Ontology engeneering
Разработка онтологий

Что такое "Ontology Engineering"?

Разработка онтологий —

это процесс определения терминов домена и отношений между ними:

- определение **концептов** домена (классов)

Что такое "Ontology Engineering"?

Разработка онтологий —

это процесс определения терминов домена и отношений между ними:

- определение **концептов** домена (классов)
- организация концептов в **иерархию** (подкласс-суперкласс)

Что такое "Ontology Engineering"?

Разработка онтологий —

это процесс определения терминов домена и отношений между ними:

- определение **концептов** домена (классов)
- организация концептов в **иерархию** (подкласс-суперкласс)
- определение **атрибутов** и **свойств** классов, а также **ограничений** на их значения

Что такое "Ontology Engineering"?

Разработка онтологий —

это процесс определения терминов домена и отношений между ними:

- определение **концептов** домена (классов)
- организация концептов в **иерархию** (подкласс-суперкласс)
- определение **атрибутов** и **свойств** классов, а также **ограничений** на их значения
- определение **индивидов** и придание значений атрибутам/свойствам.

Зачем разрабатывать онтологии?

- Чтобы добиться **общего понимания** терминов и для обмена информацией
  - между людьми
  - между программами

Зачем разрабатывать онтологии?

- Чтобы добиться **общего понимания** терминов и для обмена информацией
  - между людьми
  - между программами
- Для **повторного использования** знаний
  - не "изобретаем колесо"
  - стандарты

Зачем разрабатывать онтологии?

- Чтобы добиться **общего понимания** терминов и для обмена информацией
  - между людьми
  - между программами
- Для **повторного использования** знаний
  - не "изобретаем колесо"
  - стандарты
- Чтобы **явно** записать что мы знаем о какой-то области
  - эволюция знаний
  - легче понимать и поддерживать

Зачем разрабатывать онтологии?

- Чтобы добиться **общего понимания** терминов и для обмена информацией
  - между людьми
  - между программами
- Для **повторного использования** знаний
  - не "изобретаем колесо"
  - стандарты
- Чтобы **явно** записать что мы знаем о какой-то области
  - эволюция знаний
  - легче понимать и поддерживать
- Чтобы **отделить** знания об области от процедур
  - повторное использование знаний и повторное использование процедур

Разработка онтологий vs. ОО-моделирование

Онтология

ОО классы

Разработка онтологий vs. ОО-моделирование

Онтология

- отражает строение мира

ОО классы

- отражает построение кода и данных

Разработка онтологий vs. ОО-моделирование

Онтология

- отражает строение мира
- фокусируется на структуре концептов

ОО классы

- отражает построение кода и данных
- фокусируется на поведении (методы)

Разработка онтологий vs. ОО-моделирование

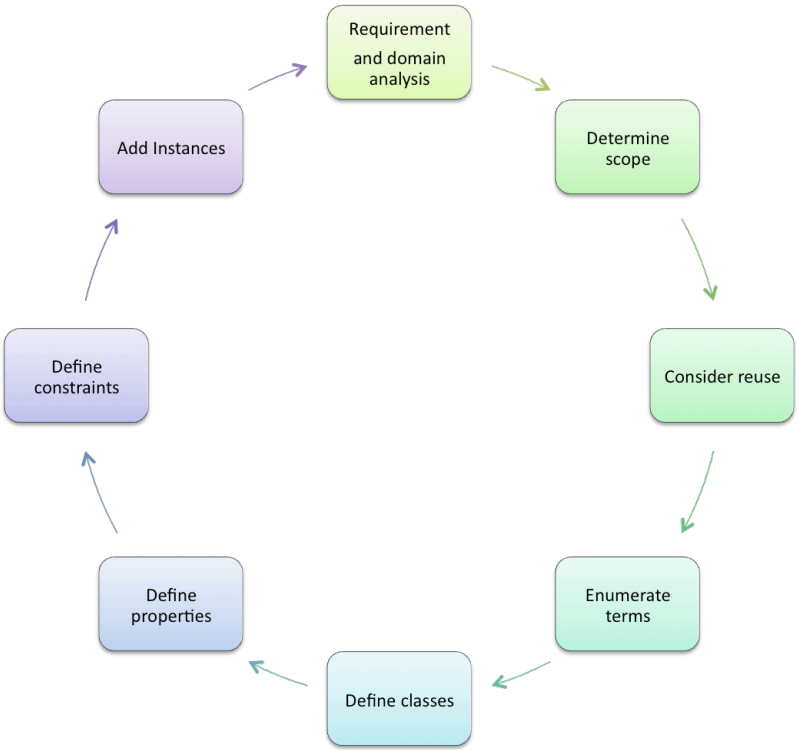
Онтология

- отражает строение мира
- фокусируется на структуре концептов
- физическое представление не важно

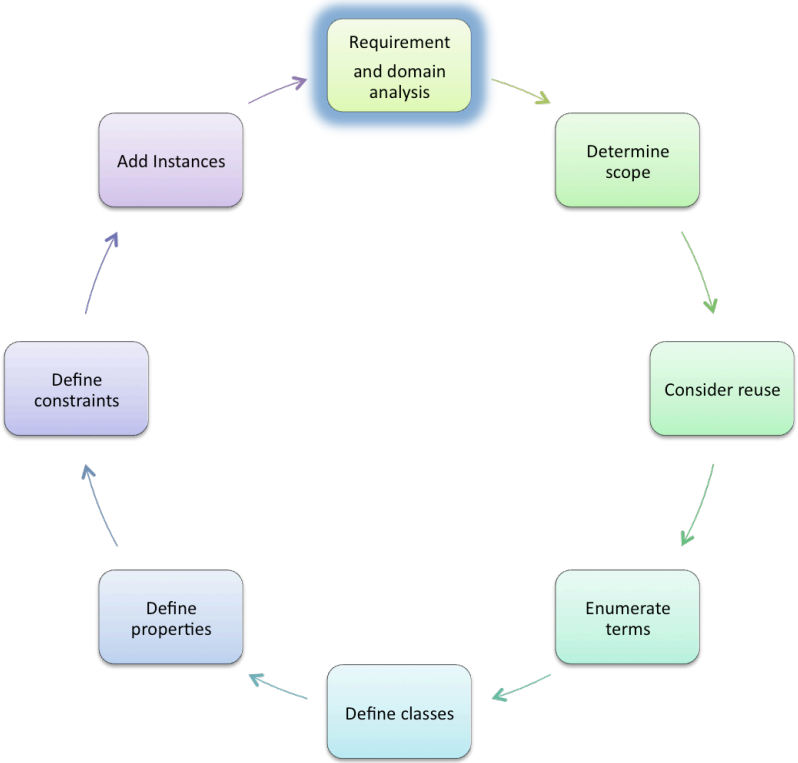
ОО классы

- отражает построение кода и данных
- фокусируется на поведении (методы)
- описывает физическое представление данных (int, char, etc.)

