованным.

Если лес завершения можно расширить так, что в результате не будет конфликтов, то исходная база знаний согласована.

Если все возможные расширения леса завершаются конфликтом, то база знаний не согласована.

Пример 6. Пусть дан концепт:

$C:=\exists R.A \sqcap \exists R.(B \sqcup \neg A) \sqcap \forall R.\neg B$

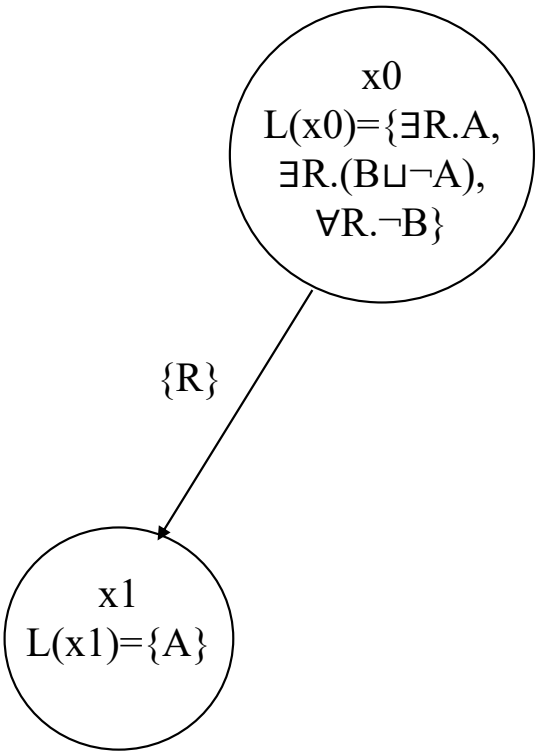
Мы хотим проверить его выполнимость, то есть существует ли модель, в которой C интерпретируется как непустое множество.

Шаг 1 (инициализация):

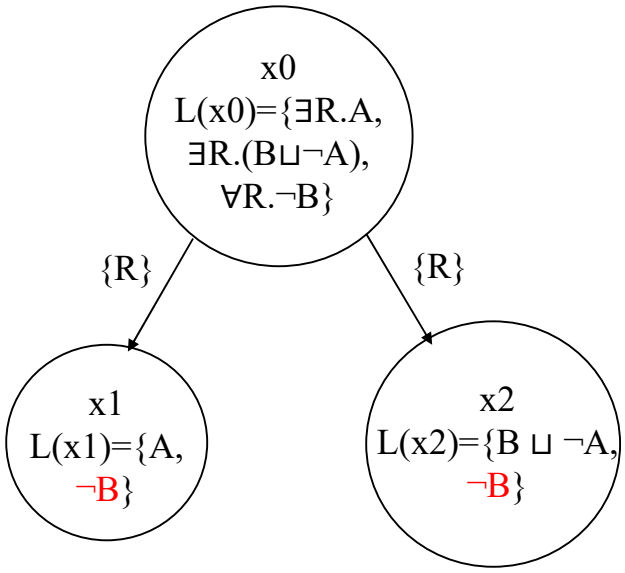
Создается вершина x_0 с меткой $L(x_0)=\{\exists R.A, \exists R.(B \sqcup \neg A), \forall R.\neg B\}$

Шаг 2: Применим $\exists$-правило к $\exists R.A \in L(x_0,)$

Создаем вершину x_1, $L(x_1) = \{A\}$, $L(\langle x_0, x_1 \rangle) = \{R\}$



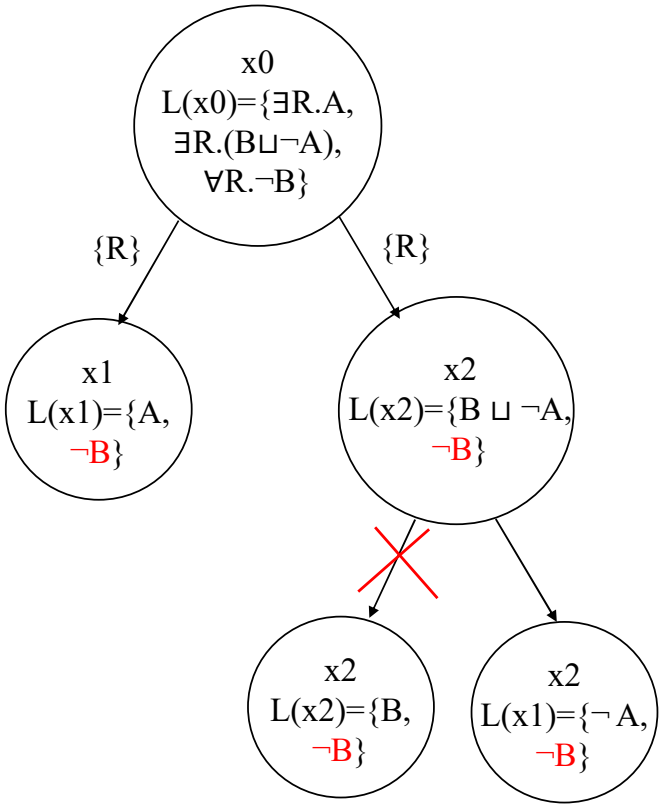
Шаг 3: Применим ∃-правило к $\exists R.(B \sqcup \neg A) \in L(x_0)$
Создаем вершину x_2, $L(x_2) = \{B \sqcup \neg A\}$, $L(\langle x_0, x_2 \rangle) = \{R\}$
Шаг 4: Применим ∀-правило к $\forall R.\neg B \in L(x_0)$
Добавляем ¬B в метки всех R-потомков x_0
$L(x_1) := \{A, \neg B\}$, $L(x_2) := \{B \sqcup \neg A, \neg B\}$
Действия этого шага выделены красным



Шаг 5: Применим (⊔)-правило к $B \sqcup \neg A \in L(x_2)$
Недетерминированно выбираем один из вариантов:
Ветка 1: выбрать B
⇒ $L(x_2) := \{B, \neg B\}$ — возникает конфликт, ветка закрывается

Ветка 2: выбрать ¬A
⇒ $L(x_2) := \{\neg A, \neg B\}$ — конфликтов нет, ветка допустима

⇒Концепт выполним



Из допустимого леса завершения строим модель - интерпретацию I:
Область $\Delta^I = \{x_0, x_1, x_2\}$
Интерпретация имен концептов:
$A^I = \{x_1\}$
$B^I = \varnothing$ (¬B принадлежит всем вершинам)
3) Интерпретация роли:
$R^I = \{ (x_0, x_1), (x_0, x_2) \}$
4) Проверим, что $x_0 \in C^I$:
- x_0 имеет R-последователя $x_1 \in A^I$, следовательно $x_0 \in \exists R.A$
- x_0 имеет R-последователя x_2, и $x_2 \in \{\neg A, \neg B\}$,
следовательно, $x_2 \in B \sqcup \neg A$, а значит $x_0 \in \exists R.(B \sqcup \neg A)$ выполнено;
- видим, что для всех R-последователей x_1 и x_2 выполняется ¬B , следовательно, $x_0 \in \forall R.\neg B$ выполнено;
Тогда $x_0 \in C^I$, что и требовалось показать.

Отметим две важных черты табличного алгоритма для ALC:
1) Табличный алгоритм является недетерминированным: (⊔) - правила дизъюнкции требуют рассмотрения альтернатив;

2) ∃-правило может порождать новые вершины, что может привести к бесконечности числа шагов построения леса (см. пример ниже).
Для обеспечения конечности построения леса используется техника “свёртки (blocking)”:

- Если во время построения появляется вершина y, которая “повторяет” метку ранее появившейся вершины x, и y — потомок x, то дальнейшее расширение из y прекращается.

Пример 7. Рассмотрим табличный алгоритм для ALC с блокированием вершины.

Пусть дан концепт:

$C \equiv A \sqcap \exists r.(B \sqcap \exists r.C)$

Проверим его выполнимость.

Начнем с вершины x_0, где $L(x_0) = \{C\}$

Применим правила:

1. $C = A \sqcap \exists r.(B \sqcap \exists r.C)$

⇒ по (⊓)-правилу: добавить A и $\exists r.(B \sqcap \exists r.C)$ в $L(x_0)$

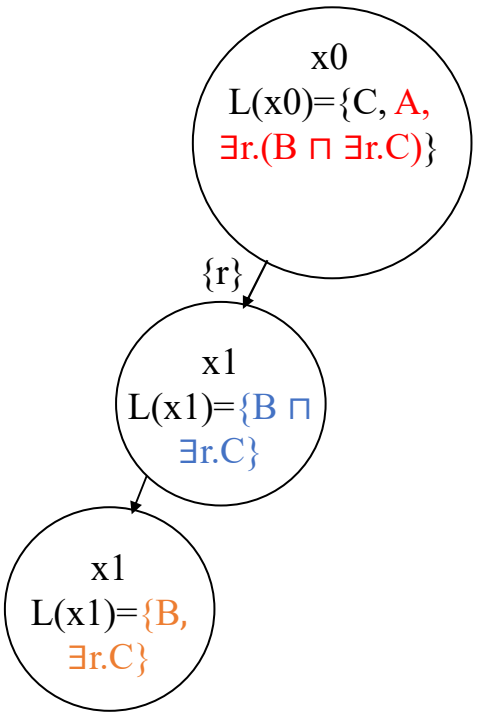
2. $\exists r.(B \sqcap \exists r.C) \in L(x_0)$

