法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院和讲者共同拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：小象
  - 新浪微博：ChinaHadoop

第七讲：知识推理

大纲

- 本体知识推理简介与任务分类
- 本体推理方法与工具介绍
- 实践展示：使用Jena进行知识推理

大纲

- 本体知识推理简介与任务分类
- 本体推理方法与工具介绍
- 实践展示：使用Jena进行知识推理

OWL本体语言

W3C　描述逻辑　一阶谓词逻辑的子集

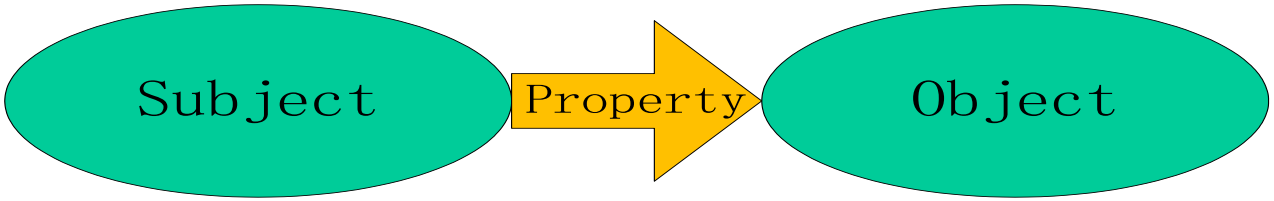
OWL本体语言

- 是知识图谱语言中**最规范**，**最严谨**，**表达能力最强**的语言
- 基于RDF语法，使表示出来的文档具有语义理解的结构基础
- 促进了统一词汇表的使用，定义了丰富的语义词汇
- 允许逻辑推理

OWL本体语言

语法

- RDF语法，三元组



逻辑基础：描述逻辑

- 描述逻辑 (Description Logic) 是基于对象的知识表示的形式化，也叫概念表示语言或术语逻辑。是一阶谓词逻辑的一个可判定子集

OWL本体语言

描述逻辑系统

- 一个描述逻辑系统包括四个基本的组成部分
  1) 最基本的元素：概念、关系和个体
  2) TBox术语集(概念术语的公理集合)
  3) ABox断言集(个体的断言集合)
  4) TBox和ABox上的推理机制

- 不同的描述逻辑系统的表示能力与推理机制由于对这四个组成部分的不同选择而不同。

OWL本体语言

- 概念
  - 解释为一个领域的子集
  - 例如，学生：$\{x|\text{student}(x)\}$
- 关系
  - 解释为该领域上的二元关系（笛卡尔乘积）
  - 例如，朋友：$\{\langle x,y\rangle|\text{friend}(x,y)\}$
- 个体
  - 一个领域内的实例
  - 例如，小明：$\{\text{Ming}\}$

OWL本体语言

- TBox——泛化的知识
  - 描述概念和关系的知识，被称之为**公理 (Axiom)**
  - 由于概念之间存在包含关系，TBox知识形成类似**格 (Lattice)**的结构，这种数学结构是由包含关系决定的，与具体实现无关

- ABox——具体个体的信息
  - ABox包含外延知识 (又称**断言 (Assertion)**)，描述论域中的特定个体

描述逻辑的知识库$K := \langle T, A \rangle$， T即TBox， A即ABox

OWL本体语言

- Tbox语言
  - 定义：引入概念以及关系的名称
    - 例如，Mother，Person，has_child
  - 包含：声明包含关系的公理
    - 例如，$\text{Mother} \sqsubseteq \exists \text{has\_child}.\text{Person}$

概念和概念，关系和概念之间可以组成更复杂的概念。

- Abox语言
  - 概念断言——表示一个对象是否属于某个概念
    - 例如，Mother(Alice)，Person(Bob)
  - 关系断言——表示两个对象是否满足特定关系
    - 例如，has_child(Alice, Bob)

OWL本体语言

描述逻辑语义

- 解释I是知识库K的模型，当且仅当I是K中每个断言的模型。若一个知识库K有一个模型，则称K是可满足的。若断言σ对于K的每个模型都是满足的，则称K逻辑蕴含σ，记为$K \models \sigma$。对概念C，若K有一个模型I使得$C^I \neq \emptyset$，则称C是可满足的。

描述逻辑依据提供的构造算子，在简单的概念和关系上构造出复杂的概念和关系。描述逻辑至少包含以下构造算子：交 ($\sqcap$)，并 ($\sqcup$)，非 ($\neg$)，存在量词 ($\exists$)和全称量词 ($\forall$)

OWL本体语言

描述逻辑的语义

构造算子	语法	语义	例子
原子概念	A	$A^I \subseteq \Delta^I$	Human
原子关系	R	$R^I \subseteq \Delta^I \times \Delta^I$	has_child
对概念C，D和关系(role) R			
合取	$C \sqcap D$	$C^I \cap D^I$	Human ⊓ Male
析取	$C \sqcup D$	$C^I \cup D^I$	Doctor ⊔ Lawyer
非	$\neg C$	$\Delta^I \backslash C$	¬Male
存在量词	$\exists R.C$	$\{x \mid \exists y.\langle x,y\rangle \in R^I \wedge y \in C^I\}$	∃has_child.Male
全称量词	$\forall R.C$	$\{x \mid \forall y.\langle x,y\rangle \in R^I \Rightarrow y \in C^I\}$	∀has_child.Doctor

有了语义之后，我们可以进行推理。通过语义来保证推理的正确和完备性。

OWL本体语言

描述逻辑与OWL词汇的对应

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
sameClassAs	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
samePropertyAs	$P_1 \equiv P_2$	cost $\equiv$ price
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{Prsident_Bush} $\equiv$ {G_W_Bush}
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$Female
differentIndividualFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{John} $\sqsubseteq \neg${Peter}
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+ \sqsubseteq$ ancestor
uniqueProperty	$T \sqsubseteq \leq 1P$	$T \sqsubseteq \leq$ 1hasMother
unambiguousProperty	$T \sqsubseteq \leq 1P^-$	$T \sqsubseteq \leq$ 1isMotherOf$^-$

知识推理任务

- 可满足性 (satisfiability)
- 分类 (classification)
- 实例化 (materialization)

推理就是通过各种方法获取新的知识或者结论，这些知识和结论满足语义。

OWL本体推理

- 可满足性 (satisfiability)
  - 本体可满足性
    - 检查一个本体是否可满足，即检查该本体是否有模型。如果本体不可满足，说明存在不一致
  - 概念可满足性
    - 检查某一概念的可满足性，即检查是否具有模型，使得针对该概念的解释不是空集

一个不可满足的本体

$\text{Man} \sqcap \text{Women} \sqsubseteq \bot$

Man(Allen)

Women(Allen)

一个不可满足的概念

$\text{Eternity} \sqsubseteq \bot$

针对可满足性推理采用Tableaux算法将在第二部分进一步介绍

OWL本体推理

- 分类 (classification)
  - 针对Tbox的推理，计算新的概念包含关系

$$\text{Mother} \sqsubseteq \text{Women},\ \text{Women} \sqsubseteq \text{Person} \Rightarrow \text{Mother} \sqsubseteq \text{Person}$$

注意：这里的分类和机器学习中分类的区别！

OWL本体推理

分类的例子

$Apple \sqsubseteq \exists beInvestedBy.(Fidelity \sqcap BlackStone)$

苹果由富达和黑石投资。

$\exists beFundedBy.Fidelity \sqsubseteq InnovativeCompanies$

借助富达融资的公司都是创新企业。

$\exists beFundedBy.BlackStone \sqsubseteq InnovativeCompanies$

借助黑石融资的公司都是创新企业。

$beInvestedBy \sqsubseteq beFundedBy$

投资即是帮助融资。

富达基金和黑石基金都喜欢投资高新科技公司。

OWL本体推理

分类的例子

$Apple \sqsubseteq \exists beInvestedBy.(Fidelity \sqcap BlackStone)$

$\exists beFundedBy.Fidelity \sqsubseteq InnovativeCompanies$

$\exists beFundedBy.BlackStone \sqsubseteq InnovativeCompanies$

$beInvestedBy \sqsubseteq beFundedBy$

OWL本体推理

分类的例子

Apple ⊑ ∃beInvestedBy.(Fidelity ⊓BlackStone)

∃beFundedBy.Fidelity ⊑ InnovativeCompanies

∃beFundedBy.BlackStone ⊑ InnovativeCompanies

beInvestedBy ⊑ beFundedBy

Apple ⊑ ∃beInvestedBy.Fidelity

OWL本体推理

分类的例子

Apple ⊑ ∃beInvestedBy.(Fidelity ⊓BlackStone)