Цикл разработки онтологий



Цикл разработки онтологий



Анализ требований и области применения

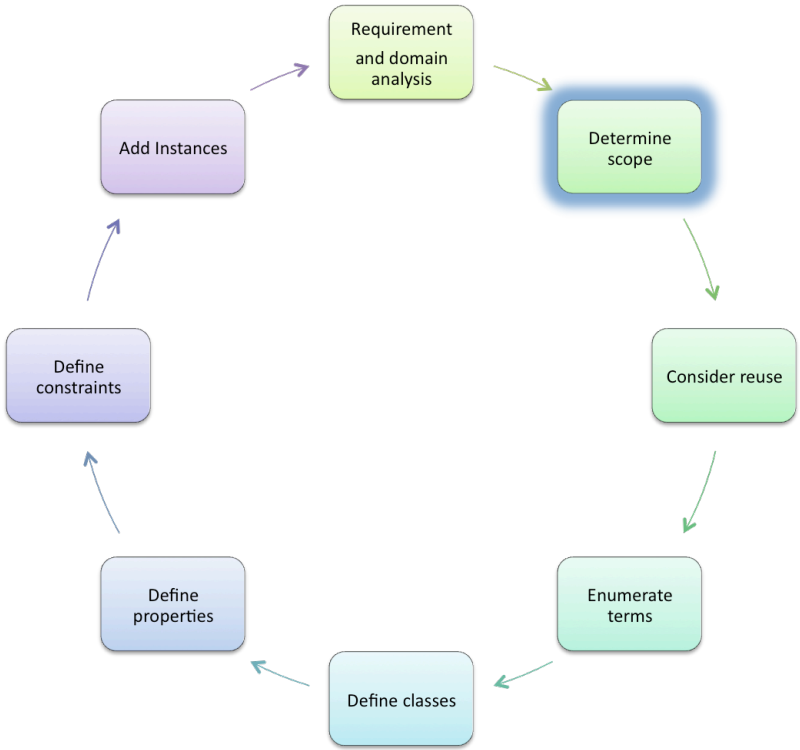
- Какую область знаний должна покрывать онтология?
- Какие термины необходимы в данной области?
- На какие **типы вопросов** информация, хранимая в онтологии, должна позволять найти ответ?
- Какая выразительная сила языка представления онтологий потребуется?
- Какие программные средства будут использованы?
- Гетерогенность, распределенность, автономность
- Гипотеза открытого мира vs гипотеза закрытого мира
- Статический или динамический процесс
- Ограниченные, неполные знания
- Анализ случаев
  - Какая информация доступна
  - Какие вопросы зададут
  - Типы и роли пользователей

Пример: онтология животных и растений

Онтология животных (и растений), упоминающихся в детских книжках, с целью создания предметного указателя. Должна включать:

- где живут
- что едят
  (хищное, травоядное, всеядное)
- опасны ли они
- большие ли они
- немного анатомии
  (количество ног, крыльев, пальцев, и т.п.)

Цикл разработки онтологий



Анализ области применения

- Для чего будет использована эта онтология?
  - **Онтология не должна содержать ВСЕ возможные знания о домене**
    * не требуется уточнять или обобщать более, чем необходимо
    * не требуется включать все возможные свойства классов

Анализ области применения

- Для чего будет использована эта онтология?
  - **Онтология не должна содержать ВСЕ возможные знания о домене**
    * не требуется уточнять или обобщать более, чем необходимо
    * не требуется включать все возможные свойства классов

Пример: онтология биологических экспериментов содержит

BiologicalOrganism и Experimenter.

Анализ области применения

- Для чего будет использована эта онтология?
  - **Онтология не должна содержать ВСЕ возможные знания о домене**
    * не требуется уточнять или обобщать более, чем необходимо
    * не требуется включать все возможные свойства классов

Пример: онтология биологических экспериментов содержит

BiologicalOrganism и Experimenter.

Должен ли класс Experimenter быть подклассом BiologicalOrganism?

Анализ области применения

- Для чего будет использована эта онтология?
  - **Онтология не должна содержать ВСЕ возможные знания о домене**
    * не требуется уточнять или обобщать более, чем необходимо
    * не требуется включать все возможные свойства классов

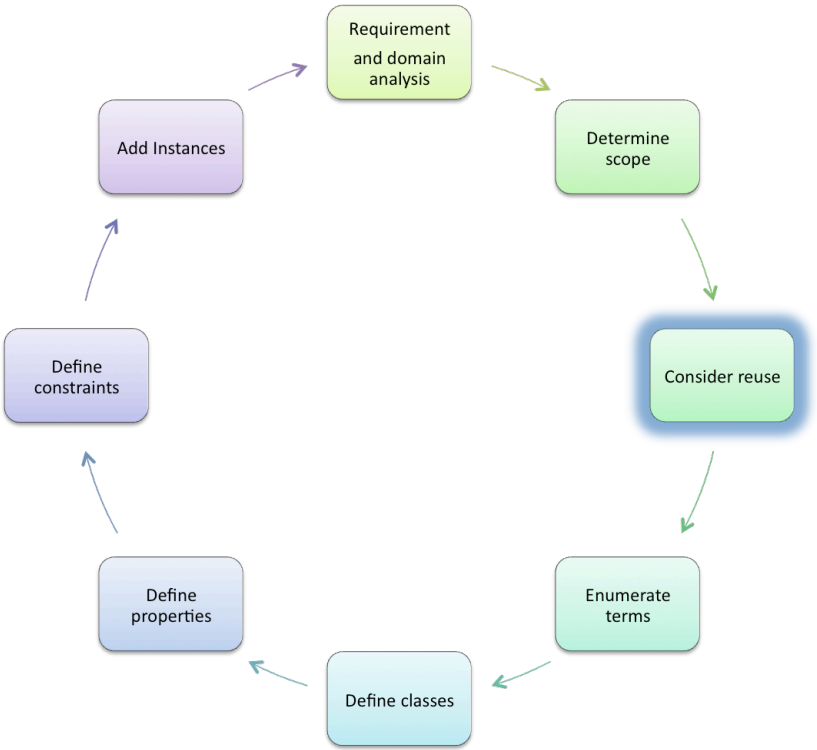
Пример: онтология биологических экспериментов содержит

BiologicalOrganism и Experimenter.

Должен ли класс Experimenter быть подклассом BiologicalOrganism?

  - **Предметный указатель**

Цикл разработки онтологий



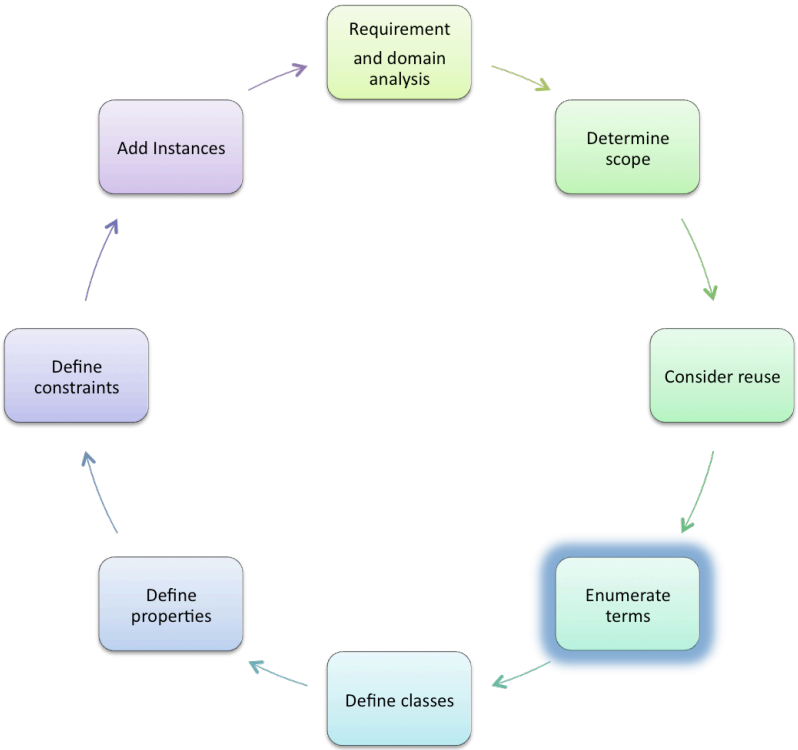
Повторное использование

- Редко начинаем на пустом месте
  - меньше усилий
  - средства разработки
  - проверенные временем
- Стандартные термины легко доступны
- Использовать модули

Где брать?