⇒ по (∃)-правилу: создать x_1,

$L(x_1) = \{B \sqcap \exists r.C\}$, $L(\langle x_0, x_1 \rangle) = \{r\}$

3. $B \sqcap \exists r.C \in L(x_1)$

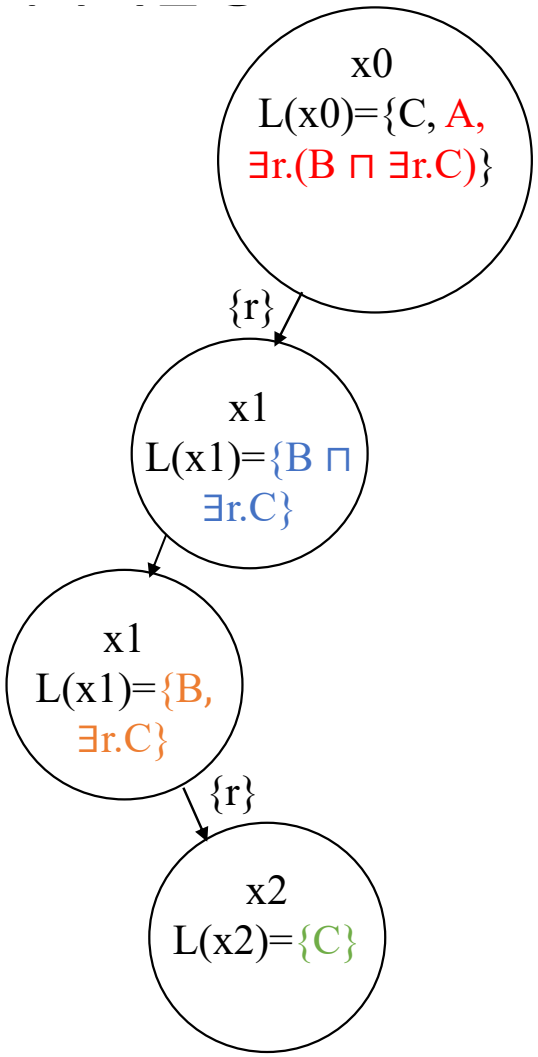
⇒ по (⊓)-правилу: добавить B и $\exists r.C$ в $L(x_1)$



4. $\exists r.C \in L(x_1)$
$\Rightarrow$ по ($\exists$)-правилу: создать x_2, $L(x_2) = \{C\}$, $L(\langle x_1, x_2 \rangle) = \{r\}$

5. $C \in L(x_2)$, повторяется как в x_0
$\Rightarrow$ если $L(x_2) = L(x_0)$, то x_2 блокируется по x_0

Таким образом, получаем конечное дерево, и можно развернуть модель, удовлетворяющую C $\Rightarrow$ концепт выполним.



Из построенного допустимого леса завершения строим модель - интерпретацию I:
1) Область $\Delta^I = \{x_0, x_1, x_2, \ldots\}$ (теоретически бесконечна)
2) Интерпретация имен концептов:

A^I = {x0, x3, x6, …}
B^I = {x1, x4, x7, …}
C^I = {x0, x2, x5, …}
3) Интерпретация роли:
r^I = { (x_0, x_1), (x_0, x_2), (x2, x3), …}
4) Проверим, что $x_0 \in C^I$:
$A \in L(x_0) \Rightarrow x_0 \in A^I$

$\exists r.(B \sqcap \exists r.C) \in L(x_0)$: x_0 связан с x_1 через r, $x_1 \in B^I$ и $\exists r.C$ выполняется, так как есть x2: r(x1, x2) и $x2 \in C^I$.

5. Важные расширения ALC

С точки зрения языка для представления знаний логика ALC была расширена несколькими возможностями, важными для онтологических языков, включая:

- (качественные) числовые ограничения;
- обратные роли;
- транзитивные роли;
- иерархии ролей (подроли);
- конкретные домены;
- номиналы.

Числовые ограничения позволяют описывать количество отношений определённого типа, в которых могут участвовать индивиды.
Например, мы можем захотеть указать, что человек может быть женат не более чем на одном человеке. Это можно выразить формулой
Person $\sqsubseteq \leq 1$ married.
Мы можем также расширить определение HappyMan из примера 3, включив в него, что у него должно быть от двух до четырёх детей:
HappyMan ≡ Human ⊓ ¬Female ⊓ (∃married.Doctor) ⊓ (∀hasChild. (Doctor ⊔ Professor)) ⊓ ≥2 hasChild ⊓ ≤4 hasChild
С качественными числовыми ограничениями можно указать тип индивидов, которые учитываются в числовом ограничении. Например, мы можем далее уточнить, что у HappyMan по меньшей мере два ребёнка должны быть докторами:
HappyMan ≡ Human ⊓ ¬Female ⊓ (∃married.Doctor) ⊓ (∀hasChild. (Doctor ⊔ Professor)) ⊓ ≥2hasChild.Doctor ⊓ ≤4 hasChild

С помощью обратных ролей, транзитивных ролей и иерархий ролей мы можем,
помимо роли hasChild, использовать её обратную роль hasParent,
указать, что что роль hasParent — подроль роли hasAncestorhas, и что роль hasAncestor — транзитивна.

Конкретные домены интегрируют DL с конкретными множествами, такими как действительные числа, целые числа или строки, а также с конкретными предикатами, определёнными на этих множествах, например:

- числовые сравнения (например, ≤),
- строковые сравнения (например, isPrefixOf),
- сравнения с константами (например, ≤17).

Конструктор номинала позволяет использовать имена индивидов внутри описаний концептов:
если a — имя индивида, то {a} — это концепт, называемый **номиналом**, который интерпретируется как одноэлементное множество.
Например, используя индивидуум Turing, мы можем описать всех учёных в области компьютерных наук, которые встречались с Тьюрингом, так:
$CScientist \sqcap \exists hasMet.\{Turing\}$
Так называемый конструктор **"one-of"** расширяет номиналы до конечного множества индивидов.
В присутствии дизъюнкции он может быть выражен через номиналы: $\{a_1,..., a_n\}$ эквивалентно $\{a_1\} \sqcup ... \sqcup \{a_n\}$. Заметим однако, что наличие номиналов может радикально повлиять на сложность алгоритмов вывода.

На сегодняшний день разработаны многочисленные расширения логики ALC дополнительными конструктами для построения концептов, ролей, а также дополнительными видами аксиом в TBox . Имеется неформальное соглашение об именовании получающихся при этом логик — обычно путём добавления к имени логики букв, отвечающих добавленным в язык конструктам. Наиболее известными расширениями являются:

$\mathcal{N}$	Ограничения кардинальности ролей: концепты вида $(\leq n\, R)$, означающие: *существует не более n R-последователей*
$\mathcal{Q}$	Качественные ограничения кардинальности ролей: концепты вида $(\leq n\, R.\, C)$, означающие: *существует не более n R-последователей в C*
$\mathcal{I}$	Обратные роли: если R есть роль, то R^{-} тоже является ролью, означающей обращение бинарного отношения
$\mathcal{O}$	Номиналы: если a есть имя объекта, то $\{a\}$ есть концепт, означающий одноэлементное множество
$\mathcal{H}$	Иерархия ролей: в TBox допускаются аксиомы вложенности ролей $R \sqsubseteq S$
$\mathcal{S}$	Транзитивные роли: в TBox допускаются *аксиомы транзитивности* вида $\mathbf{Tr}(R)$
$\mathcal{R}$	Составные аксиомы вложенности ролей в TBox ($R \circ S \sqsubseteq R$, $R \circ S \sqsubseteq S$) с условием ацикличности, где $R \circ S$ есть композиция ролей
(D)	Расширение языка *конкретными доменами* (типами данных)

Например, логика ALC, расширенная инверсными ролями, номиналами и ограничениями кардинальности ролей, обозначается как ALCIOQ .

Буква S не добавляется к имени логики, а замещает в нём буквы ALC . Так, например, логика ALC, расширенная инверсными ролями (буква I), качественными ограничениями кардинальности

ролей (буква Q), транзитивными ролями (буква S) и иерархией ролей (буква H), имеет название SHIQ. Происхождение всех букв понятно из английских названий конструктов; буква $\mathcal{S}$ же была выбрана из-за тесной связи получающейся DL с модальной логикой S4 (хотя в последней буква S означает просто *system*, саму же логику S4 выделяет среди других модальных логик именно цифра **4**).

Рассматриваются также дескрипционные логики, в которых можно строить составные роли с помощью операций объединения, пересечения, дополнения, инверсии, композиции, транзитивного замыкания и других. Кроме того, исследованы DL, в которых имеются многоместные роли (обозначающие n-арные отношения).

6. Связь DL и логики предикатов

Рассматривая имена ролей как бинарные отношения, а имена концептов как унарные отношения, мы определим две функции трансляции — π_x и π_y, которые индуктивно отображают концепты ALC в формулы логики первого порядка с одной свободной переменной x или y:

$\pi_x(A) = A(x)$, $\pi_y(A) = A(y)$
$\pi_x(C \sqcap D) = \pi_x(C) \wedge \pi_x(D)$, $\pi_y(C \sqcap D) = \pi_y(C) \wedge \pi_y(D)$
$\pi_x(C \sqcup D) = \pi_x(C) \vee \pi_x(D)$, $\pi_y(C \sqcup D) = \pi_y(C) \vee \pi_y(D)$
$\pi_x(\exists r.C) = \exists y.\ r(x, y) \wedge \pi_y(C)$,
$\pi_x(\forall r.C) = \forall y.\ r(x, y) \rightarrow \pi_y(C)$,
$\pi_y(\exists r.C) = \exists x.\ r(y, x) \wedge \pi_x(C)$
$\pi_y(\forall r.C) = \forall x.\ r(y, x) \rightarrow \pi_x(C)$

Используя формулы выше, мы можем преобразовать TBox T и ABox A следующим образом ($\psi[x/a]$ означает формулу ψ с заменой всех свободных вхождений x на a):
$\pi(T) = \wedge\_\{C \sqsubseteq D \in T\}\ \forall x.\ (\pi_x(C) \rightarrow \pi_x(D))$
$\pi(A) = \wedge\_\{a:C \in A\}\ \pi_x(C)[x/a] \wedge \wedge\_\{(a,b):r \in A\}\ r(a, b)$

Теорема. Пусть (T, A) — база знаний ALC, C и D — (возможно, сложные) концепты ALC, a — имя индивида. Тогда:
1. (T, A) согласована $\Longleftrightarrow$ формула $\pi(T) \wedge \pi(A)$ выполнима
2. $(T, A) \models C \sqsubseteq D \Longleftrightarrow \pi(T) \wedge \pi(A) \models \pi(C \subseteq D)$
3. $(T, A) \models a : C \Longleftrightarrow \pi(T) \wedge \pi(A) \models \pi(a : C)$

Преобразование более выразительных DL может быть простым или сложным в зависимости от дополнительных конструктов. Например:

- **обратные роли** можно выразить в двухпеременном фрагменте, просто поменяв местами переменные:

$\pi_x(\exists R^{-}.C) = \exists y.\ R(y, x) \wedge \pi_y(C)$

- **числовые ограничения** можно выразить с помощью (не)равенства или так называемых кванторов подсчёта (counting quantifiers).