∃beFundedBy.Fidelity ⊑ InnovativeCompanies

∃beFundedBy.BlackStone ⊑ InnovativeCompanies

beInvestedBy ⊑ beFundedBy

Apple ⊑ ∃beInvestedBy.Fidelity

Apple ⊑ ∃beFundedBy.Fidelity

OWL本体推理

分类的例子

$\text{Apple} \sqsubseteq \exists \text{beInvestedBy}.(\text{Fidelity} \sqcap \text{BlackStone})$

$\exists \text{beFundedBy}.\text{Fidelity} \sqsubseteq \text{InnovativeCompanies}$

$\exists \text{beFundedBy}.\text{BlackStone} \sqsubseteq \text{InnovativeCompanies}$

$\text{beInvestedBy} \sqsubseteq \text{beFundedBy}$

$\text{Apple} \sqsubseteq \exists \text{beInvestedBy}.\text{Fidelity}$

$\text{Apple} \sqsubseteq \exists \text{beFundedBy}.\text{Fidelity}$

$\text{Apple} \sqsubseteq \text{InnovativeCompanies}$

OWL本体推理

- 实例化 (materialization)
  - 实例化即计算属于某个概念或关系的所有实例的集合

计算新的类实例信息

Mother(Alice)
Mother ⊑ Women

Women(Alice)

计算新的二元关系

has_son(Alice, Bob)
has_son ⊑ has_child

has_child(Alice, Bob)

OWL本体推理

实例化(materialization)的一个例子：

一个兼并重组(可以是业务兼并，不是收购)套利策略：
策略思想：与大盘股公司兼并重组的上市企业有很高的预期收益。

$$\exists merge.BigCapital \sqsubseteq ValueSecurity$$

定义什么是大盘股(按照策略自己调整)：
上证50和沪深300指数中的标的属于大盘股。

$$SZ50 \sqsubseteq BigCapital,\ HS300 \sqsubseteq BigCapital,\ SZ180 \sqsubseteq HS300$$

选股目标：找出兼并重组策略下所有高预期公司：

使用OWL的实例化推理

OWL本体推理

实例化(materialization)的一个例子：

一个兼并重组套利策略：

$$\exists merge.BigCapital \sqsubseteq ValueSecurity$$

$$SZ50 \sqsubseteq BigCapital,\ HS300 \sqsubseteq BigCapital,\ SZ180 \sqsubseteq HS300$$

选股目标：找出兼并重组策略下所有高预期公司：

merge(SZ300377, SH600570)　赢时胜和恒生电子在区块链方面有业务兼并

SZ180(SH600570)　恒生电子是上证180的成分股

推理：

OWL本体推理

实例化(materialization)的一个例子：

一个兼并重组套利策略：

$\exists merge.BigCapital \sqsubseteq ValueSecurity$

$SZ50 \sqsubseteq BigCapital, HS300 \sqsubseteq BigCapital, \mathbf{SZ180 \sqsubseteq HS300}$

选股目标：找出兼并重组策略下所有高预期公司：

merge(SZ300377, SH600570) 赢时胜和恒生电子在区块链方面有业务兼并

SZ180(SH600570) 恒生电子是上证180的成分股

推理：

HS300(SH600570)

OWL本体推理

实例化(materialization)的一个例子：

一个兼并重组套利策略：

$\exists merge.BigCapital \sqsubseteq ValueSecurity$

$SZ50 \sqsubseteq BigCapital$, **$HS300 \sqsubseteq BigCapital$**, $SZ180 \sqsubseteq HS300$

选股目标：找出兼并重组策略下所有高预期公司：

merge(SZ300377, SH600570) 赢时胜和恒生电子在区块链方面有业务兼并

SZ180(SH600570) 恒生电子是上证180的成分股

推理：

HS300(SH600570), BigCapital(SH600570)

OWL本体推理

实例化(materialization)的一个例子：

一个兼并重组套利策略：

$\exists merge.BigCapital \sqsubseteq ValueSecurity$

$SZ50 \sqsubseteq BigCapital, HS300 \sqsubseteq BigCapital, SZ180 \sqsubseteq HS300$

选股目标：找出兼并重组策略下所有高预期公司：

merge(SZ300377, SH600570) 赢时胜和恒生电子在区块链方面有业务兼并

SZ180(SH600570) 恒生电子是上证180的成分股

推理：

HS300(SH600570), **BigCapital(SH600570),** ValueSecurity(SZ300377)

OWL本体推理

实例化(materialization)的一个例子：

一个兼并重组套利策略：

$\exists merge.BigCapital \sqsubseteq ValueSecurity$

$SZ50 \sqsubseteq BigCapital, HS300 \sqsubseteq BigCapital, SZ180 \sqsubseteq HS300$

选股目标：找出兼并重组策略下所有高预期公司：

merge(SZ300377, SH600570) 赢时胜和恒生电子在区块链方面有业务兼并

SZ180(SH600570) 恒生电子是上证180的成分股

推理：

HS300(SH600570), **BigCapital(SH600570),** ValueSecurity(SZ300377)

结论：SZ300377赢时胜在短期内是一家高收益公司。

这本质上用基于消息面的套利，推理机可以完成复杂股票筛选的过程。

大纲

- 知识推理简介与任务分类
- 本体推理方法与工具介绍
- 实践展示：使用Jena进行知识推理

基于Tableaux运算的方法

- Tableaux运算
  - 适用场合
    - 检查某一本体的可满足性，以及实例检测
  - 基本思想
    - 通过一系列规则构建Abox，以检测可满足性，或者检测某一实例是否存在于某概念
    - 基本思想类似于一阶逻辑的归结反驳

基于Tableaux运算的方法

Tableaux运算规则（以主要DL算子举例）

初始情况下，∅是原始的Abox，迭代运用如下规则：

$\sqcap^{+}$-规则：若$C \sqcap D(x) \in \emptyset$，且$C(x), D(x) \notin \emptyset$，则$\emptyset := \emptyset \cup \{C(x), D(x)\}$;

$\sqcap^{-}$-规则：若$C(x), D(x) \in \emptyset$，且$C \sqcap D(x) \notin \emptyset$，则$\emptyset := \emptyset \cup \{C \sqcap D(x)\}$;

$\exists$-规则：若$\exists R.C(x) \in \emptyset$，且$R(x,y), C(y) \notin \emptyset$，则$\emptyset := \emptyset \cup \{R(x,y), C(y)\}$，

其中，y是新加进来的个体;

$\forall$-规则：若$\forall R.C(x), R(x,y) \in \emptyset$，且$C(y) \notin \emptyset$，则$\emptyset := \emptyset \cup \{C(y)\}$;

$\sqsubseteq$-规则：若$C(x) \in \emptyset, C \sqsubseteq D$，且$D(x) \notin \emptyset$，则$\emptyset := \emptyset \cup \{D(x)\}$;

$\bot$-规则：若$\bot(x) \in \emptyset$，则拒绝∅;

基于Tableaux运算的方法

Tableaux运算规则举例

给定如下本体，检测实例Allen是否在Woman中？

$\text{Man} \sqcap \text{Women} \sqsubseteq \bot$
$\text{Man}(\text{Allen})$

基于Tableaux运算的方法

Tableaux运算规则举例

给定如下本体，检测实例Allen是否在Woman中？

$Man \sqcap Women \sqsubseteq \bot$
Man(Allen)
Woman(Allen)