- Библиотеки онтологий
  - DAML ontology library (www.daml.org/ontologies)
  - Protégé ontology library (protege.stanford.edu/plugins.html)
- Онтологии верхнего уровня
  - IEEE Standard Upper Ontology (suo.ieee.org)
  - Cyc (www.cyc.com)
- Общие онтологии
  - DMOZ (www.dmoz.org)
  - WordNet (www.cogsci.princeton.edu/ wn/)
- Специальные онтологии
  - UMLS Semantic Net
  - GO (Gene Ontology) (www.geneontology.org)

Цикл разработки онтологий



Перечисление терминов

- собрать термины
  - какие термины
  - какие у них свойства?
  - что мы хотим сказать про эти термины?
- неформально сгруппировать
- **перефразировать** и уточнить термины, чтобы получить неформальные определения концептов
- нарисовать неформальную диаграмму

Сортировка карточек

- написать название концепции/класса/идеи на карточке
- сложить в кучки по смыслу
- понять связи между кучками
- повторить (можно в небольшой группе людей)

Пример

Dog	Carnivore	Dangerous
Cat	Plant	Pet
Cow	Animal	Domestic Animal
Person	Draught Animal[a]	Farm Animal
Tree	Child	Food Animal
Grass	Parent	Fish
Herbivore	Mother	Carp
Male	Father	Goldfish
Female	Pig	

[a] ездовые и вьючные животные

Пример

Dog	Carnivore	Dangerous
Cat	Plant	Pet
Cow	Animal	Domestic Animal
Person	Draught Animal[a]	Farm Animal
Tree	Child	Food Animal
Grass	Parent	Fish
Herbivore	Mother	Carp
Male	Father	Goldfish
Female	Pig	

[a] ездовые и вьючные животные

Группировать, добавлять и обобщать

Для группы вещей/классов/идей и задаться вопросом **что у них общего**

и какие '**родственники**' у них есть.

Группировать, добавлять и обобщать

Для группы вещей/классов/идей и задаться вопросом **что у них общего**

и какие '**родственники**' у них есть.

Например

- Plant, Animal

Группировать, добавлять и обобщать

Для группы вещей/классов/идей и задаться вопросом **что у них общего**

и какие '**родственники**' у них есть.

Например

- Plant, Animal — Живое Существо (а как быть с Bacteria, Fungi?)

Группировать, добавлять и обобщать

Для группы вещей/классов/идей и задаться вопросом **что у них общего**

и какие '**родственники**' у них есть.

Например

- Plant, Animal — Живое Существо (а как быть с Bacteria, Fungi?)
- Cat, Dog, Cow, Person

Группировать, добавлять и обобщать

Для группы вещей/классов/идей и задаться вопросом **что у них общего**

и какие '**родственники**' у них есть.

Например

- Plant, Animal — Живое Существо (а как быть с Bacteria, Fungi?)
- Cat, Dog, Cow, Person — Млекопитающее (добавить Goat, Rabbit?)

Группировать, добавлять и обобщать

Для группы вещей/классов/идей и задаться вопросом **что у них общего**

и какие '**родственники**' у них есть.

Например

- Plant, Animal — Живое Существо (а как быть с Bacteria, Fungi?)
- Cat, Dog, Cow, Person — Млекопитающее (добавить Goat, Rabbit?)
- Cow, Goat, Sheep, Horse

Группировать, добавлять и обобщать

Для группы вещей/классов/идей и задаться вопросом **что у них общего**

и какие '**родственники**' у них есть.

Например

- Plant, Animal — Живое Существо (а как быть с Bacteria, Fungi?)
- Cat, Dog, Cow, Person — Млекопитающее (добавить Goat, Rabbit?)
- Cow, Goat, Sheep, Horse — Копытное

(кто они такие? ? как они делятся/группируются? четное/нечетное число пальцев?)

Группировать, добавлять и обобщать

Для группы вещей/классов/идей и задаться вопросом **что у них общего**

и какие '**родственники**' у них есть.

Например

- Plant, Animal — Живое Существо (а как быть с Bacteria, Fungi?)
- Cat, Dog, Cow, Person — Млекопитающее (добавить Goat, Rabbit?)
- Cow, Goat, Sheep, Horse — Копытное

  (кто они такие? ? как они делятся/группируются? четное/нечетное число пальцев?)
- Wild, Domestic

Группировать, добавлять и обобщать

Для группы вещей/классов/идей и задаться вопросом **что у них общего**

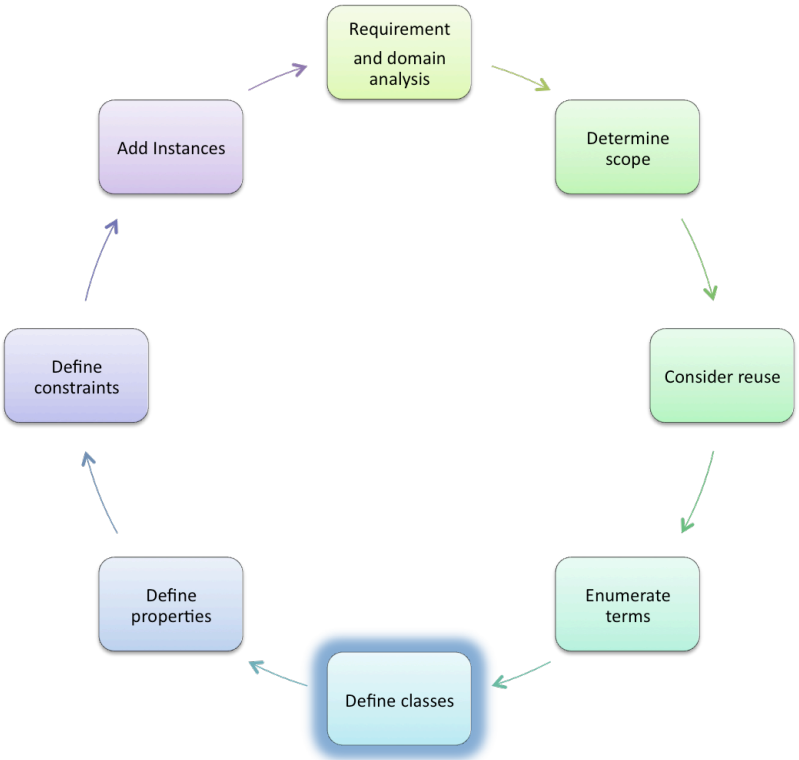
и какие '**родственники**' у них есть.

Например

- Plant, Animal — Живое Существо (а как быть с Bacteria, Fungi?)
- Cat, Dog, Cow, Person — Млекопитающее (добавить Goat, Rabbit?)
- Cow, Goat, Sheep, Horse — Копытное

  (кто они такие? ? как они делятся/группируются? четное/нечетное число пальцев?)
- Wild, Domestic — Одомашненные (другие стадии?)

Цикл разработки онтологий



Задать классы

выбрать **основные направления**:

- добавить **абстракции** где необходимо

(например, Living Thing, Mammal, Fish)

Задать классы

выбрать **основные направления**:

- добавить **абстракции** где необходимо (например, Living Thing, Mammal, Fish)
- определить **отношения** (например, eats, owns, parent of)

Задать классы

выбрать **основные направления**:

- добавить **абстракции** где необходимо (например, Living Thing, Mammal, Fish)
- определить **отношения** (например, eats, owns, parent of)
- идентифицировать какие **понятия определяются** (e.g., Draught Animal, Father, Herbivore)

как определить собаку?

Задать классы

выбрать **основные направления**:

- добавить **абстракции** где необходимо (например, Living Thing, Mammal, Fish)
- определить **отношения** (например, eats, owns, parent of)
- идентифицировать какие **понятия определяются** (e.g., Draught Animal, Father, Herbivore)

как определить собаку?

Сущности vs. модификаторы

- **сущности** существуют сами по себе (грубо, существительные)

(например, people, animals, houses, actions, processes)

- **модификаторы** ‘меняют’ другие вещи (грубо, прилагательные, наречия,…)

(например, wild/domestic, male/female, healthy/sick, dangerous/safe)

Организация концептов/свойств в иерархию

Собрать все, кроме определяемых понятий в **деревья** —

это будут "примитивы"

Организация концептов/свойств в иерархию

Собрать все, кроме определяемых понятий в **деревья** —

это будут "примитивы"

сущности	модификаторы	отношения	определяемы
– LivingThing	Domestication	eats	Carnivore
– Animal	– Domestic	owns	Herbivore
– Mammal	– Wild	parentOf	Child
– Cat	Use	...	Parent
– Dog	– Pet		Mother
– Cow	– Food		Father
– Person	– Draught		FoodAnimal
– Pig	Dangerousness		DraughtAnimal
– Fish	– Dangerous		
– Carp	– Safe		
– Goldfish	Sex		
– Plant	– Male		
– Tree	– Female		
– Grass	Age		
	– Adult		
	– Child		

Классы и иерархия классов

- Важно знать
  - не существует **единственной верной** иерархии классов
  - но есть некоторые **рекомендации**

Классы и иерархия классов

- Важно знать
  - не существует **единственной верной** иерархии классов
  - но есть некоторые **рекомендации**

- Вопросы:

является ли каждый экземпляр подкласса экземпляром суперкласса?