Известно, что двухпеременный фрагмент с кванторами подсчёта также разрешим за NEXPTIME.

- **транзитивные роли** не могут быть выражены формулами только с двумя переменными. А известно, что фрагмент логики предикатов с тремя переменными — неразрешим.

7. Модификации табличного алгоритма для расширений ALC

Табличный алгоритм для расширений ALC с числовыми ограничениями.

Расширение ALCN. В таких логиках допускаются числовые ограничения (Number restrictions) и используются конструкты вида $\geq n\ r$, $\leq n\ r$.

Для индивидов этих концептов при применении соответствующего правила создаются n разных r-потомков, которые в дальнейшем могут либо дублироваться, либо объединяться (сливаться).

≥-Правило: Если L(x) содержит ($\geq n\ r$) и x нет отдельных r-потомков z1, . . . , zn, то создать r-потомков y1,…,yn с новыми индивидуальными именами, не встречающимися в L(x).

≤-Правило: Если x имеет r-потомков y1, . . . , yn+1 такие, $y_i \neq y_j$ не находится в L(x) для некоторых $i \neq j$, то для каждой пары yi, yj такой, что $i > j$ и $y_i \neq y_j$ не в L(x), отождествим вхождения yi и yj.

Расширение ALCQ. Допускаются качественные числовые ограничения (Qualified number restrictions) и используются конструкты вида $\geq$ n r.C, $\leq$ n r.C. Для индивидов этих концептов создаются n разных r-потомков с C в их метках. В дальнейшем они могут либо дублироваться, либо объединяться (сливаться).

≥-Q Правило: Если L(x) содержит ($\geq$ n r.C) и у x нет отдельных r-потомков z1, . . . , zn таких что L(zi) содержит C ($1 \leq i \leq n$), то создать r-потомков y1,…,yn с новыми индивидуальными именами, не встречающимися в L(x) и поместить C в L(yi).

≤-Q Правило: Если L(x) содержит ($\leq$ n r.C) и x имеет r-потомков y1, . . . , yn+1 такие что C содержится в L(yi), yi $\neq$ yj не находится в L(x) для некоторых i$\neq$ j , то для каждой пары yi, yj такой, что i > j и yi $\neq$ yj не в L(x), отождествим вхождения yi и yj.

Пример (Продолжение примера 5).

В этом примере мы добавим численное ограничение к первому концепту приведенного выше примера 5, т.е. теперь мы хотим знать, включается ли концепт $(\exists R.A) \sqcap (\exists R.B) \sqcap \leq 1\ R$ в концепт $\exists R.(A \sqcap B)$.

Интуитивно понятно, что ответ теперь должен быть «да», так как ≤ 1 R в первом концепце гарантирует, что R-последователь в A совпадает с R-последователем в B, и, таким образом, является R-последователем в $A \sqcap B$. Алгоритм сначала выполняется, как было указано выше, с единственной разницей в том, что есть дополнительное ограничение $b \in (\leq 1\ R)^I$. Для того, чтобы удовлетворить это ограничение два R-последователя объекта b объекты c и d должны быть идентичными (по $\leq$-правилу). Назовем этот объект cd.

В примере объект cd должен принадлежать как A^I, так и B^I, что вместе с $cd \in (\neg A \sqcup \neg B)^I$ всегда приводит к противоречию. Таким образом, поиск контрпримера к вложению концептов терпит неудачу, и алгоритм приходит к выводу, что $(\exists R.A) \sqcap (\exists R.B) \sqcap \leq 1R \sqsubseteq \exists R.(A \sqcap B)$.

Таким образом, если численное ограничение в $\leq$- правиле на количество последователей нарушено, алгоритм должен отождествить разные последователи.

Табличный алгоритм для расширений ALC с обратными ролями.

Расширение ALCI или SHI. Используются обратные роли (Inverse roles, r^-). Если в метке вершины x имеется конструкт $\forall r^-.C$, то применяется следующее

Правило $\forall r^-.C$ применяется к r-предку x:

если $y \in$ r-предки(x), и $\forall r^-.C \in L(x)$, то добавить C в L(y)

Правило $\exists r^-.C$ создаёт r-предка y вершины x.

Пример табличного алгоритма для расширения ALC с транзитивными ролями.

Расширение: S, логика S(H)I. Используются транзитивные роли (Transitive roles). Если r — транзитивная, то:$\forall r.C$ должно выполняться не только у r-потомков, но и у r-потомков их r-потомков, и т.д.

Правило: если $\forall r.C \in L(x)$, $x \rightarrow r\ y$, $y \rightarrow r\ z$, то $C \in L(z)$.

Пусть роль hasAncestor объявлена транзитивной и задан концепт:

$C \equiv Person \sqcap \exists hasAncestor.(Human \sqcap \exists hasAncestor.\neg Human) \sqcap \forall hasAncestor.Human$

Проверим его выполнимость.

Шаг 1:П (инициализация): $L(x_0) = \{Person, \exists hasAncestor.(Human \sqcap \exists hasAncestor.\neg Human), \forall hasAncestor.Human\}$

Шаг 2: Применим ∃-правило к $\exists hasAncestor.(Human \sqcap \exists hasAncestor.\neg Human)$

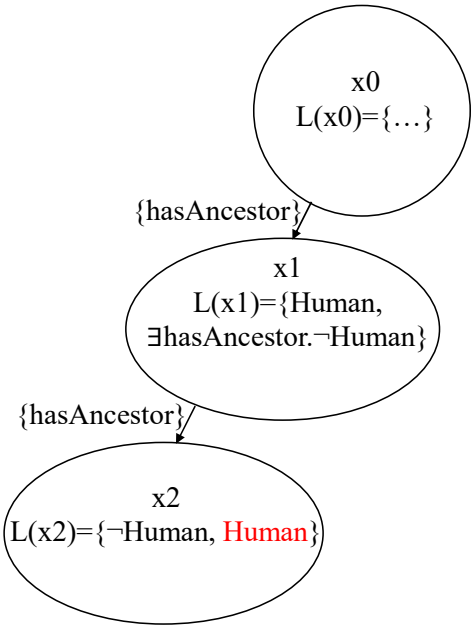
Создаем вершину x_1, $L(x_1) = \{Human, \exists hasAncestor.\neg Human\}$, $L(\langle x_0, x_1\rangle) = \{ hasAncestor\}$

Шаг 3: Применим ∃-правило к $\exists hasAncestor.\neg Human$

Создаем вершину x_2, $L(x_2) = \{\neg Human\}$, $L(\langle x_1, x_2\rangle) = \{hasAncestor\}$

Шаг 4: Применим ∀-правило к $\forall hasAncestor.Human \in L(x_0)$

По правилу транзитивных ролей добавляем Human во все hasAncestor-потомки x_0 $L(x2) = \{ \neg Human, Human\}$ – конфликт, следовательно, невозможно построить модель, в которой данный концепт был бы выполним.



Отметим, что модификации табличного алгоритма для различных расширений логики ALC могут быть очень изощренными и достаточно сложными.

8. Диаграммы сложности решений задач в расширениях логики ALC

Ниже проводятся 6 диаграмм сложности решений задач выполнимости концепта и непротиворечивости ABox в трех вариантах (при пустом TBox, при ацикличном TBox, при произвольном TBox) для различных дескриптивных логик.
В овалах приводятся названия логик стрелками обозначается вложимость соответствующих логик.

Напомним определения классов сложности, к которым относятся задачи для указанных логик.

Класс сложности PSPACE — это множество задач в теории сложности вычислений, которые могут быть разрешены машиной Тьюринга с полиномиальным ограничением пространства.