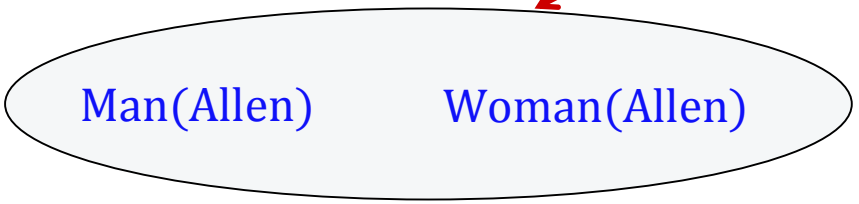
加入待反驳的结论

基于Tableaux运算的方法

Tableaux运算规则举例

初始Abox，记为∅

给定如下本体，检测实例Allen是否在Woman中？



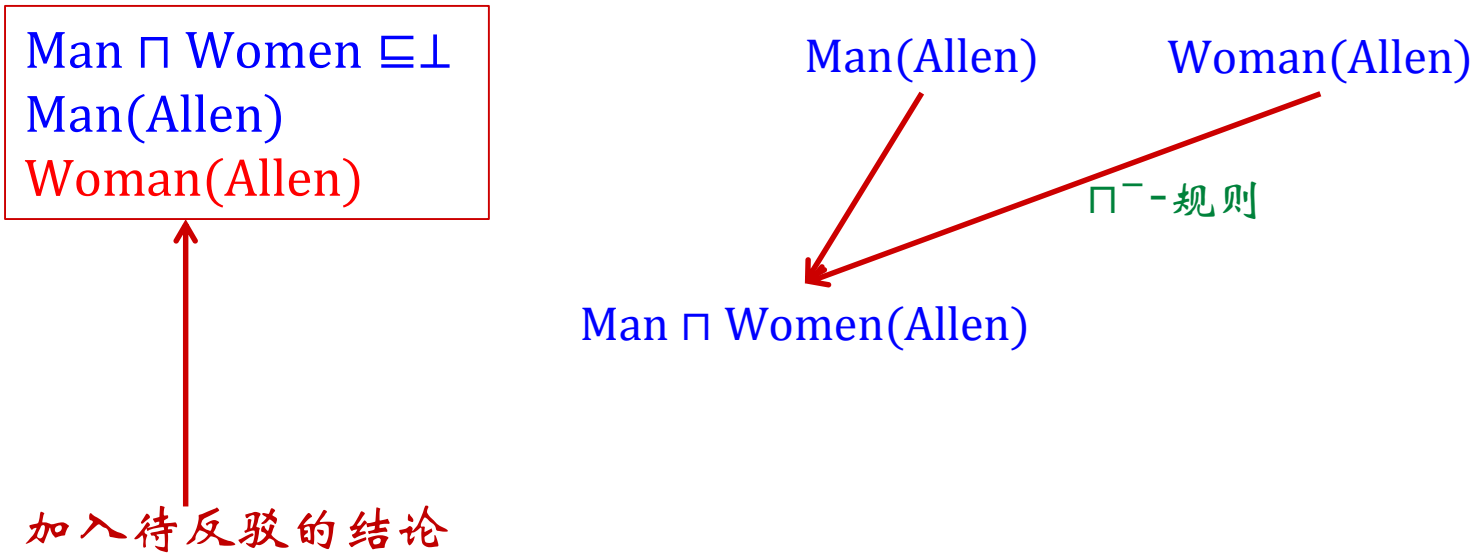
$Man \sqcap Women \sqsubseteq \bot$
Man(Allen)
Woman(Allen)