Классы и иерархия классов

- Все **родственные понятия** в иерархии классов должны быть на **одном уровне**

(ср. с разделами и главами книг)

Классы и иерархия классов

- Все **родственные понятия** в иерархии классов должны быть на **одном уровне**

  (ср. с разделами и главами книг)

- Если у класс более **десяти** непосредственных подклассов,

  дополнительные **подкатегории** могут быть необходимы

  (ср. со списками)

  Если нет естественного подразбиения, длинный список может быть более естественным

Классы и иерархия классов

- Все **родственные понятия** в иерархии классов должны быть на **одном уровне**

  (ср. с разделами и главами книг)

- Если у класс более **десяти** непосредственных подклассов,

  дополнительные **подкатегории** могут быть необходимы

  (ср. со списками)

  Если нет естественного подразбиения, длинный список может быть более естественным

- Имена классов должны быть или **все в единственном числе** или **все в множественном числе**

  (*Animal* is **not a kind-of** *Animals*)

Классы и иерархия классов

- Все **родственные понятия** в иерархии классов должны быть на **одном уровне**

  (ср. с разделами и главами книг)

- Если у класс более **десяти** непосредственных подклассов,

  дополнительные **подкатегории** могут быть необходимы

  (ср. со списками)

  Если нет естественного подразбиения, длинный список может быть более естественным

- Имена классов должны быть или **все в единственном числе** или **все в множественном числе**

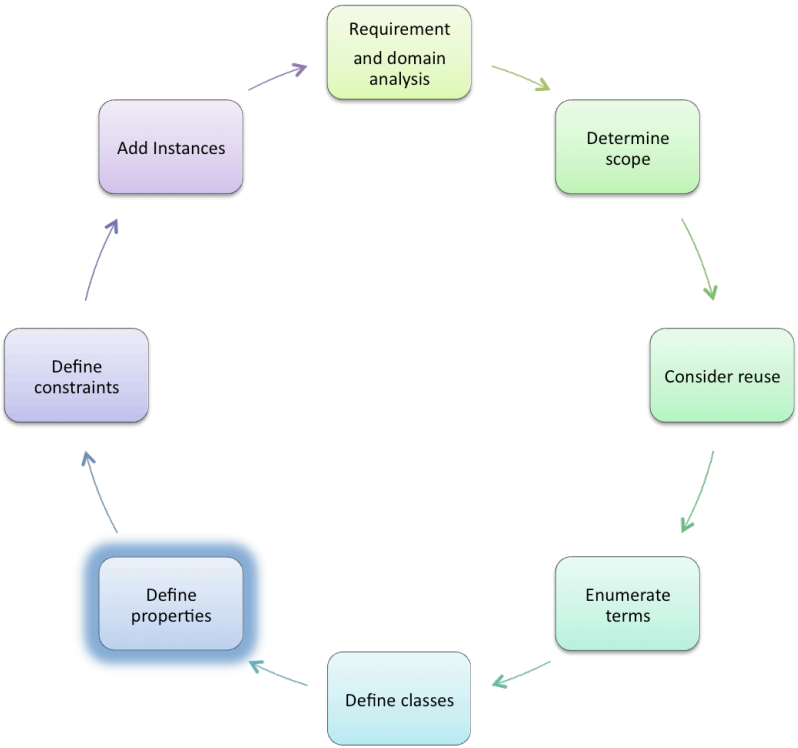
  (*Animal* is **not a kind-of** *Animals*)

- Классы представляют концепции области знаний, не их названия!

  название может меняться, но будет означать ту же концепцию.

  (синонимы для той же концепции не представляются разными классами)

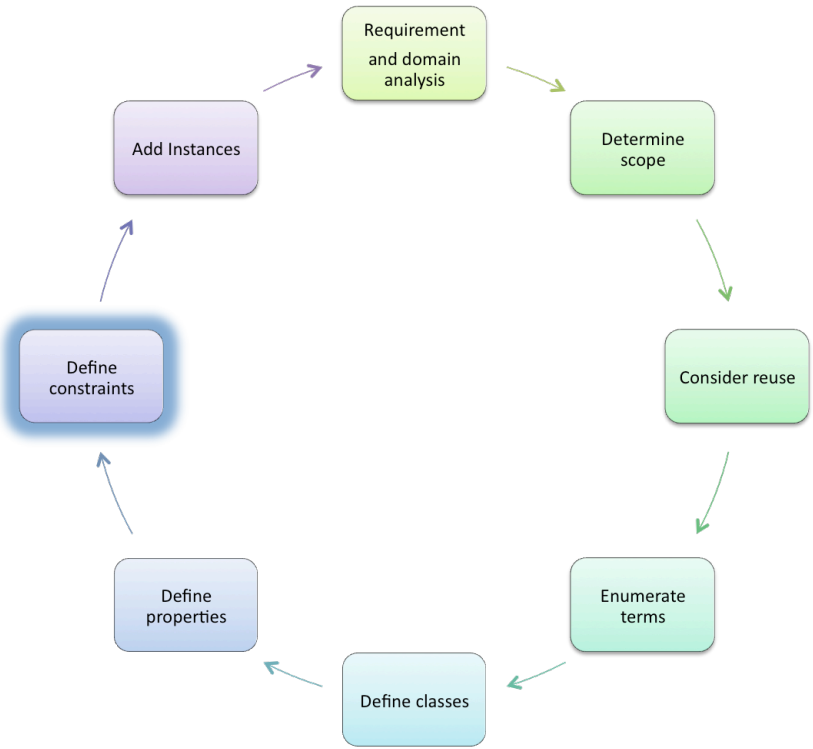
Цикл разработки онтологий



Свойства

- Перекликается с предыдущим шагом
- Свойства (роли) описывают атрибуты членов класса
- Если A подкласс B, то каждое свойство, которое имеют элементы B, будет применяться к элементам A.
  - Задавать свойства для самого высокого в иерархии класса, к которому применяются
- "присущие свойства": цвет, запах,...
- "внешние свойства": название и цена

Цикл разработки онтологий



Ограничения

Ограничения на **область определения** и **принимаемые значения** для свойств

(если что-то используется каким-то специальным образом, оставить текстовый комментарий)

Ограничения

Ограничения на **область определения** и **принимаемые значения** для свойств

(если что-то используется каким-то специальным образом, оставить текстовый комментарий)

- Animal *eats* LivingThing

domain: Animal
range: LivingThing

(NB: игнорируем разницу между частями LivingThings и LivingThings)

Ограничения

Ограничения на **область определения** и **принимаемые значения** для свойств

(если что-то используется каким-то специальным образом, оставить текстовый комментарий)

- Animal *eats* LivingThing

  domain: Animal
  range: LivingThing

(NB: игнорируем разницу между частями LivingThings и LivingThings)

- Person *owns* LivingThing except Person

  domain: Person
  range: LivingThing
  and not Person

Ограничения

Ограничения на **область определения** и **принимаемые значения** для свойств

(если что-то используется каким-то специальным образом, оставить текстовый комментарий)

- Animal *eats* LivingThing

  domain: Animal
  range: LivingThing

(NB: игнорируем разницу между частями LivingThings и LivingThings)

- Person *owns* LivingThing except Person

  domain: Person
  range: LivingThing
  and not Person

- Animal *parentOf* Animal

  domain: Animal
  range: Animal

Ограничения

Ограничения на **область определения** и **принимаемые значения** для свойств

(если что-то используется каким-то специальным образом, оставить текстовый комментарий)

- Animal *eats* LivingThing

  domain: Animal
  range: LivingThing

(NB: игнорируем разницу между частями LivingThings и LivingThings)

- Person *owns* LivingThing except Person

  domain: Person
  range: LivingThing
  and not Person

- Animal *parentOf* Animal

  domain: Animal
  range: Animal

Идентифицировать **ограничения на свойства**: что можно сказать о **всех примерах класса**?