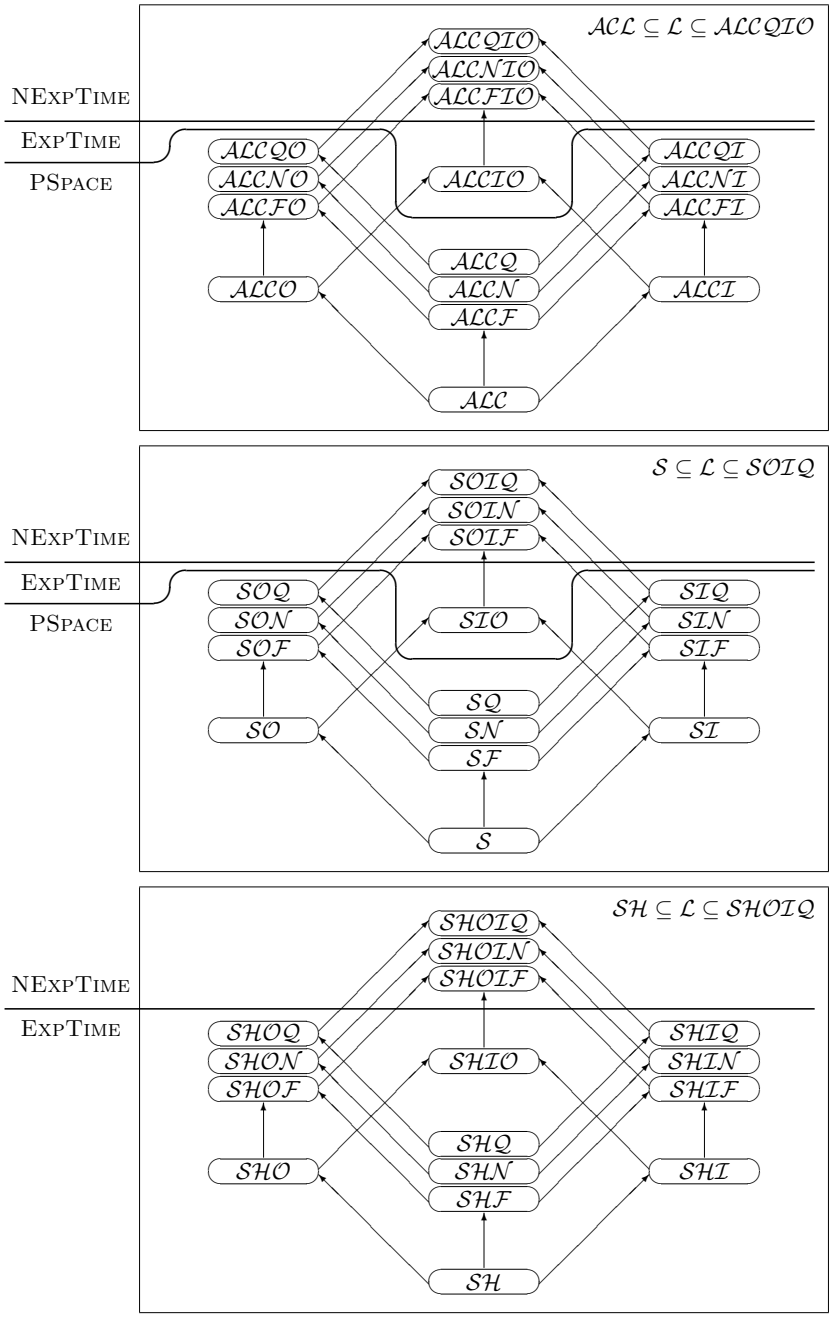
Класс сложности EXPTIME (иногда называемый просто EXP) — это множество задач, в теории сложности вычислений, решаемых с помощью детерминированной машины Тьюринга за время $O(2^{p(n)})$, где p(n) это полиномиальная функция от n.

Класс сложности NEXPTIME (иногда называемый просто NEXP) — это множество задач, в теории сложности вычислений, решаемых с помощью недетерминированной машины Тьюринга за время $O(2^{p(n)})$, где p(n) это полиномиальная функция от n.

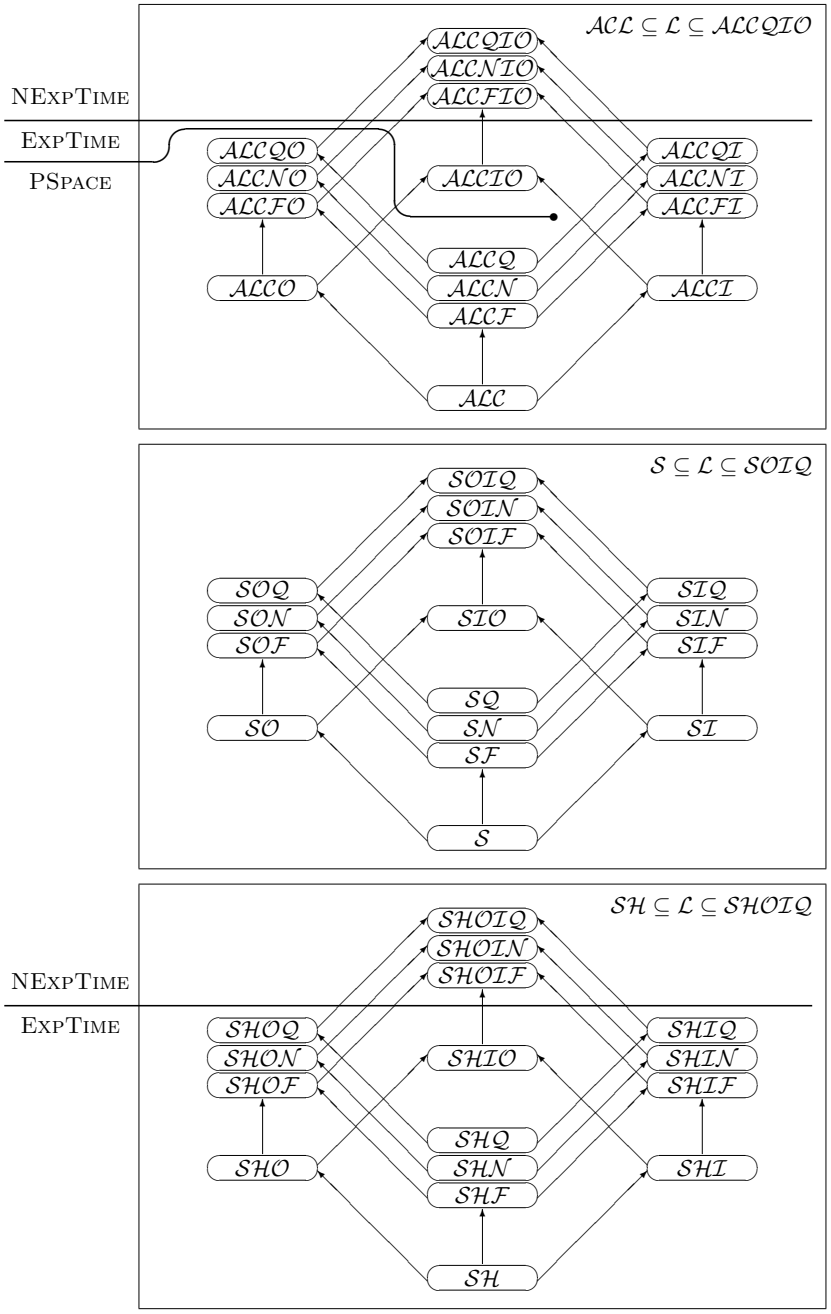
Известно, что

$P \subseteq NP \subseteq$ **PSPACE** $\subseteq$ **EXPTIME** $\subseteq$ **NEXPTIM**E $\subseteq$ EXPSPACE

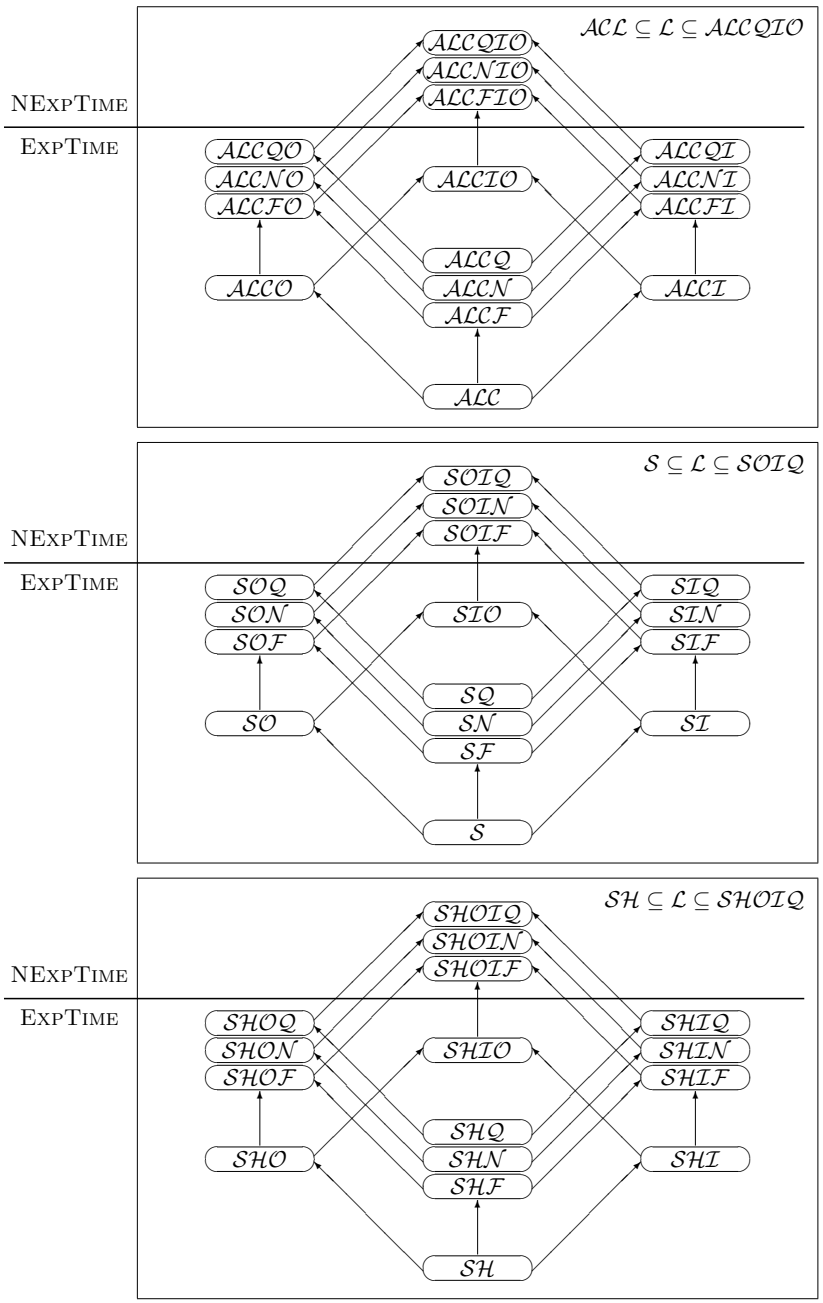
Complexity of **concept satisfiability** problem w.r.t. **empty TBox**