加入待反驳的结论

基于Tableaux运算的方法

Tableaux运算规则举例

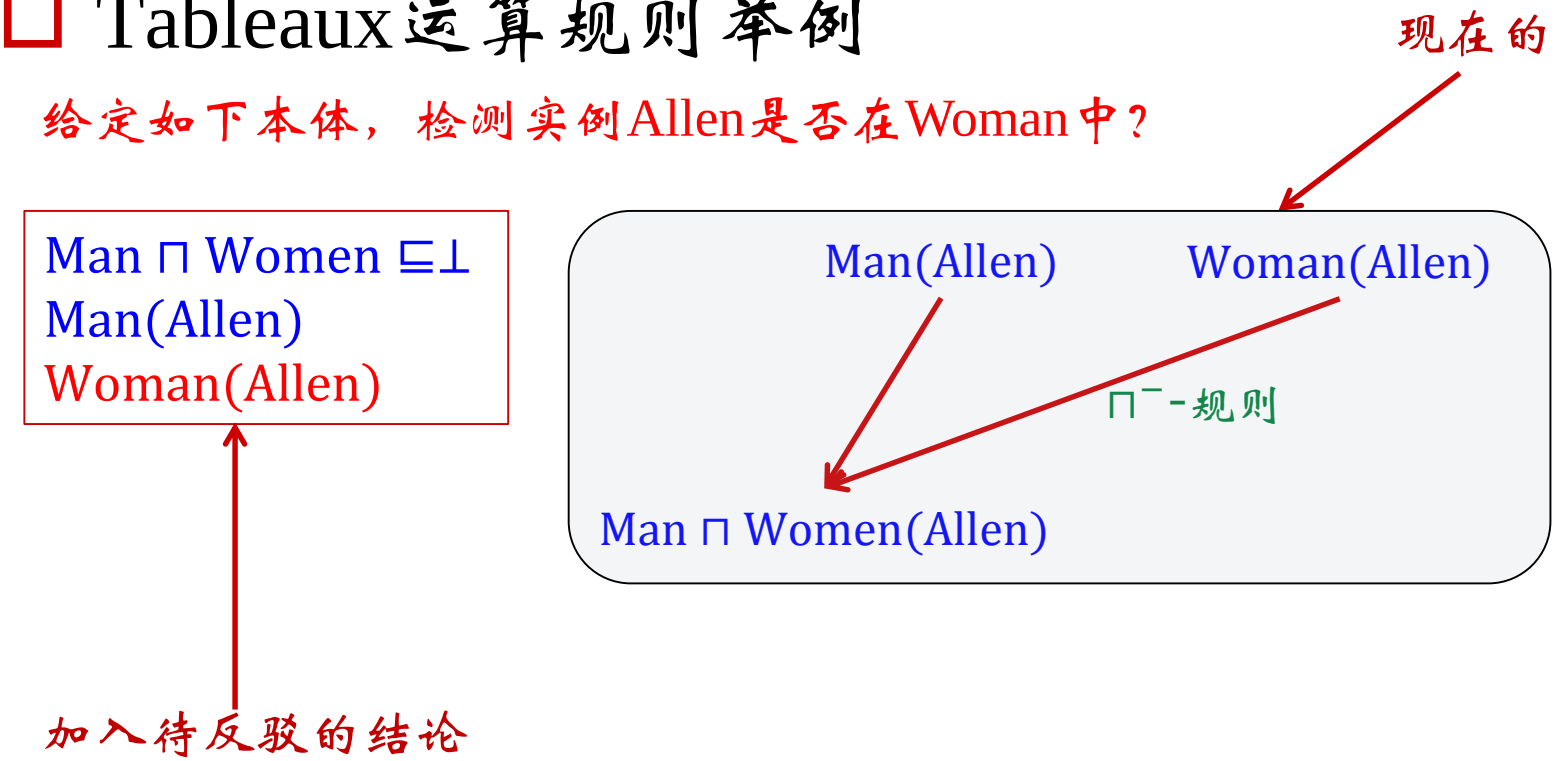
给定如下本体，检测实例Allen是否在Woman中？



基于Tableaux运算的方法

Tableaux运算规则举例

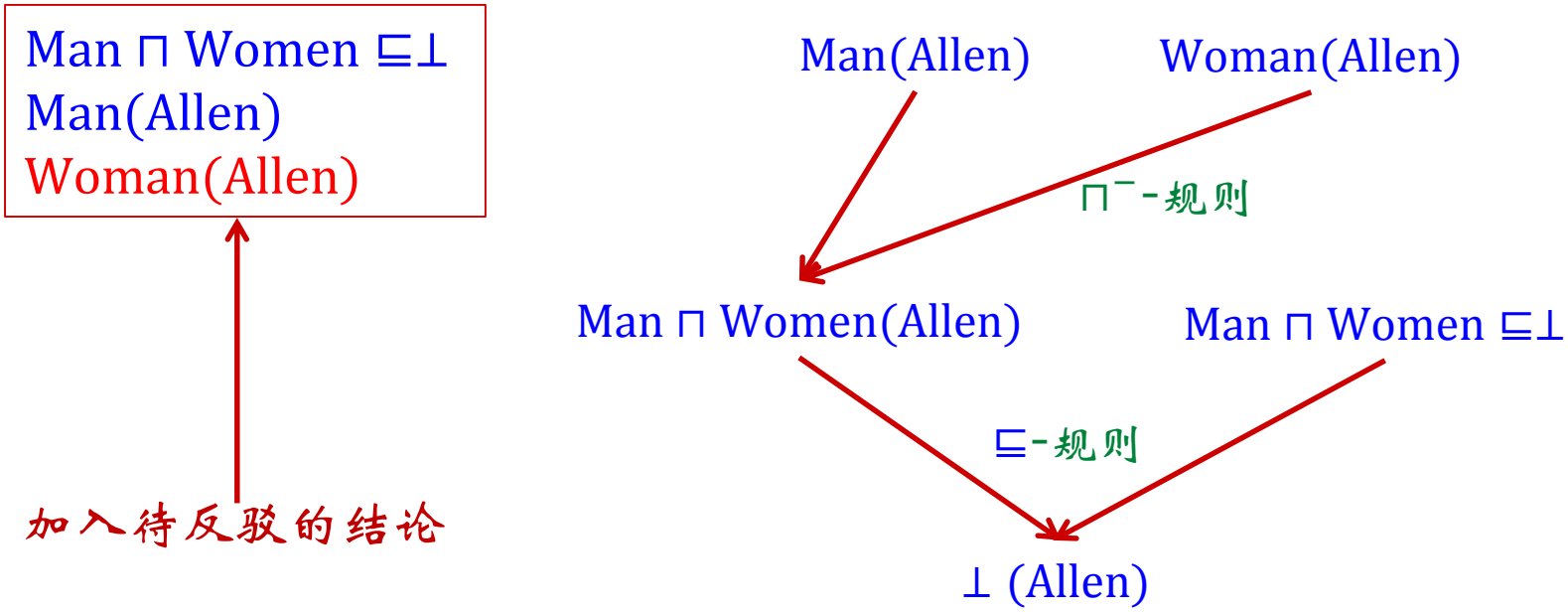
给定如下本体，检测实例Allen是否在Woman中？



基于Tableaux运算的方法

Tableaux运算规则举例

给定如下本体，检测实例Allen是否在Woman中？



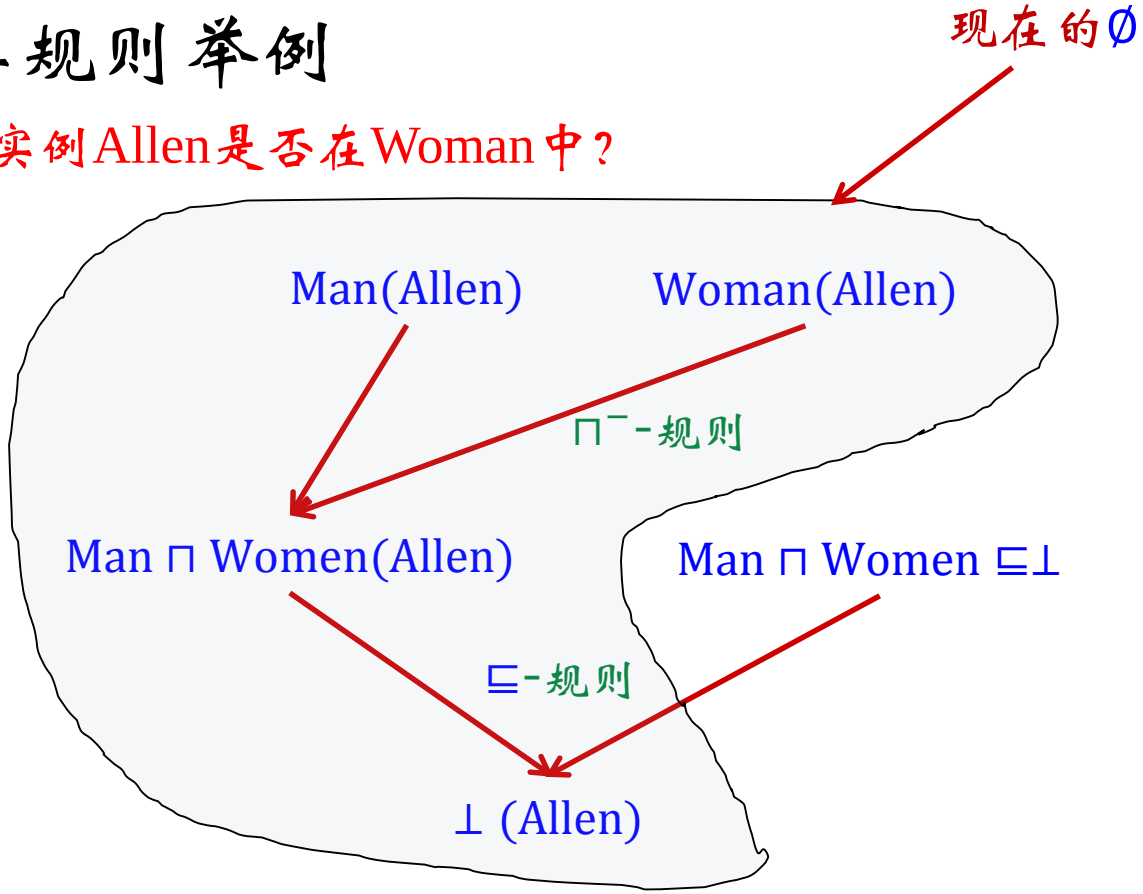
基于Tableaux运算的方法

Tableaux运算规则举例

给定如下本体，检测实例Allen是否在Woman中？

Man ⊓ Women ⊑ ⊥
Man(Allen)
Woman(Allen)