Ограничения

Ограничения на **область определения** и **принимаемые значения** для свойств

(если что-то используется каким-то специальным образом, оставить текстовый комментарий)

- Animal *eats* LivingThing

  domain: Animal
  range: LivingThing

(NB: игнорируем разницу между частями LivingThings и LivingThings)

- Person *owns* LivingThing except Person

  domain: Person
  range: LivingThing
  and not Person

- Animal *parentOf* Animal

  domain: Animal
  range: Animal

Идентифицировать **ограничения на свойства**: что можно сказать о **всех примерах класса**?

- all Cows *eat* some Plants
- all Cats *eat* some Animals
- all Pigs *eat* some Animals and *eat* some Plants
- ...

Ограничения

Ограничения на **область определения** и **принимаемые значения** для свойств

(если что-то используется каким-то специальным образом, оставить текстовый комментарий)

- Animal *eats* LivingThing

  domain: Animal
  range: LivingThing

(NB: игнорируем разницу между частями LivingThings и LivingThings)

- Person *owns* LivingThing except Person

  domain: Person
  range: LivingThing
  and not Person

- Animal *parentOf* Animal

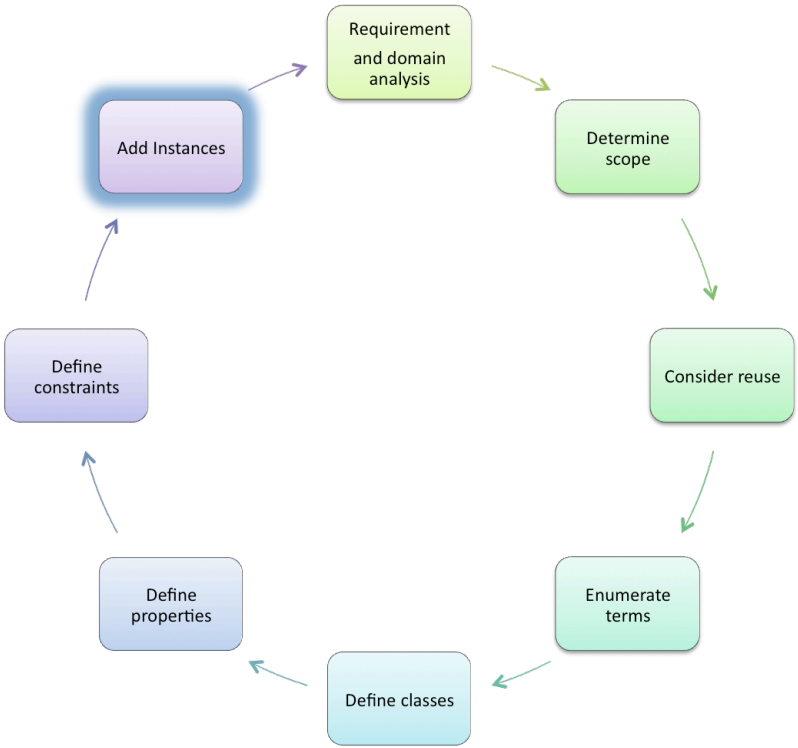
  domain: Animal
  range: Animal

Идентифицировать **ограничения на свойства**: что можно сказать о **всех примерах класса**?

- all Cows *eat* some Plants
- all Cats *eat* some Animals
- all Pigs *eat* some Animals and *eat* some Plants
- ...

описания
сущностей

Цикл разработки онтологий



Определяемые понятия

Перефразировать и **формализовать** **определения** в терминах
примитивов, отношений и других определяемых понятий

оставлять заметки **на будущее**

(добавить комментарии)

- "A Parent is an Animal that is a *parent of* some other Animal"

(NB: пока забудем про Plants)

Определяемые понятия

Перефразировать и **формализовать** **определения** в терминах
примитивов, отношений и других определяемых понятий

оставлять заметки **на будущее**

(добавить комментарии)

- "A Parent is an Animal that is a *parent of* some other Animal"

(NB: пока забудем про Plants)

Parent = *Animal* and *parentOf* some *Animal*

Определяемые понятия

Перефразировать и **формализовать** **определения** в терминах примитивов, отношений и других определяемых понятий

оставлять заметки **на будущее**

(добавить комментарии)

- "A Parent is an Animal that is a *parent of* some other Animal"

(NB: пока забудем про Plants)

Parent = *Animal* and *parentOf* some *Animal*

- "A Herbivore is an Animal that *eats* only Plants"

(NB: all Animals *eat* some LivingThings)

Определяемые понятия

Перефразировать и **формализовать** **определения** в терминах примитивов, отношений и других определяемых понятий

оставлять заметки **на будущее**

(добавить комментарии)

- "A Parent is an Animal that is a *parent of* some other Animal"

(NB: пока забудем про Plants)

Parent = *Animal* and *parentOf* some *Animal*

- "A Herbivore is an Animal that *eats* only Plants"

(NB: all Animals *eat* some LivingThings)

Herbivore = *Animal* and *eats* only *Plant*

Определяемые понятия

Перефразировать и **формализовать** **определения** в терминах
примитивов, отношений и других определяемых понятий

оставлять заметки **на будущее**

(добавить комментарии)

- "A Parent is an Animal that is a *parent of* some other Animal"

(NB: пока забудем про Plants)

Parent = *Animal* and *parentOf* some *Animal*

- "A Herbivore is an Animal that *eats* only Plants"

(NB: all Animals *eat* some LivingThings)

Herbivore = *Animal* and *eats* only *Plant*

- "An Omnivore is an Animal that *eats* both Plants and Animals"

Определяемые понятия

Перефразировать и **формализовать** **определения** в терминах
примитивов, отношений и других определяемых понятий

оставлять заметки **на будущее**

(добавить комментарии)

- "A Parent is an Animal that is a *parent of* some other Animal"

(NB: пока забудем про Plants)

Parent = *Animal* and *parentOf* some *Animal*

- "A Herbivore is an Animal that *eats* only Plants"

(NB: all Animals *eat* some LivingThings)

Herbivore = *Animal* and *eats* only *Plant*

- "An Omnivore is an Animal that *eats* both Plants and Animals"

Omnivore = *Animal* and *eats* some *Plant* and *eats* some *Animal*

Определяемые понятия

Перефразировать и **формализовать** **определения** в терминах
примитивов, отношений и других определяемых понятий

оставлять заметки **на будущее**

(добавить комментарии)

- "A Parent is an Animal that is a *parent of* some other Animal"

(NB: пока забудем про Plants)

Parent = *Animal* and *parentOf* some *Animal*

- "A Herbivore is an Animal that *eats* only Plants"

(NB: all Animals *eat* some LivingThings)

Herbivore = *Animal* and *eats* only *Plant*

- "An Omnivore is an Animal that *eats* both Plants and Animals"

Omnivore = *Animal* and *eats* some *Plant* and *eats* some *Animal*

Не перефразируя невозможно определить не разошлись ли смыслы
того что **хотели** представить и того, что **представили**.