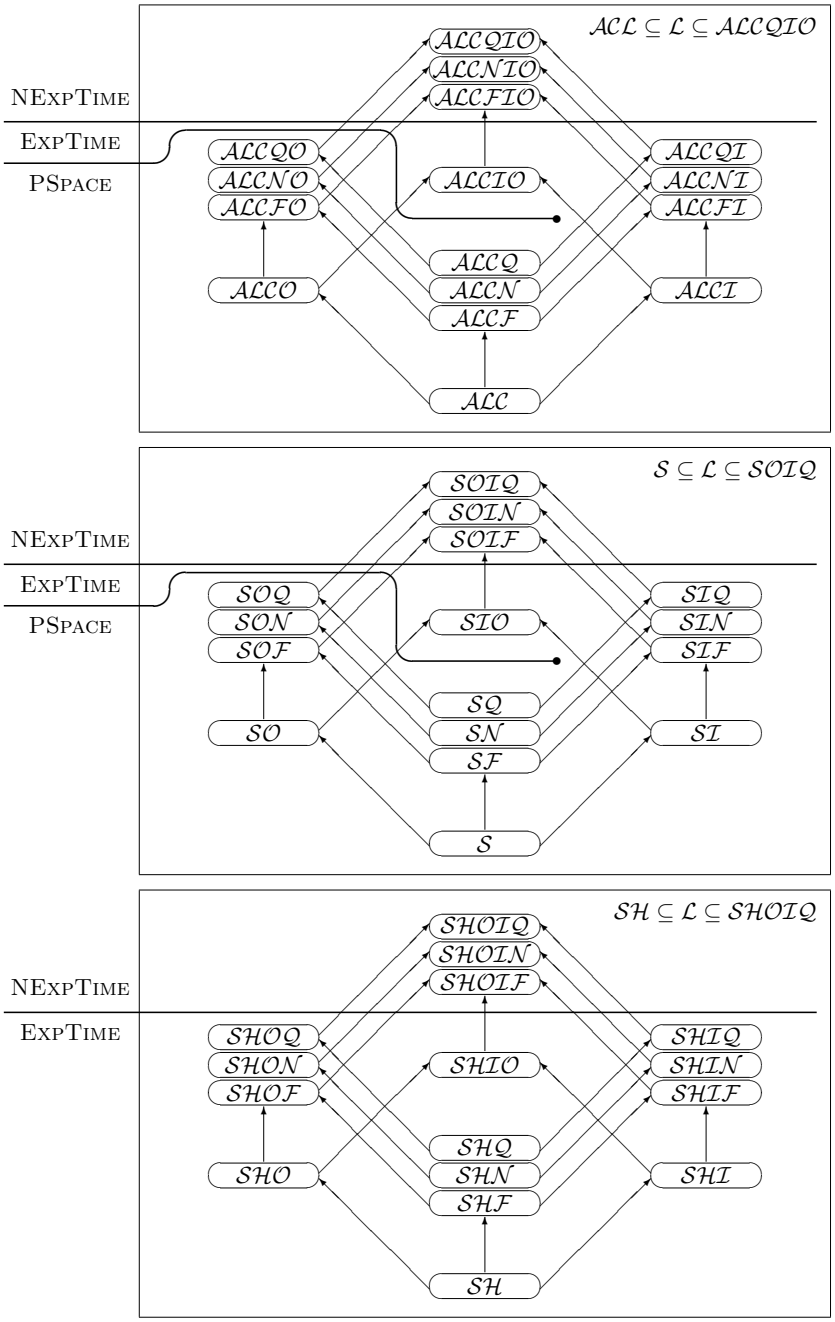
Complexity of **concept satisfiability** problem w.r.t. **acyclic TBox**



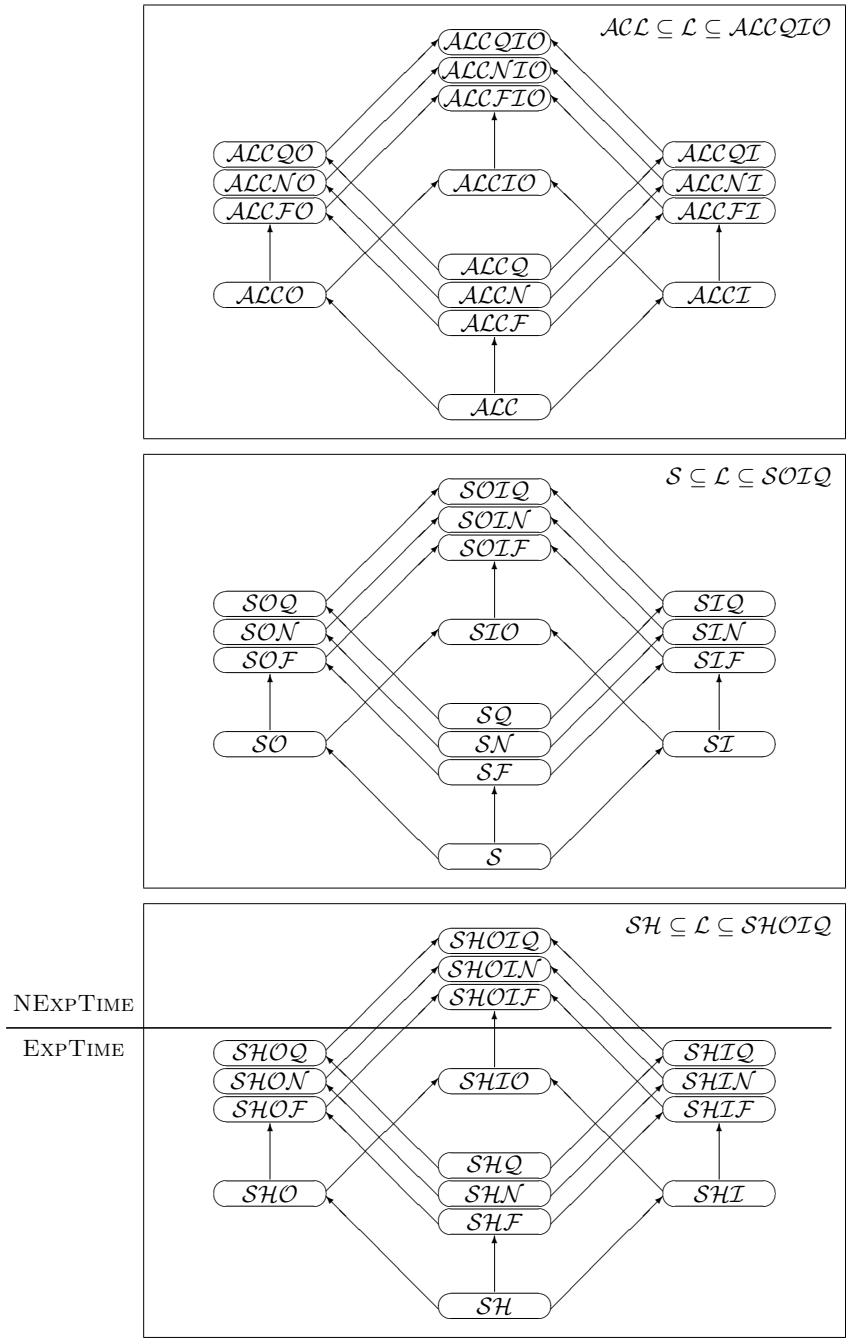
Complexity of **concept satisfiability** problem w.r.t. **general TBox**



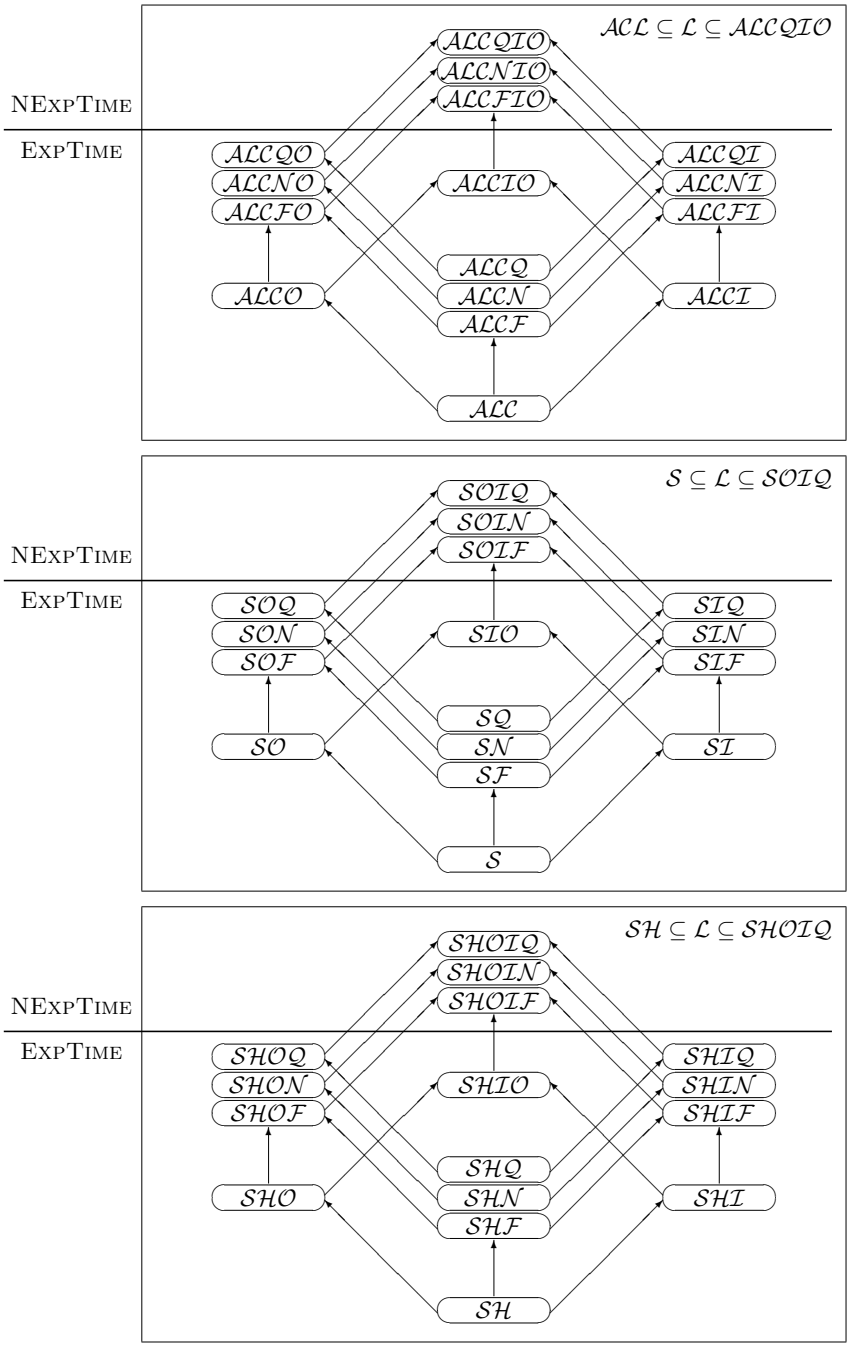
Complexity of **ABox consistency** problem w.r.t. **empty TBox**