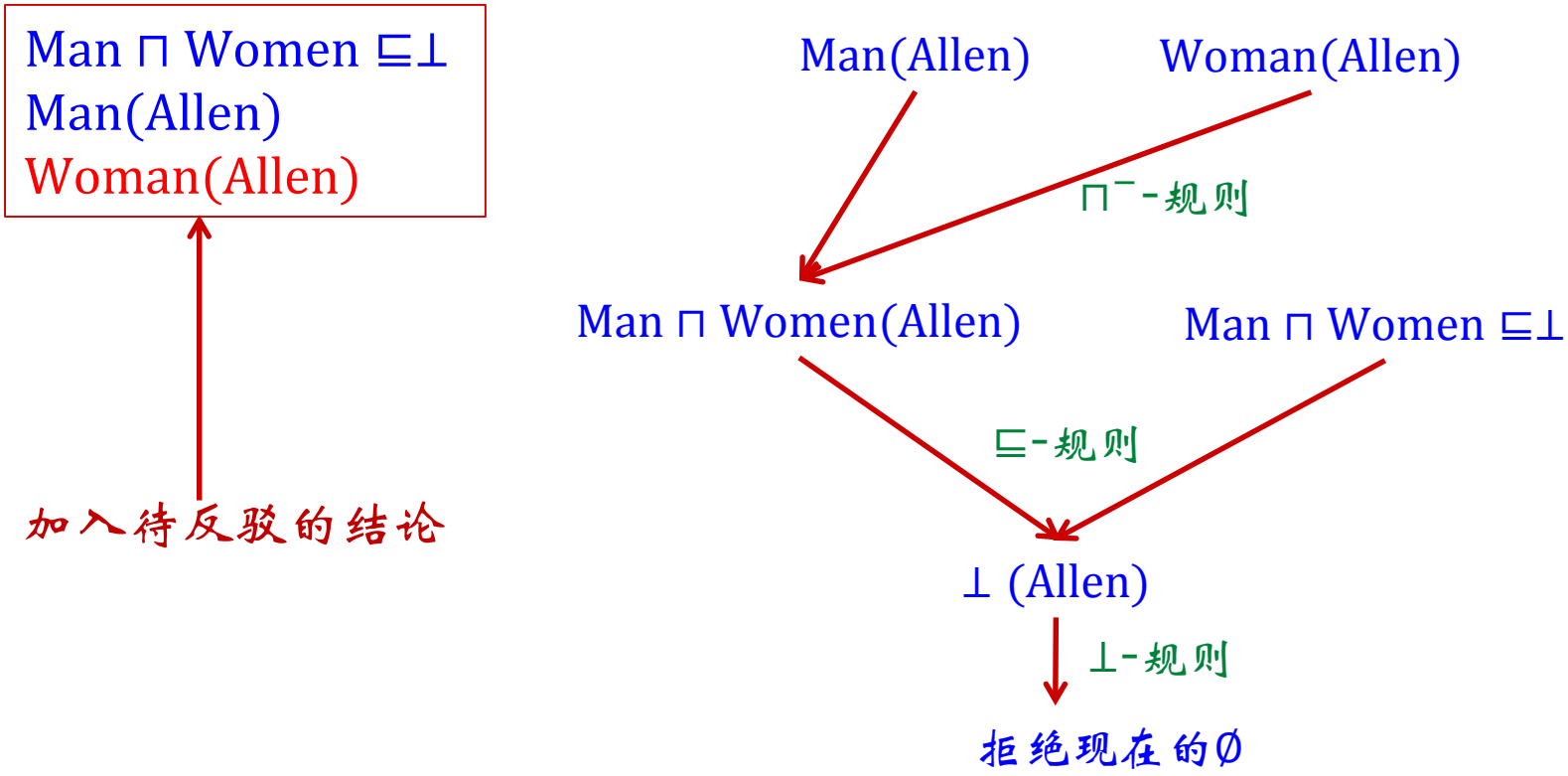
加入待反驳的结论



基于Tableaux运算的方法

Tableaux运算规则举例

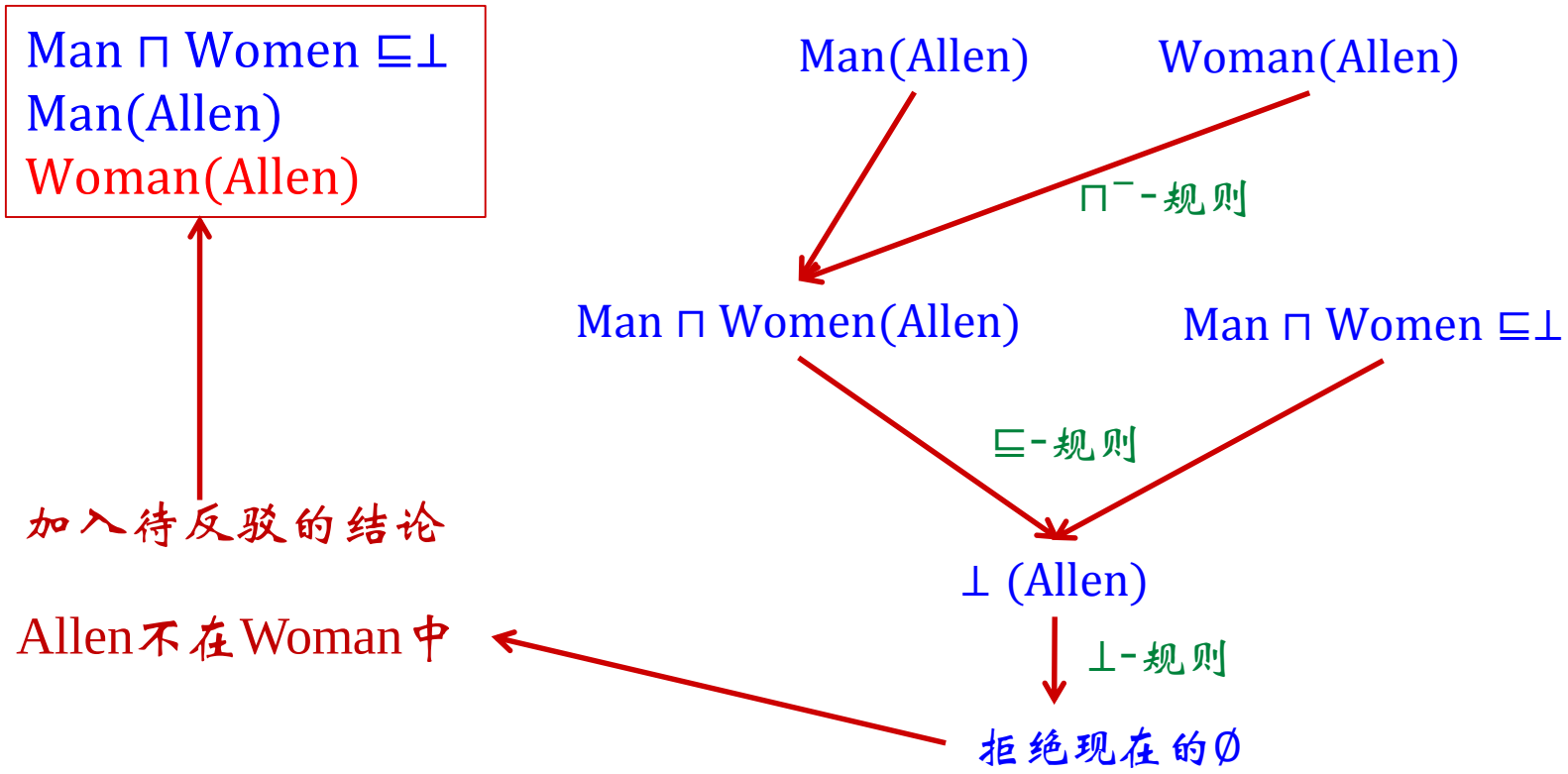
给定如下本体，检测实例Allen是否在Woman中？



基于Tableaux运算的方法

Tableaux运算规则举例

给定如下本体，检测实例Allen是否在Woman中？



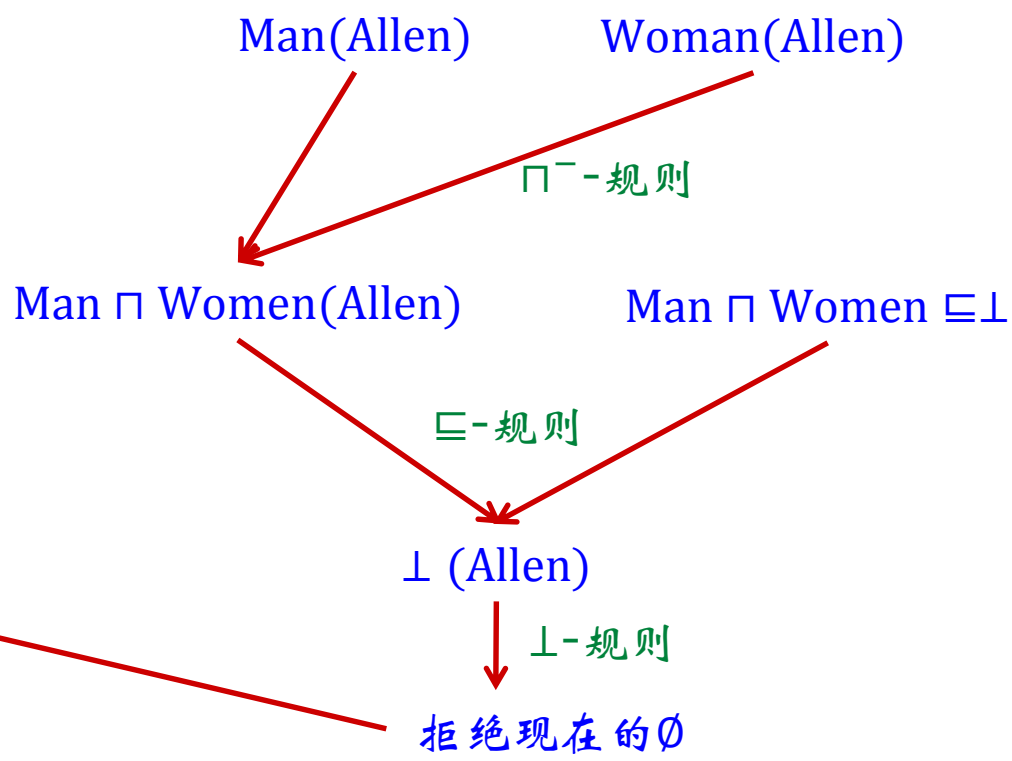
基于Tableaux运算的方法

Tableaux运算规则举例

给定如下本体，检测是否可满足？

Man ⊓ Women ⊑ ⊥
Man(Allen)
Woman(Allen)

如果Woman(Allen)在初始情况已存在于原始本体，那么推导出该本体不可满足！



基于Tableaux运算的方法

- Tableaux运算的正确性
  - 基于Herbrand模型
    - 利用Tableaux算法构建的Abox本质上是该本体的Herbrand模型
    - 一个本体的Herbrand模型与该本体任意模型的一个子集是同构的
    - 拒绝Herbrand模型，就是拒绝了本体的所有模型，因此该本体一定是不可满足的
    - 无法拒绝Herbrand模型，那么该模型本身就是本体的一个模型；只要本体有模型，那么一定是可满足的

相关工具简介

- FaCT++
  - 曼彻斯特大学开发的描述逻辑推理机
  - 使用C++实现，且能与Protégé集成
  - Java版本名为Jfact，基于OWL API
- 使用举例

构建推理机

```
OWLReasonerFactory reasonerFactory = new JFactFactory();
OWLReasoner reasoner = this.reasonerFactory.createReasoner(ontology);
```

进行推理（分类）

```
reasoner.precomputeInferences(InferenceType.CLASS_HIERARCHY);
```

http://owl.man.ac.uk/factplusplus/

相关工具简介

Racer

- 美国Franz Inc.公司开发的以描述逻辑为基础的本体推理，也可以用作语义知识库
- 支持OWL DL，支持部分OWL 2 DL
- 支持单机和客户端/服务器两种模式
- 用Allegro Common Lisp实现