Нормализация

Дерево

все узлы (кроме корня)
имеют одного родителя

‘строгая иерархия’

Направленный ациклический граф (DAG)

узлы могут иметь
несколько родителей

‘полииерархия’

Нормализация

Дерево

все узлы (кроме корня)
имеют одного родителя

'строгая иерархия'

Направленный ациклический граф (DAG)

узлы могут иметь
несколько родителей

'полииерархия'

Нормализация:

- разделить *примитивы* в **непересекающиеся деревья**

Нормализация

Дерево

все узлы (кроме корня)
имеют одного родителя

'строгая иерархия'

Направленный ациклический граф (DAG)

узлы могут иметь
несколько родителей

'полииерархия'

Нормализация:

- разделить *примитивы* в **непересекающиеся деревья**
- пусть **классификатор** создает DAG

Нормализация

Дерево

все узлы (кроме корня)
имеют одного родителя

'строгая иерархия'

Направленный ациклический граф (DAG)

узлы могут иметь
несколько родителей

'полииерархия'

Нормализация:

- разделить *примитивы* в **непересекающиеся деревья**
- пусть **классификатор** создает DAG

Деревья легче понять

Нормализация

Дерево

все узлы (кроме корня) имеют одного родителя

'строгая иерархия'

Направленный ациклический граф (DAG)

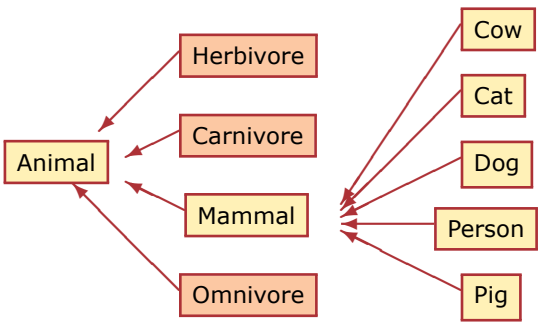
узлы могут иметь несколько родителей

'полииерархия'

Нормализация:

- разделить *примитивы* в **непересекающиеся деревья**
- пусть **классификатор** создает DAG

Деревья легче понять



Нормализация

Дерево

все узлы (кроме корня)
имеют одного родителя

'строгая иерархия'

Направленный ациклический граф (DAG)

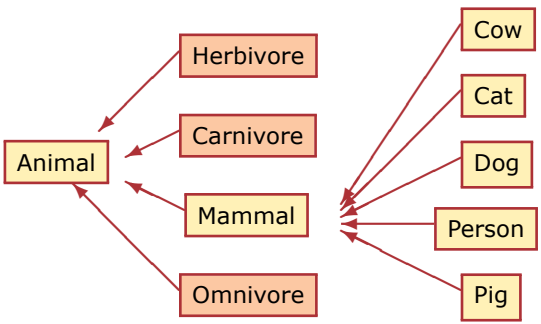
узлы могут иметь
несколько родителей

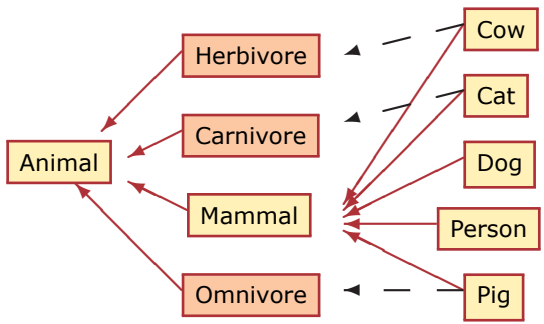
'полииерархия'

Нормализация:

- разделить *примитивы* в **непересекающиеся деревья**
- пусть **классификатор** создает DAG

Деревья легче понять





Модификаторы

- Идентифицировать модификаторы с

  взаимно исключающими значениями

Domestication
- Domestic
- Wild

Use
- Pet
- Food
- Draught

Dangerousness
- Dangerous
- Safe

Sex
- Male
- Female

Age
- Adult
- Child

Модификаторы

- Идентифицировать модификаторы с

  взаимно исключающими значениями
  (Domestication, Dangerousness, Sex, Age)

  NB. Применения могут быть не взаимно исключающими
  (может быть одновременно Вьючное и Еда)

Domestication
- Domestic
- Wild

Use
- Pet
- Food
- Draught

Dangerousness
- Dangerous
- Safe

Sex
- Male
- Female

Age
- Adult
- Child

Модификаторы

- Идентифицировать модификаторы с

  взаимно исключающими значениями

  (Domestication, Dangerousness, Sex, Age)

  NB. Применения могут быть не взаимно исключающими

  (может быть одновременно Вьючное и Еда)

- Расширить и дополнить список значений

  (Dangerousness: Dangerous, Risky, Safe)

Domestication
- Domestic
- Wild

Use
- Pet
- Food
- Draught

Dangerousness
- Dangerous
- Safe

Sex
- Male
- Female

Age
- Adult
- Child

Модификаторы

- Идентифицировать модификаторы с

  взаимно исключающими значениями

  (Domestication, Dangerousness, Sex, Age)

  NB. Применения могут быть не взаимно исключающими

  (может быть одновременно Вьючное и Еда)

- Расширить и дополнить список значений

  (Dangerousness: Dangerous, Risky, Safe)

- Задать как **функциональное свойство**

Domestication
- Domestic
- Wild

Use
- Pet
- Food
- Draught

Dangerousness
- Dangerous
- Safe

Sex
- Male
- Female

Age
- Adult
- Child

Модификаторы

- Идентифицировать модификаторы с

  взаимно исключающими значениями

  (Domestication, Dangerousness, Sex, Age)

  NB. Применения могут быть не взаимно исключающими

  (может быть одновременно Вьючное и Еда)

- Расширить и дополнить список значений

  (Dangerousness: Dangerous, Risky, Safe)

- Задать как **функциональное свойство**

- Два способа задать значения модификаторов
  - **разбиения** (классы разбивают величину)
  - **перечисления**

    (индивиды для всех степеней величины)

Domestication
- Domestic
- Wild

Use
- Pet
- Food
- Draught

Dangerousness
- Dangerous
- Safe

Sex
- Male
- Female

Age
- Adult
- Child

Разбиения

Пример: величина — Dangerousness

Разбиения

Пример: величина — Dangerousness

- Определить **под-величины** для разных степеней: Dangerous, Risky, Safe
  - классы **дизъюнктны**
  - ‘**покрывают**’ исходную величину т.е.,

Dangerousness = *Dangerous* or *Risky* or *Safe*

Разбиения

Пример: величина — Dangerousness

- Определить **под-величины** для разных степеней: Dangerous, Risky, Safe
  - классы **дизъюнктны**
  - '**покрывают**' исходную величину т.е.,

Dangerousness = *Dangerous* or *Risky* or *Safe*

- Определить **функциональное свойство** *hasDangerousness*
  - область значений: исходная величина, например, Dangerousness
  - домен задается отдельно

Разбиения

Пример: величина — Dangerousness

- Определить **под-величины** для разных степеней: Dangerous, Risky, Safe
  - классы **дизъюнктны**
  - '**покрывают**' исходную величину т.е.,

Dangerousness = *Dangerous* or *Risky* or *Safe*

- Определить **функциональное свойство** *hasDangerousness*
  - область значений: исходная величина, например, Dangerousness
  - домен задается отдельно

DangerousAnimal = *Animal* and *hasDangerousness* some *Dangerous*

Перечисления

Пример: величина — SexValue

Перечисления

Пример: величина — SexValue

- Определим **индивиды** для каждого значения: male, female
  - значения **различны** (**НЕ предполагается в OWL**)
  - значением является '**перечисление**' значений, т.е.,

$$SexValue = \{ female, male \}$$

Перечисления

Пример: величина — SexValue

- Определим **индивиды** для каждого значения: male, female
  - значения **различны** (**НЕ предполагается в OWL**)
  - значением является '**перечисление**' значений, т.е.,

$$SexValue = \{ female, male \}$$

- Определим **функциональное свойство** *hasSex*
  - область значений: исходное свойство т.е. SexValue
  - домен задается дополнительно

Перечисления

Пример: величина — SexValue

- Определим **индивиды** для каждого значения: male, female
  - значения **различны** (**НЕ предполагается в OWL**)
  - значением является '**перечисление**' значений, т.е.,

$$SexValue = \{ female, male \}$$

- Определим **функциональное свойство** *hasSex*
  - область значений: исходное свойство т.е. SexValue
  - домен задается дополнительно

$$MaleAnimal = Animal \text{ and } hasSex \text{ is } male$$

Два способа задать значения

Разбиение

Перечисление

Два способа задать значения

Разбиение

- может быть разбито далее и уточнено

Перечисление

- не может быть разбито далее

Два способа задать значения

Разбиение

- может быть разбито далее и уточнено
- соответствует философскому понятию пространства величин

Перечисление

- не может быть разбито далее
- соответствует интуиции

Два способа задать значения

Разбиение

- может быть разбито далее и уточнено
- соответствует философскому понятию пространства величин

Перечисление

- не может быть разбито далее
- соответствует интуиции

Два способа задать значения

Разбиение

- может быть разбито далее и уточнено
- соответствует философскому понятию пространства величин
- больше подходит для использования с OWL DL классификаторами

Перечисление

- не может быть разбито далее
- соответствует интуиции
- не очень подходит для использования с существующими классификаторами

Поиск объяснений

Зачем нужны объяснения?