Complexity of **ABox consistency** problem w.r.t. **acyclic TBox**



Complexity of **ABox consistency** problem w.r.t. **general TBox**



9. Решение задач с помощью DL и онтологических моделей

Имеется множество программных систем (reasoners), позволяющих совершать логический анализ в дескрипционных логиках (проверять базу знаний на непротиворечивость, строить таксономии, проверять выполнимость и вложенность концептов, реализовывать запросы к базам знаний и др.). Подобные системы различаются по поддерживаемым ими дескрипционным логикам, по типу реализованной в них разрешающей процедуры (например, табличный алгоритм, метод резолюций и т. п.), по поддерживаемым форматам данных, языку программирования, на котором они реализованы, и другим параметрам. Некоторые известные системы:

- CEL — поддерживает логику EL+ , имеющую полиномиальную сложность, написана на Листе;
- FaCT++ — поддерживает логику SROIQ(D), реализует табличный алгоритм, написана на C++;
- Kaon2 — поддерживает логику SHIQ, расширенную специальными правилами вывода, реализует алгоритм, основанный на методе резолюций, написана на Java;
- Pellet — поддерживает логику SROIQ(D), реализует табличный алгоритм, написана на Java;
- RacerPro — поддерживает логику SHIQ(D), реализует табличный алгоритм, написана на Лиспе.

Существуют также редакторы онтологий, позволяющие создавать онтологии, сохранять их в различных форматах, некоторые позволяют подключить блок рассуждений (reasoner) и с его помощью произвести логический анализ онтологии. Одним из

наиболее известных является редактор онтологий Protégé, позволяющий работать с онтологиями в языке OWL Full.

Рассмотрим популярную задачу, которая широко известна как задача Эйнштейна. Это логическая задача, заданная 15-ми утверждениями, из которых нужно найти ответы на вопросы, о том, кто пьёт воду и держит зебру.

Оригинальный текст задачи выглядит следующим образом:

1. На улице стоят пять домов.
2. Англичанин живёт в красном доме.
3. У испанца есть собака.
4. В зелёном доме пьют кофе.
5. Украинец пьёт чай.
6. Зелёный дом стоит сразу справа от белого дома.
7. Тот, кто курит Old Gold, разводит улиток.
8. В жёлтом доме курят Kool.
9. В центральном доме пьют молоко.
10. Норвежец живёт в первом доме.
11. Сосед того, кто курит Chesterfield, держит лису.
12. В доме по соседству с тем, в котором держат лошадь, курят Kool.
13. Тот, кто курит Lucky Strike, пьёт апельсиновый сок.
14. Японец курит Parliament.
15. Норвежец живёт рядом с синим домом.

Кто пьёт воду? Кто держит зебру?

В целях ясности следует добавить, что каждый из пяти домов окрашен в свой цвет, а их жители — разных национальностей, владеют разными животными, пьют разные напитки и курят разные марки американских сигарет. Ещё одно замечание: в утверждении 6 *справа* означает справа относительно *вас*.

Возможное практическое решение

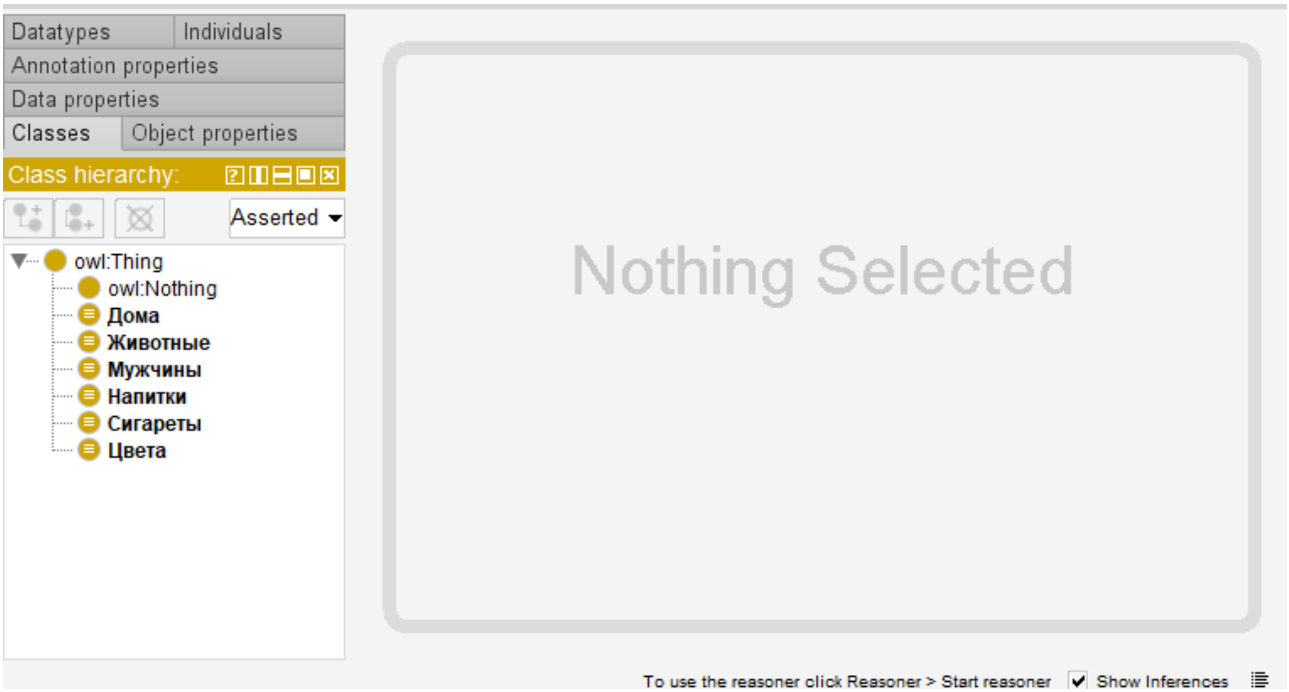
Будем описывать предметную область этой задачи с помощью конструкций DL и одновременно строить онтологическую модель задачи с помощью редактора онтологий Protege.

На первом этапе определяем концепты (классы). Из утверждений можно понять, что их пять: «Дома», «Мужчины», «Животные», «Напитки», «Сигареты». Так как у класса «Дома» есть два признака – это номер дома и цвет, то выделим в отдельный концепт 'цвет дома' с названием «Цвета».

Первым шагом является создание нового проекта онтологии в Protege. Для этого запускаем Protege, выбираем «File» -> «New» и указываем IRI (Internationalized Resource Identifier) для новой онтологии. В качестве IRI можно использовать URL вида «http://www.example.com/webenshtein.owl», который будет служить уникальным идентификатором онтологии. Также полезно добавить метаданные онтологии, такие как название, версия, авторы и описание, используя аннотационные свойства rdfs:label, owl:versionInfo, dc:creator и dc:description.

После создания проекта приступаем к определению основных классов онтологии и их иерархии. Используя вкладку «Classes» в Protege, создаем корневые классы онтологии: Дома, Мужчины, Животные, Напитки, Сигареты, Цвета.

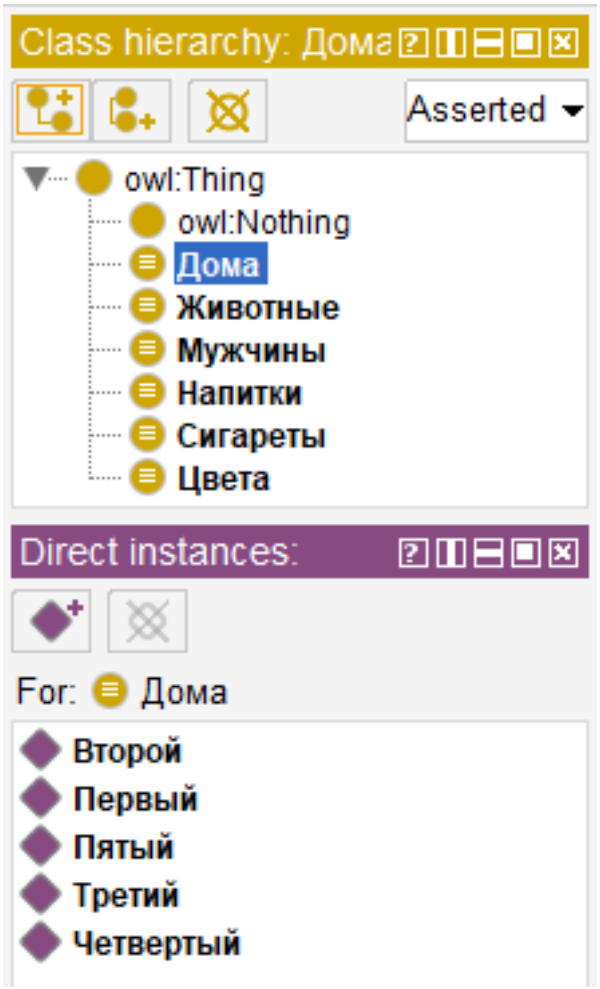
Protege:



Затем создаем объекты - индивимы (экземпляры классов)

DL:
Дома ≡ {Первый, Второй, Третий, Четвертый, Пятый}
Животные ≡ {собака, улитка, лиса, лошадь, зебра}
Мужчины ≡{ англичанин, испанец, украинец, норвежец, японец}
Напитки ≡{ кофе, чай, вода, молоко, апельсиновый сок}
Сигареты ≡{Old Gold, Kool, Chesterfield, Lucky Strike, Parliament}
Цвета ≡ { красный, зеленый, белый, желтый, синий}