使用举例

```
(classify-tbox &optional (tbox (current-tbox)))
```

进行TBox推理

```
(realize-abox &optional (abox (current-abox)))
```

进行ABox推理

https://www.ifis.uni-luebeck.de/index.php?id=385

相关工具简介

Pellet

https://github.com/stardog-union/pellet

- 马里兰大学开发的本体推理机
- 支持OWL DL的所有特性，包括枚举类和XML数据类型的推理
- 支持OWL API以及Jena的接口

使用举例

构建一个推理机

```
PelletReasoner reasoner =
PelletReasonerFactory.getInstance().createReasoner(
ontology);
```

通过查询接口进行推理

```
NodeSet<OWLNamedIndividual> individuals =
reasoner.getInstances(Person, true);
```

相关工具简介

- HermiT
  - 牛津大学开发的本体推理机
  - 基于hypertableau运算，更加高效
  - 支持OWL 2规则
- 使用举例

```
Reasoner hermit = new Reasoner(ontology);
```

构建一个推理机

```
System.out.println(hermit.isConsistent());
```

一致性检测

http://www.hermit-reasoner.com/

相关工具简介

相关工具总结

工具名称	支持本体语言	编程语言	算法
FaCT++	OWL DL	C++	tableau-based
Racer	OWL DL	Common Lisp	tableau-based
Pellet	OWL DL	Java	tableau-based
HermiT	OWL 2 Profiles	Java	hypertableau

基于逻辑编程改写的方法

规则推理

- **本体推理的局限**:

  (1) 仅支持预定义的本体公理上的推理

  (无法针对自定义的词汇支持灵活推理)

  (2) 用户无法定义自己的推理过程

- 引入规则推理

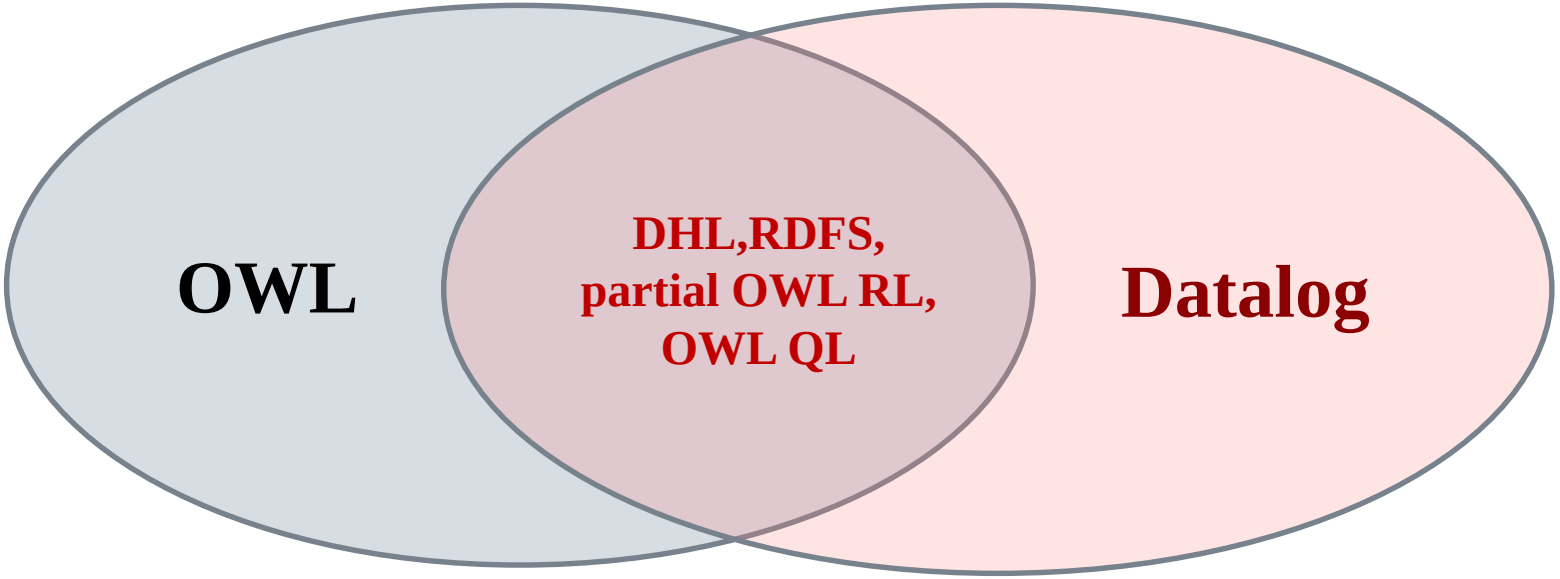
  (1) 可以根据特定的场景定制规则，以实现用户自定义的推理过程

  (2) Datalog语言可以结合本体推理和规则推理

基于逻辑编程改写的方法

Datalog语言

- 面向知识库和数据库设计的逻辑语言，表达能力与OWL相当，支持递归
- 便于撰写规则，实现推理



基于逻辑编程改写的方法

- Datalog语法
  - 原子 (Atom)
    - $p(t_1, t_2, \ldots, t_n)$
    - 其中p是谓词，n是目数，t_i是项(变量或常量)
    - 例如：has_child(X, Y)
  - 规则 (Rule)
    - $H: -B_1, B_2, \ldots, B_m.$
    - 由原子构建，其中H是头部原子，$B_1, B_2, \ldots, B_m$是体部原子
    - 例如：has_child(X, Y): −has_son(X, Y).

基于逻辑编程改写的方法

- Datalog语法
  - 事实 (Fact)
    - $F(c_1, c_2, \ldots, c_n):-$
    - 没有体部且没有变量的规则
    - 例如：has_child(Alice, Bob):−
  - Datalog程序是规则的集合
    - 例如：

Datalog程序P:
has_child(X, Y):−has_son(X, Y).
has_child(Alice, Bob):−

基于逻辑编程改写的方法

Datalog推理举例

规则集

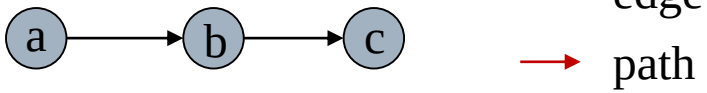
$path(x, y) :- edge(x, y).$

$path(x, y) :- path(x, z), path(z, y).$

事实集

$edge(a, b).$

$edge(b, c).$



基于逻辑编程改写的方法

Datalog推理举例

规则集

$path(x, y) :- edge(x, y).$

$path(x, y) :- path(x, z), path(z, y).$

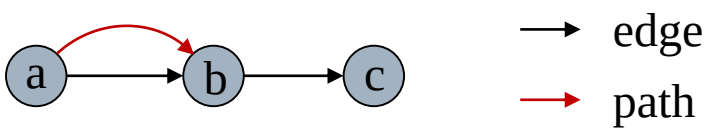
事实集

$edge(a, b).$

$edge(b, c).$

结果集

$path(a, b).$



基于逻辑编程改写的方法

Datalog推理举例

规则集

$path(x, y) :- edge(x, y).$

$path(x, y) :- path(x, z), path(z, y).$

事实集

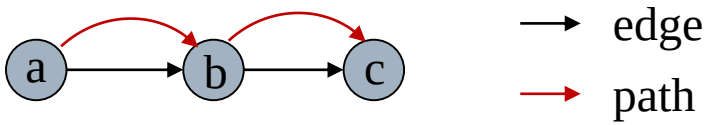
$edge(a, b).$

$edge(b, c).$

结果集

$path(a, b).$

$path(b, c).$



基于逻辑编程改写的方法

Datalog推理举例

规则集

$path(x, y) :- edge(x, y).$

$path(x, y) :- path(x, z), path(z, y).$

事实集

$edge(a, b).$

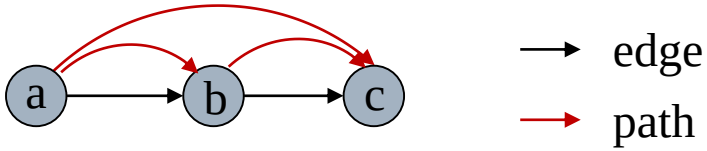
$edge(b, c).$

结果集

$path(a, b).$

$path(b, c).$

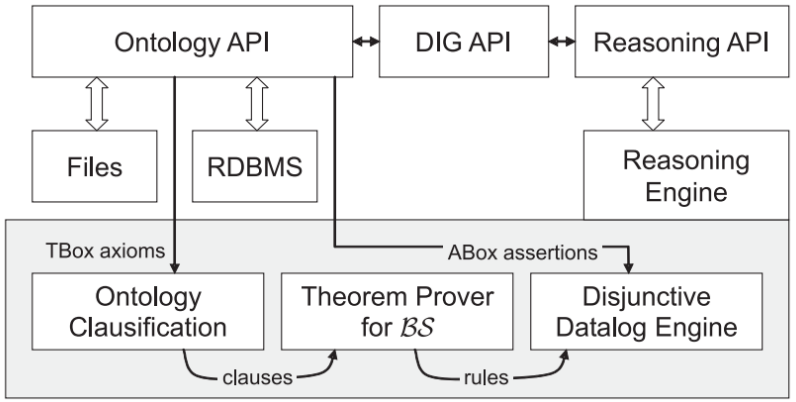
$path(a, c).$



相关工具介绍

KAON2

- OWL推理机、本体管理API
- 基于一阶消解原理
- 针对大规模ABox进行优化



使用举例

http://kaon2.semanticweb.org/

创建本体

```
OntologyManager ontologyManager=KAON2Manager.newOntologyManager();
Ontology ontology=ontologyManager.createOntology("http://kaon2.semanticweb.org/example04",new
HashMap<String,Object>());
```