Поиск объяснений это техника, помогающая поддерживать онтологии.

Пример Ранняя версия SNOMED CT (в которой > 300000 аксиом) утверждала

$$\textbf{Amputation\_of\_Finger} \sqsubseteq \textbf{Amputation\_of\_Arm}$$

Задача Найти аксиомы (среди 300 000 аксиом SNOMED) ответственные за этот вывод.

Определение

Рассмотрим TBox $\mathcal{T}$ и концепты C, D т.ч.

$$\mathcal{T} \models C \sqsubseteq D.$$

Множество объяснений $\mathbf{Pin}(\mathcal{T}, C \sqsubseteq D)$ для $\mathcal{T}$ и $C \sqsubseteq D$ содержит все минимальные подмножества $\mathcal{T}'$ из $\mathcal{T}$ т.ч.

$$\mathcal{T}' \models C \sqsubseteq D.$$

Пример

$$\mathcal{T} = \{\mathbf{Father} \sqsubseteq \mathbf{Male}, \mathbf{Male} \sqsubseteq \mathbf{Human}, \mathbf{Car} \sqsubseteq \mathbf{Vehicle}\}$$

Множество $\mathbf{Pin}(\mathcal{T}, \mathbf{Father} \sqsubseteq \mathbf{Human}\}$ состоит из единственного множества

$$\{\mathbf{Father} \sqsubseteq \mathbf{Male}, \mathbf{Male} \sqsubseteq \mathbf{Human}\}$$

Пример

Рассмотрим

$$\mathcal{T} = \{A \sqsubseteq B, B \sqsubseteq E, A \sqsubseteq F, F \sqsubseteq E, C \sqsubseteq D\}$$

Тогда $\mathcal{T} \models A \sqsubseteq E$.

Имеется два множества объяснений $\mathbf{Pin}(\mathcal{T}, A \sqsubseteq E\}$:

$$\{A \sqsubseteq B, B \sqsubseteq E\}$$

и

$$\{A \sqsubseteq F, F \sqsubseteq E\}.$$

Пример

Рассмотрим TBox

a **Human** $\sqsubseteq$ $\exists$**child_of.Human**;

b **Human** $\sqsubseteq$ **Living_Being**;

c $\exists$**child_of.Living_Being** $\sqsubseteq$ **Has_Birthday**;

d **Living_Being** $\sqsubseteq$ **Has_Birthday**.

Тогда $\mathcal{T} \models$ **Human** $\sqsubseteq$ **Has_Birthday**. Множество **Pin**$(\mathcal{T}, $**Human** $\sqsubseteq$ **Has_Birthday**$)$ состоит из

$$\{b, d\}$$

и

$$\{a, b, c\}$$

Вычисление

- Множество **Pin**$(\mathcal{T}, C \sqsubseteq D)$ может содержать экспоненциально много объяснений;
- Проверка существует ли элемент **Pin**$(\mathcal{T}, C \sqsubseteq D)$ размера $\leq n$ трудна даже для $\mathcal{EL}$;
- Найти **один** член **Pin**$(\mathcal{T}, C \sqsubseteq D)$ не труднее классификации
  - В частности, полиномиальный алгоритм поиска одного члена **Pin**$(\mathcal{T}, C \sqsubseteq D)$ для $\mathcal{T}$, C и D в $\mathcal{EL}$.

Алгоритм поиска одного члена **Pin**$(\mathcal{T}, C \sqsubseteq D)$

Вход $\mathcal{T} = \{\alpha_1, \ldots, \alpha_n\}$ и $C \sqsubseteq D$.

1. if $C \not\sqsubseteq_{\mathcal{T}} D$, then
2. return **Pin**$(\mathcal{T}, C \sqsubseteq D)$ empty
3. set $\mathcal{S} := \mathcal{T}$
4. for $1 \leq i \leq n$ do
5. if $C \sqsubseteq_{\mathcal{S} \setminus \{\alpha_i\}} D$ then
6. $\mathcal{S} := \mathcal{S} \setminus \{\alpha_i\}$
7. return $\mathcal{S}$.

Лаконичные объяснения

Множество объяснение может быть нелегко понять.

TBox

$$\textbf{Doctor} \sqsubseteq some\_large\_concept\_description \sqcap \textbf{Person} \sqcap some\_other\_large\_concept\_description$$

Doctor $\sqsubseteq$ **Person** потому, что **Person** явно употребляется в определении

но это может быть сразу не видно.

Examples

(due to Michael Zakharyaschev)

Brief Summary of Guidelines for Ontology Building

- Always **paraphrase** a description or definition before encoding it in OWL
- Make all **primitives disjoint** — which requires that primitives form trees
- Use ***someValuesFrom*** as the default quantifier in restrictions
- Be careful to make **defined classes** defined (the default is primitive in Protégé).
  The classifier will place nothing under a primitive class
  (except in the presence of axioms/domain/range constrains)
- Remember the **open world assumption**.
  Insert closure restrictions if that is what you mean
- Be careful with **domain** and **range** constraints.
  Check them carefully if classification does not work as expected
- Be careful about the use of "and" and "or" (***intersectionOf*** and ***unionOf***)
- To spot **trivially satisfiable** restrictions early, always have an existential (***someValuesFrom***) restriction corresponding to every universal (***allValuesFrom***) restriction, either in the class or in one of its superclasses
- Run the classifier **frequently**; spot errors early

Travel service ontology

Sketch a normalised ontology for use by travel agency covering the following:

> Hotel, restaurant, sports, luxury hotel, bed and breakfast,
> safari, activity, hiking, spa treatment, sunbathing, sightseeing,
> accommodation rating (three stars, etc.), campground, surfing.

Build a class hierarchy and indicate which classes in it are primitive and which are definable. Indicate the required roles, their properties, domains and ranges, as well as individuals.

Define the following classes using OWL abstract syntax:

1. A two star hotel.
2. A spa resort (i.e., a destination offering a spa treatment).
3. A destination with sport activities but without safari.
4. A destination where all hotels have three star rating.
5. A destinations with at least three restaurants and at least four hotels.

1. Card sorting

hotel
restaurant
sports
luxury hotel
bed & breakfast
two stars

sunbathing
sightseeing
accommodation rating
three stars
campground
swimming pool

hiking
activity
spa treatment
safari
surfing
destination

1. Card sorting

hotel	sunbathing	hiking
restaurant	sightseeing	activity
sports	accommodation rating	spa treatment
luxury hotel	three stars	safari
bed & breakfast	campground	surfing
two stars	swimming pool	destination

2. Arrange Concepts/Properties into Hierarchy

self-standing	modifiers	relations	definable
– accommodation – hotel – luxury hotel – b&b – campground – activity – sports – surfing – hiking – safari – sightseeing – relaxation – sunbathing – spa treatment – restaurant – destination	accommodation rating – two stars – three stars – four stars – five stars hotel facility – swimming pool – meeting facilities	hasAccommodation hasActivity hasRating hasRestaurant hasFacility	two star hotel …

NB. All siblings in the hierarchy of self-standing entities are **disjoint**.