Protege:

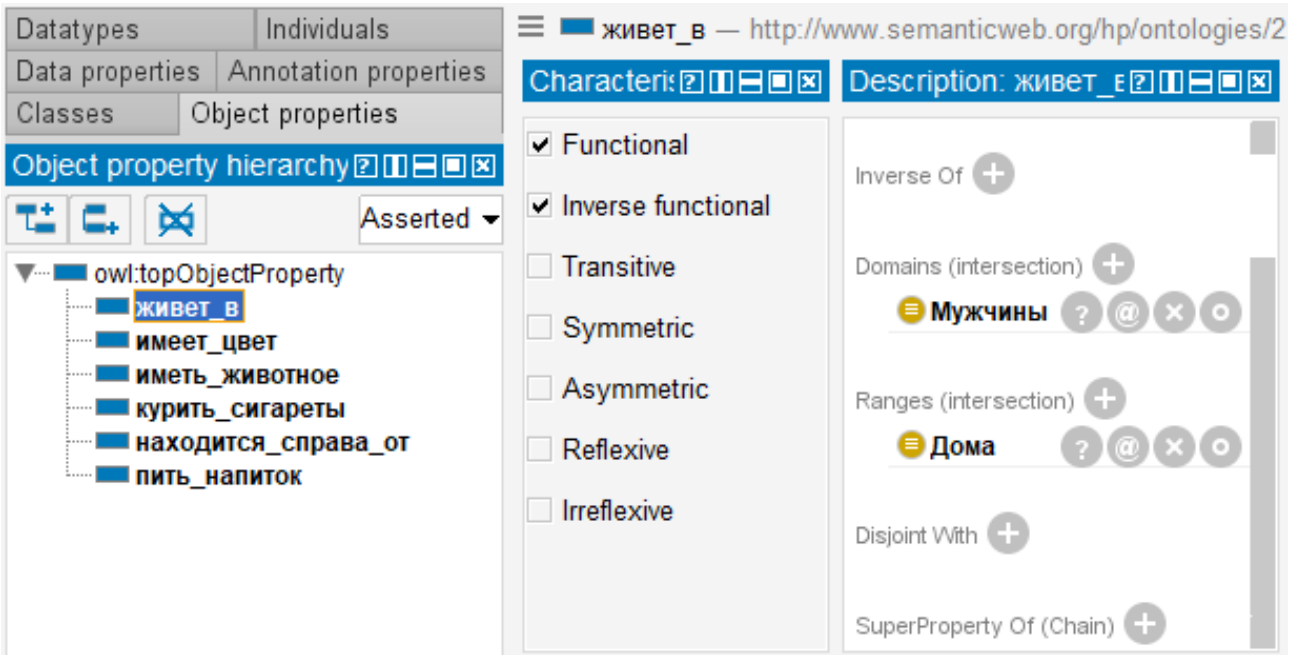


и то же самое для всех остальных классов.

Следующим шагом создадим роли или свойства объектов и обозначим их характеристики. Создадим 6 ролей:
«жить_в», «иметь_цвет», «иметь_животное», «курить_сигареты», «находиться_справа_от», «пить_напиток».

Роль *живет_в* имеет область определения Мужчины и область значений Дома. Она является функциональной ролью, и обратная роль к ней тоже функциональна.
Роль *имеет_цвет*. Область определения Дома и область значений Цвета, она является функциональной ролью, и обратная роль к ней тоже функциональна.
Роль *иметь_животное*. Область определения Мужчины и область значений Животное, является функциональной ролью, и обратная роль к ней тоже функциональна.
Роль *курить_сигареты*. Область определения Мужчины и область значений Сигареты, является функциональной ролью, и обратная роль к ней тоже функциональна.
Роль *находится_справа_от*. Область определения Дома и область значений Дома, является функциональной ролью, и обратная роль к ней тоже функциональна.
Роль *пить_напиток*. Область определения Мужчины и область значений Напиток, является функциональной ролью, и обратная роль к ней тоже функциональна.
Стоит отметить, что у все шесть ролей фактически устанавливают между аргументами бинарное отношение «один-к-одному».

Protege:
В разделе «Object Properties» создаем 6 свойств (предикатов): «жить_в», «иметь_цвет», «иметь_животное», «курить_сигареты», «находиться_справа_от», «пить_напиток». При создании этих предикатов указываем сущности, которые они будут связывать, и их характеристики.



Для каждого свойства указываем домен (domain) и диапазон (range), определяющие, какие классы могут быть связаны этим свойством. Определяем характеристики свойств, такие как функциональность, транзитивность, симметричность, асимметричность, рефлексивность и иррефлексивность.

Определяем обратные свойства (inverse properties) для соответствующих пар свойств. Это позволяет выражать одни и те же отношения с разных точек зрения и упрощает формулирование запросов к онтологии.

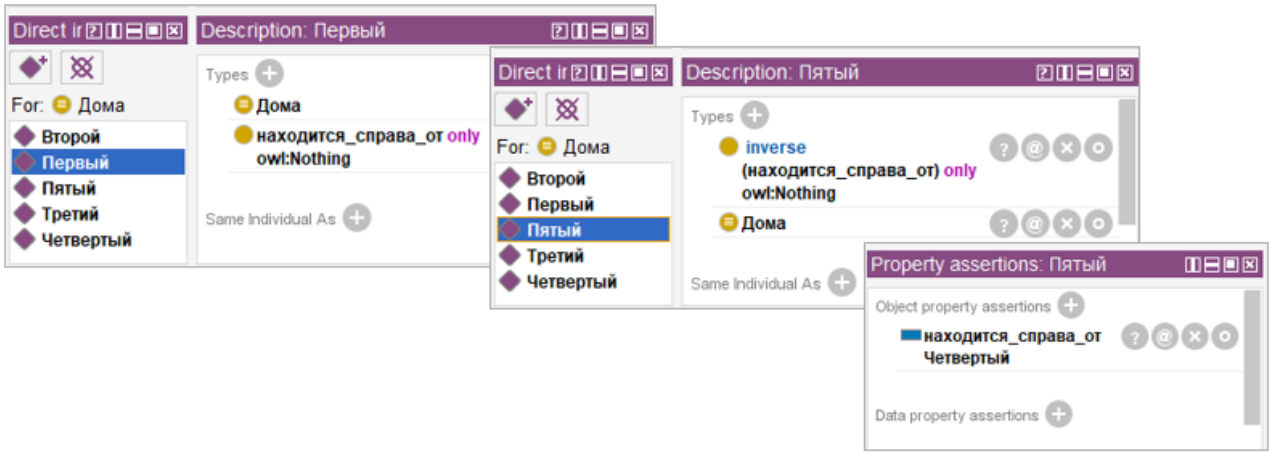
DL:

Теперь добавляем в DL-описание аксиомы, соответствующие известным фактам о наших сущностях. Первым делом добавим информацию о том, что дома стоят по порядку с 1 по 5. Для этого у объекта «Первый» укажем, что он стоит справа от пустого множества, а у объекта «Пятый» укажем, что он стоит справа от четвертого дома и слева от пустого множества.

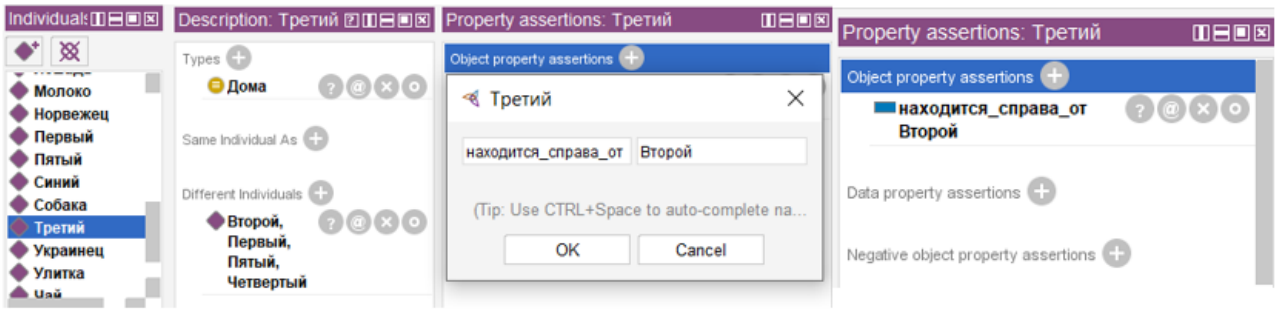
$\forall$ inverse(находится_справа_от). {Первый} $\sqsubseteq \bot$
$\forall$находится_справа_от. {Первый} $\sqsubseteq$ {Второй}
$\forall$находится_справа_от. {Второй} $\sqsubseteq$ {Третий}
$\forall$находится_справа_от. {Третий} $\sqsubseteq$ {Четвертый}

∀находится_справа_от. {Четвертый} ⊑ {Пятый}
∀находится_справа_от. {Пятый} ⊑ ⊥

Protege:



Для объектов «Второй», «Третий», «Четвертый» добавим информацию о том, что они находятся справа от предыдущего в разделе «Object property assertions».



Теперь приступаем к формализации условий задачи. Рассмотрим одно из сложных:

Условие 11. Сосед того, кто курит Chesterfield, держит лису.