创建推理机并用于查询

```
Reasoner reasoner=ontology.createReasoner();
Query whatDoPeopleKnowAbout=reasoner.createQuery(new Literal[] {
        KAON2Manager.factory().literal(true,person,new Term[] { X }),
        KAON2Manager.factory().literal(true,personKnowsAboutTopic,new Term[] { X,Y }),
    },new Variable[] { X,Y});
```

相关工具介绍

RDFox

https://www.cs.ox.ac.uk/isg/tools/RDFox/

- 由牛津大学开发的可扩展、跨平台、基于内存的RDF三元组存储系统
- 支持并行Datalog推理、SPARQL查询

使用举例

创建本体与存储

```
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
OWLOntology ontology =
manager.loadOntologyFromOntologyDocument(IRI.create(“test.owl"));
DataStore store = new DataStore(DataStore.StoreType.ParallelSimpleNN, true);
```

导入本体进行推理

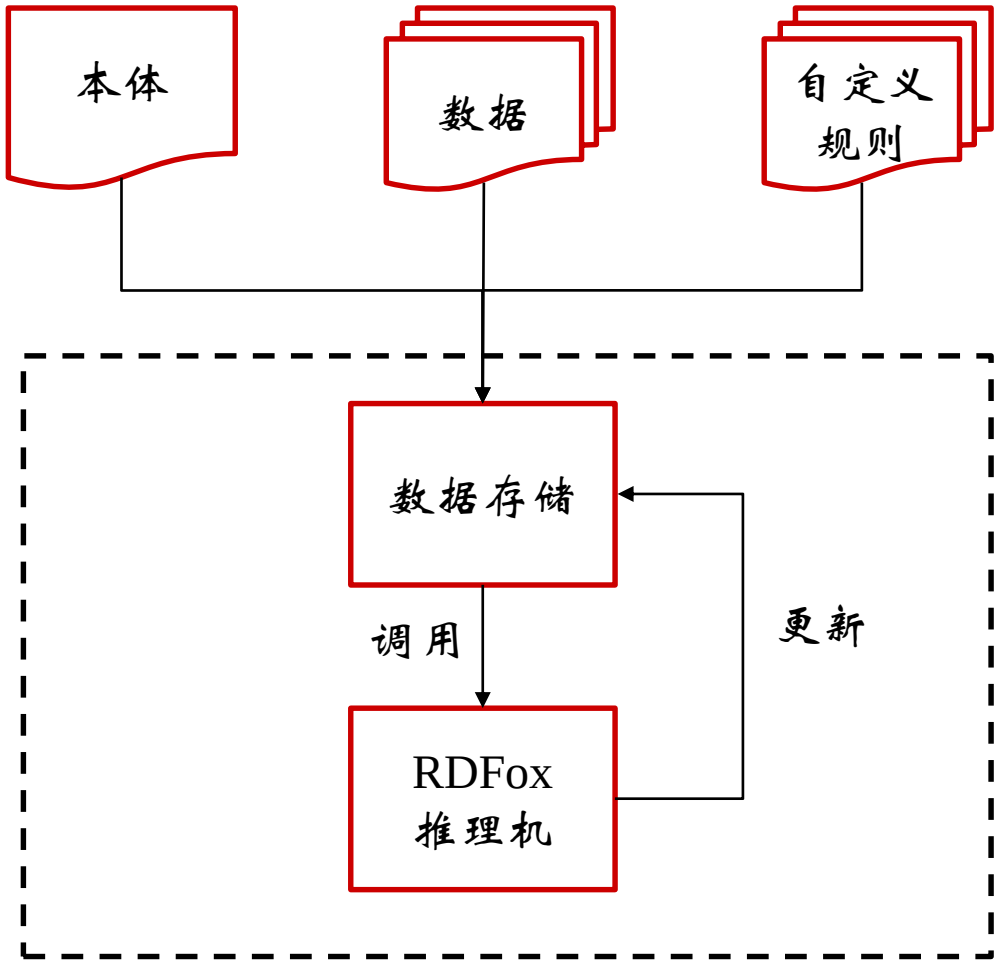
```
store.importOntology(ontology);
store.applyReasoning();
```

相关工具简介

相关工具总结

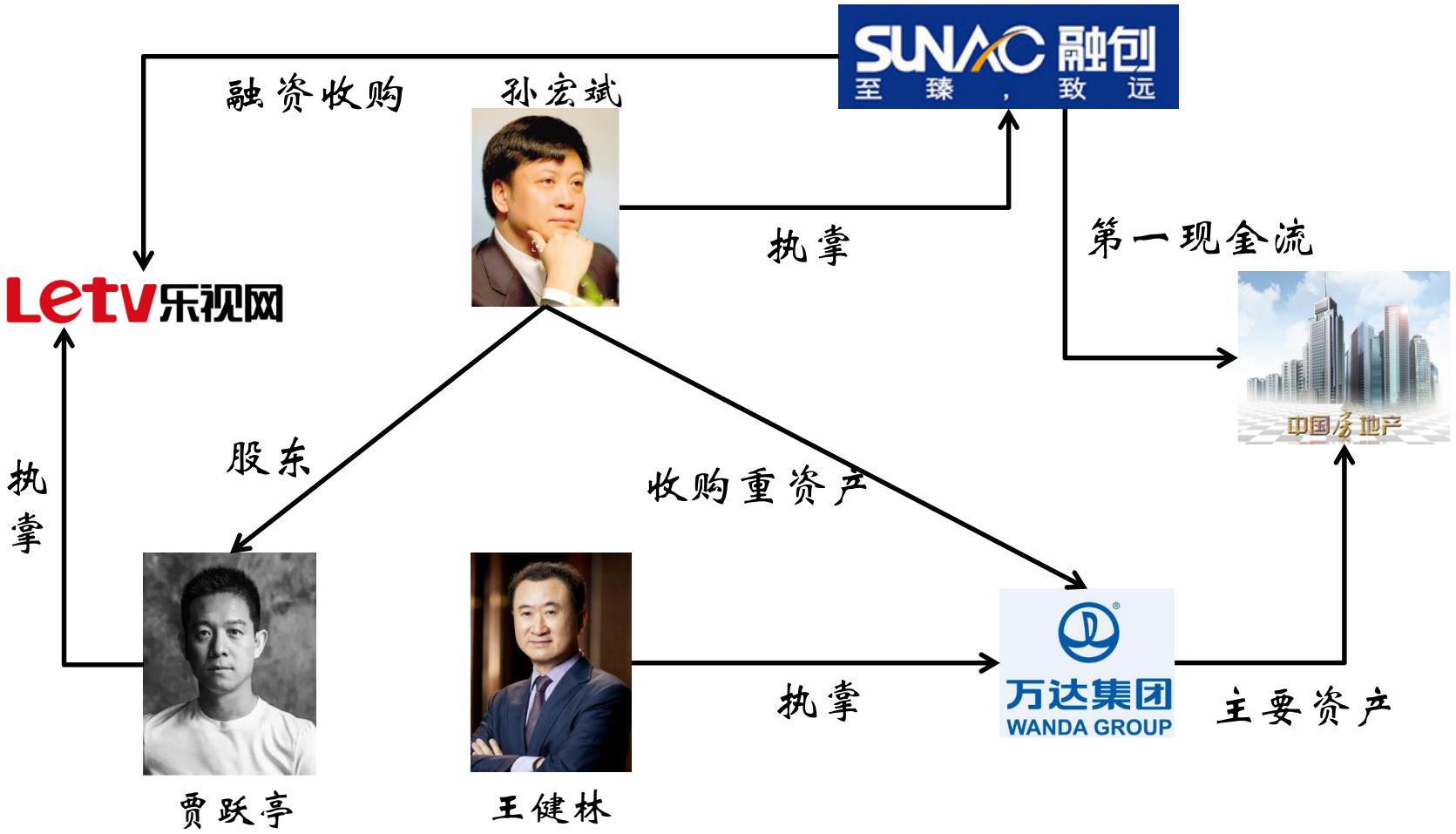
工具名称	支持本体语言	实现编程语言	支持编程语言
KAON2	OWL DL/SWRL	Java	Java
RDFox	OWL 2 RL	C++	Java/C++/Python