Class Hierarchy represents an "IS-A" Relation

a class A is a **subclass** of B if **every** instance of A is also an instance of B

Class Hierarchy represents an "IS-A" Relation

a class A is a **subclass** of B if **every** instance of A is also an instance of B

- hotels and B&B are accommodation

  therefore, *Hotel* and *B&B* may be regarded as subclasses of *Accommodation*

Class Hierarchy represents an "IS-A" Relation

a class A is a **subclass** of B if **every** instance of A is also an instance of B

- hotels and B&B are accommodation

  therefore, *Hotel* and *B&B* may be regarded as subclasses of *Accommodation*

- hiking and surfing are sport activities

  therefore, *Hiking* and *Surfing* may be regarded as subclasses of *Sports*

Class Hierarchy represents an "IS-A" Relation

a class A is a **subclass** of B if **every** instance of A is also an instance of B

- hotels and B&B are accommodation

  therefore, *Hotel* and *B&B* may be regarded as subclasses of *Accommodation*

- hiking and surfing are sport activities

  therefore, *Hiking* and *Surfing* may be regarded as subclasses of *Sports*

- **however**, neither a restaurant is a hotel nor a swimming pool is a hotel

  therefore, neither *Restaurant* nor *SwimmingPool* can be a subclass of *Hotel*

  (although a hotel may have a restaurant/swimming pool)

Do not confuse containment and a subclass relation!

Relations

```
ObjectProperty(hasAccommodation
                                        domain(Destination)   range(Accommodation))
ObjectProperty(hasActivity domain(Destination)   range(Activity))
ObjectProperty(hasRestaurant
                                domain(Destination)   domain(Hotel)   range(Restaurant))
ObjectProperty(hasFacility   domain(Hotel)   range(HotelFacility))
```

Relations

```
ObjectProperty(hasAccommodation
                                        domain(Destination)   range(Accommodation))
ObjectProperty(hasActivity domain(Destination)   range(Activity))
ObjectProperty(hasRestaurant
                                domain(Destination)   domain(Hotel)   range(Restaurant))
ObjectProperty(hasFacility   domain(Hotel)   range(HotelFacility))
```

Modifiers: using value sets

Consider modifier *AccommodationRating*
(one star, two stars, three stars, four stars, five stars)

Relations

```
ObjectProperty(hasAccommodation
                                        domain(Destination)   range(Accommodation))
ObjectProperty(hasActivity domain(Destination)   range(Activity))
ObjectProperty(hasRestaurant
                                  domain(Destination)   domain(Hotel)   range(Restaurant))
ObjectProperty(hasFacility   domain(Hotel)   range(HotelFacility))
```

Modifiers: using value sets

Consider modifier *AccommodationRating*
(one star, two stars, three stars, four stars, five stars)

```
ObjectProperty(hasRating   functional
                                  domain(Accommodation)   range(AccommodationRating))
Class(AccommodationRating   complete   oneOf(oneStar, twoStars, …, fiveStars))
DifferentIndividuals(oneStar, twoStars, …, fiveStars)
```

Specifying additional restrictions

'all luxury hotels have restaurants, swimming pools and meeting facilities'

```
Class(LuxuryHotel  partial
        restriction(hasRestaurant someValueFrom(owl:Thing))
        restriction(hasFacility someValueFrom(SwimmingPool))
        restriction(hasFacility someValueFrom(MeetingFacility)))
```

Q1: Definitions vs. Descriptions

A two star hotel (**definition**):

'a two star hotel is any hotel that has two star rating'

```
Class(TwoStarHotel  complete  Hotel
    restriction(hasRating value(twoStars)))
```

Q1: Definitions vs. Descriptions

A two star hotel (**definition**):

'a two star hotel is any hotel that has two star rating'

```
Class(TwoStarHotel complete Hotel
    restriction(hasRating value(twoStars)))
```

Consider now the following **description**:

'all w-two-star-hotels have two star rating'

```
Class(W-TwoStarHotel partial Hotel
    restriction(hasRating value(twoStars)))
```

Q1: Definitions vs. Descriptions

A two star hotel (**definition**):

'a two star hotel is any hotel that has two star rating'

```
Class(TwoStarHotel complete Hotel
    restriction(hasRating value(twoStars)))
```

Consider now the following **description**:

'all w-two-star-hotels have two star rating'

```
Class(W-TwoStarHotel partial Hotel
    restriction(hasRating value(twoStars)))
```

Then

- *chelseaInn* is an **instance** of *TwoStarHotel* **(necessary & sufficient cond.)**
- but *chelsea_inn* **not** an **instance** of *W-TwoStarHotel* **(necessary cond.)**

w.r.t. the following ABox

chelsea_inn: *Hotel*, (*chelsea_inn*, *twoStars*): *hasRating*

Q2: Use *someValuesFrom* as the default quantifier in restrictions

Destinations with spa treatment:

'a spa resort is any destination that has a spa treatment'

```
Class(SpaResort complete
        Destination
        restriction(hasActivity someValuesFrom(SpaTreatment)))
```

Q2: Use *someValuesFrom* as the default quantifier in restrictions

Destinations with spa treatment:

'a spa resort is any destination that has a spa treatment'

```
Class(SpaResort complete
        Destination
        restriction(hasActivity someValuesFrom(SpaTreatment)))
```

Note that **any destination without activities** is an instance of the concept

```
Class(W-SpaResort complete
        Destination
        restriction(hasActivity allValuesFrom(SpaTreatment)))
```

'only' does not imply 'some'!

Q3: Use *someValuesFrom* as the default quantifier (cont.)

A destination with sport activities but without safari:

'a destination with sport activities but without safari is
any destination that has some sport activity but does not have any safari'

```
Class(DestinationWithSportButNoSafari complete
        Destination
        restriction(hasActivity someValuesFrom(Sports))
        complementOf(restriction(hasActivity someValuesFrom(Safari))))
```

Q3: Use *someValuesFrom* as the default quantifier (cont.)

A destination with sport activities but without safari:

'a destination with sport activities but without safari is
any destination that has some sport activity but does not have any safari'

```
Class(DestinationWithSportButNoSafari complete
        Destination
        restriction(hasActivity someValuesFrom(Sports))
        complementOf(restriction(hasActivity someValuesFrom(Safari))))
```

or, **equivalently**,

$\neg\exists R.C$ **means** $\forall R.\neg C$

'... is any destination that has some sport activity and
has only activities that are not safari activities'

```
Class(DestinationWithSportButNoSafari complete
        Destination
        restriction(hasActivity someValuesFrom(Sports))
        restriction(hasActivity allValuesFrom(complementOf(Safari))))
```

NB. Safari and Sports are **disjoint**

Q4: Unions, Intersections and Complements

A destination where all hotels have three star rating.

'a destination where all hotels have three star rating is
any destination that has only either not hotels or three star hotels'

```
Class(ThreeStarHotelDestination complete
        Destination
        restriction(hasAccommodation allValuesFrom(unionOf(
            complementOf(Hotel)
            intersectionOf(Hotel restriction(hasRating value(threeStars)))))))
```

Q4: Unions, Intersections and Complements

A destination where all hotels have three star rating.

'a destination where all hotels have three star rating is
any destination that has only either not hotels or three star hotels'

```
Class(ThreeStarHotelDestination complete
        Destination
        restriction(hasAccommodation allValuesFrom(unionOf(
            complementOf(Hotel)
            intersectionOf(Hotel restriction(hasRating value(threeStars)))))))
```

Alternatively,

'a destination where all hotels have three star rating is
any destination that has no hotel that has not a three star rating'

```
Class(ThreeStarHotelDestination complete
        Destination
        complementOf(restriction(hasAccommodation someValuesFrom(
            intersectionOf(Hotel
                complementOf(restriction(hasRating value(threeStars))))))))
```

Q5: Number restrictions

A destinations with at least three restaurants and at least four hotels.

'a destination with at least three restaurants and at least four hotels is any destination that has at least three restaurants and has at least four **places to stay**'

```
Class(TRFHDestination complete
        Destination
        restriction(hasRestaurant minCardinality(3))
        restriction(hasAccommodation minCardinality(4)))
```

NB. OWL-DL 1.0 has no **qualified** number restrictions (OWL 1.1 has them), so in this example we have to slightly extend the definition and include any types of accommodation (not only hotels).