DL:
К1≡Мужчины ⊓ ∃иметь_животное. {лиса}

(мужчины, имеющие животное лиса)

К2≡Дом ⊓ ∃ inverse(живет_в). К1

(дома, в которых живут мужчины, имеющие животное лиса)

К3≡Дом ⊓ ∃ находится_справа_от К2

К4≡Дом ⊓ ∃ inverse (находится_справа_от) К2

К5≡К3 ⊔ К4

(соседние дома, для домов, в которых живут мужчины, имеющие животное лиса)

К6≡Мужчины ⊓ ∃ живет_в. К5

(мужчины, которые живут в соседних домах для домов, в которых живут мужчины, имеющие животное лиса)

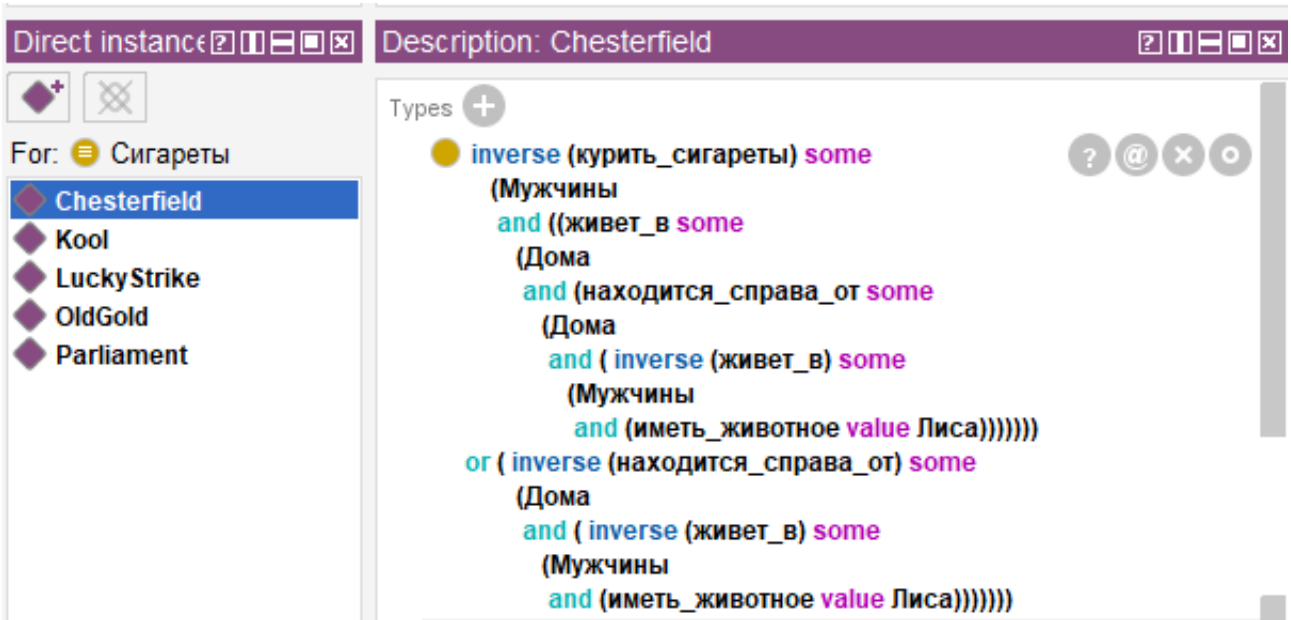
К7≡ Сигареты ⊓ ∃ inverse(курить_сигареты). К6

(Сигареты, которые курят мужчины, которые живут в соседних домах для домов, в которых живут мужчины, имеющие животное лиса)

К7 ⊑ {Chesterfield}

В приведенном тексте мы использовали служебное слово inverse для того, чтобы сделать марку сигарет субъектом высказывания, так как по логике выражения сигареты марки Chesterfield выкуриваются мужчиной, а не наоборот.

Protege:



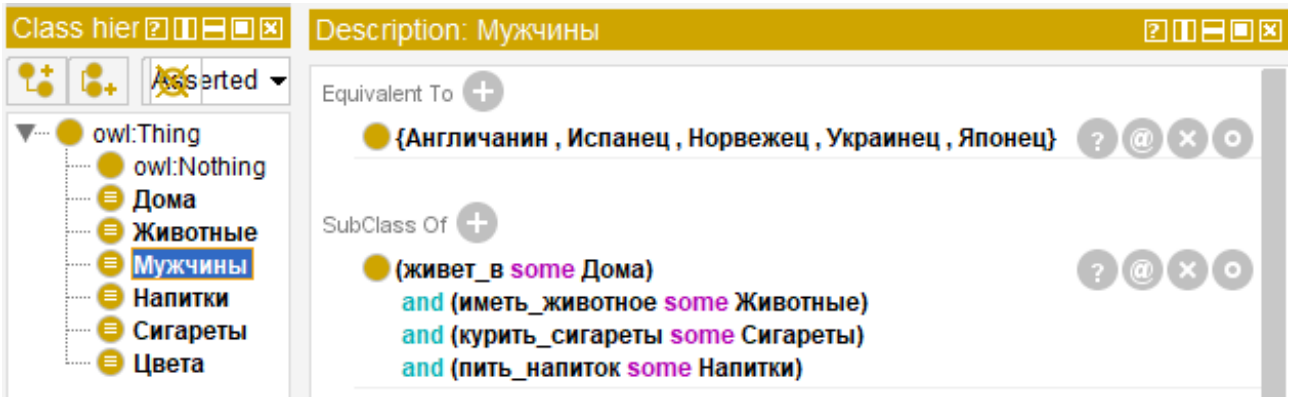
Аналогичным способом вносим информацию из остальных утверждений.

Для окончательной формализации необходимо в явном виде описать тот факт, что каждый мужчина обязательно живет в каком-то доме, пьет какой-то напиток, держит какое-то животное и курит сигареты какой-то марки.

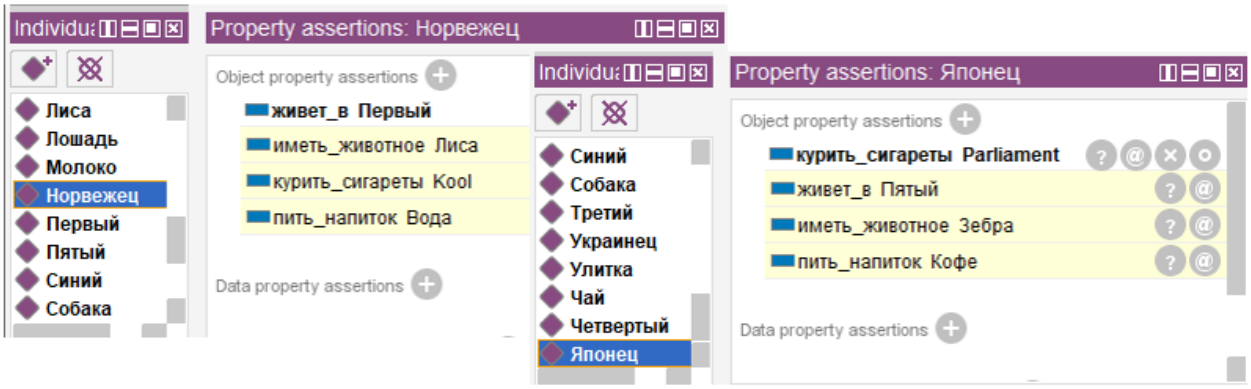
DL:

Мужчины ⊑ ∃ живет_в. Дома ⊓ ∃ иметь_животное. Животные ⊓ ∃ курит_сигареты. Сигареты ⊓ ∃ пить напиток.Напитки

Protege:



Теперь запустив ризонер в системе Protege, мы получили ответы на поставленные вопросы:



Теперь мы знаем, что Норвежец пьет воду, а Японец держит зебру.

Как мы видим, хорошие формализации в DL задач и построенные по ним онтологии могут пригодиться для практического поиска их решений.

Наиболее важным применением разработанным механизмам логического вывода в DL является применение программ-reaseners для логического анализа онтологий, который состоит из решения нескольких задач:

1) Проверка согласованности (consistency) онтологии, т.е. проверки, нет ли противоречий в определенных аксиомах и ограничениях. Если обнаружены противоречия, пересматриваем и корректируем соответствующие определения.
2) Классификация (classification) Классификация состоит в вычислении полной иерархии для всех определенных классов, включая выведенные (inferred) подклассы и эквивалентные классы. Это позволяет выявить неявные таксономические отношения, которые следуют из определенных аксиом и ограничений.

Например, для фрагмента1 онтологии SNOMED CT из примера 2 классификация даст следующую иерархию определенных классов:

$$\text{EntireDistalEpiphysisOfFemur} \sqsubseteq \text{StructureOfDistalEpiphysisOfFemur} \sqsubseteq \text{DistalFemurPart} \sqsubseteq \text{BoneStructureOfDistalFemur} \sqsubseteq \text{FemurPart}$$

3) Реализация (realization) онтологии. Реализация определяет наиболее специфичные классы, экземплярами которых являются индивиды, на основе их определенных свойств и аксиом онтологии. Это позволяет автоматически классифицировать индивиды по соответствующим классам.

ЛИТЕРАТУРА

1.Franz Baader and Diego Calvanese and eborah L. McGuinness and Daniele Nardi and Peter F. Patel-Schneider. The Description Logic Handbook: Theory, Implementation, and Applications. — Cambridge University Press, 2003. — ISBN 0-521-78176-0

2. Arne Meiera, Thomas Schneiderb. Generalized satisfiability for the description logic ALC. Theoretical Computer Science, 505 (2013) pp. 55-73

3.Рассел С., Норвиг П. Искусственный интеллект: современный подход. — М.: ИД «Вильямс», 2006. — 1408 с. — ISBN 5-8459-0887-6.

4.Введение в дедуктивную логику. Основные понятия логики и критерий правильности умозаключения – 2025 // Magisteria : [сайт]. – URL: https://magisteria.ru/introduction-to-deductive-logic/basic-concepts-of-logic-and-the-criterion-for-the-correctness-of-inference (дата обращения: 18.04.2025).

5.Применение онтологии к решению практических задач ИБ (часть1) URL: https://habr.com/ru/post/659425/

6. Description Logic: сайт. – 2023. – URL: https://www.larksuite.com/en_us/topics/ai-glossary/description-logic (дата обращения 19.02.2024).

7. Protégé 5 Documentation: сайт. – 2015. – URL: http://protegeproject.github.io/protege/ (дата обращения 12.04.2024).

8. List of Reasoners: сайт. – 2018. – URL: http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/ (дата обращения 25.04.2024).