RDFox实践



注意：
在RDFox中，
本体(Tbox)与
数据(ABox)需
要分开输入

RDFox实践


融资收购
孙宏斌
SUNAC 融创
至臻，致远
执掌
第一现金流
Letv乐视网
中国房地产
执掌
股东
收购重资产
执掌
万达集团
WANDA GROUP
主要资产
贾跃亭
王健林

RDFox实践

- 本体、数据格式
  - 命名空间
    - finance: http://www.example.org/kse/finance#
  - URI
    - http://www.example.org/kse/finance#孙宏斌
    - 使用命名空间简写为 finance:孙宏斌
  - 三元组
    - finance:融创中国　rdf:type　finance:地产事业

RDFox实践

- 本体 (TBox)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY finance "http://www.example.org/kse/finance#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&finance;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about=""/>

<owl:Class rdf:ID="PublicCompany">
  <rdfs:subClassOf rdf:resource="Company"/>
</owl:Class>


<owl:ObjectProperty rdf:ID="control">
  <rdfs:domain rdf:resource="Person"/>
  <rdfs:range rdf:resource="Company"/>
</owl:ObjectProperty>

</rdf:RDF>
```

RDFox实战

数据 (ABox)

```
<http://www.example.org/kse/finance#融创中国> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.example.org/kse/finance#地产事业> .
<http://www.example.org/kse/finance#孙宏斌> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#融创中国> .
<http://www.example.org/kse/finance#贾跃亭> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#乐视网> .
<http://www.example.org/kse/finance#王健林> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#万达集团> .
<http://www.example.org/kse/finance#孙宏斌> <http://www.example.org/kse/finance#hold_share> <http://www.example.org/kse/finance#乐视网> .
<http://www.example.org/kse/finance#万达集团> <http://www.example.org/kse/finance#main_income> <http://www.example.org/kse/finance#地产事业> .
<http://www.example.org/kse/finance#融创中国> <http://www.example.org/kse/finance#acquire> <http://www.example.org/kse/finance#乐视网> .
<http://www.example.org/kse/finance#融创中国> <http://www.example.org/kse/finance#acquire> <http://www.example.org/kse/finance#万达集团> .
```

自定义规则

1）执掌一家公司就一定是这家公司的股东；
2）某人同时是两家公司的股东，那么这两家公司一定有关联交易；

形式化为

finance:hold_share(X,Y) :- finance:control(X,Y).
finance:conn_trans(Y,Z) :- finance:hold_share(X,Y), finance:hold_share(X,Z).

RDFox实战

代码 (Java)

数据导入

读取本体、数据，声明规则

```
File ontologyFile = new File(RDFox_tutorial.class.getResource("/data/finance/finance.owl").toURI());
File dataFile = new File(RDFox_tutorial.class.getResource("/data/finance/finance-data.ttl").toURI());

String userDefinedRules = "PREFIX p: <http://www.example.org/kse/finance#>"
        + "p:hold_share(?X, ?Y) :- p:control(?X, ?Y)."
        + "p:conn_trans(?Y, ?Z) :- p:hold_share(?X, ?Y), p:hold_share(?X, ?Z).";

OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
DataStore store = new DataStore(DataStore.StoreType.ParallelSimpleNN);
store.setNumberOfThreads(2);
OWLOntology ontology = manager.loadOntologyFromOntologyDocument(ontologyFile);

store.importOntology(ontology);
store.importFiles(new File[] {dataFile});
store.importText(userDefinedRules);
```

创建数据存储对象

将本体、数据、规则导入存储

RDFox实战

- 代码 (Java)
  - 推理

定义命名空间与查询操作
(用于输出当前三元组)

```
Prefixes prefixes = Prefixes.DEFAULT_IMMUTABLE_INSTANCE;
TupleIterator tupleIterator = store.compileQuery("SELECT DISTINCT ?x ?y ?z WHERE{ ?x ?y ?z }", prefixes);
System.out.println("Retrieving all triples before materialisation.");
evaluateAndPrintResults(prefixes, tupleIterator);

store.applyReasoning();

System.out.println("Retrieving all triples after materialisation.");
evaluateAndPrintResults(prefixes, tupleIterator);
```

进行推理

RDFox实战

结果输出

注：RDFox的推理是实例化(materialization)结合规则推理

```
Retrieving all triples before materialisation.
================================================================================
<http://www.example.org/kse/finance#融创中国>  <http://www.example.org/kse/finance#acquire>  <http://www.example.org/kse/finance#万达集团>
<http://www.example.org/kse/finance#融创中国>  <http://www.example.org/kse/finance#acquire>  <http://www.example.org/kse/finance#乐视网>
<http://www.example.org/kse/finance#万达集团>  <http://www.example.org/kse/finance#main_income>  <http://www.example.org/kse/finance#地产事业>
<http://www.example.org/kse/finance#孙宏斌>  <http://www.example.org/kse/finance#hold_share>  <http://www.example.org/kse/finance#乐视网>
<http://www.example.org/kse/finance#王健林>  <http://www.example.org/kse/finance#control>  <http://www.example.org/kse/finance#万达集团>
<http://www.example.org/kse/finance#贾跃亭>  <http://www.example.org/kse/finance#control>  <http://www.example.org/kse/finance#乐视网>
<http://www.example.org/kse/finance#孙宏斌>  <http://www.example.org/kse/finance#control>  <http://www.example.org/kse/finance#融创中国>
<http://www.example.org/kse/finance#融创中国>  rdf:type  <http://www.example.org/kse/finance#地产事业>
--------------------------------------------------------------------------------
  The number of rows returned: 8
================================================================================

Retrieving all triples after materialisation.
================================================================================
<http://www.example.org/kse/finance#万达集团>  <http://www.example.org/kse/finance#conn_trans>  <http://www.example.org/kse/finance#万达集团>
<http://www.example.org/kse/finance#融创中国>  <http://www.example.org/kse/finance#conn_trans>  <http://www.example.org/kse/finance#乐视网>
<http://www.example.org/kse/finance#融创中国>  <http://www.example.org/kse/finance#conn_trans>  <http://www.example.org/kse/finance#融创中国>
<http://www.example.org/kse/finance#乐视网>  <http://www.example.org/kse/finance#conn_trans>  <http://www.example.org/kse/finance#乐视网>
<http://www.example.org/kse/finance#乐视网>  <http://www.example.org/kse/finance#conn_trans>  <http://www.example.org/kse/finance#融创中国>
<http://www.example.org/kse/finance#万达集团>  rdf:type  <http://www.example.org/kse/Company>
<http://www.example.org/kse/finance#王健林>  rdf:type  <http://www.example.org/kse/Person>
<http://www.example.org/kse/finance#王健林>  <http://www.example.org/kse/finance#hold_share>  <http://www.example.org/kse/finance#万达集团>
<http://www.example.org/kse/finance#乐视网>  rdf:type  <http://www.example.org/kse/Company>
<http://www.example.org/kse/finance#贾跃亭>  rdf:type  <http://www.example.org/kse/Person>
<http://www.example.org/kse/finance#贾跃亭>  <http://www.example.org/kse/finance#hold_share>  <http://www.example.org/kse/finance#乐视网>
<http://www.example.org/kse/finance#融创中国>  rdf:type  <http://www.example.org/kse/Company>
<http://www.example.org/kse/finance#孙宏斌>  rdf:type  <http://www.example.org/kse/Person>
<http://www.example.org/kse/finance#孙宏斌>  <http://www.example.org/kse/finance#hold_share>  <http://www.example.org/kse/finance#融创中国>
<http://www.example.org/kse/finance#融创中国>  <http://www.example.org/kse/finance#acquire>  <http://www.example.org/kse/finance#万达集团>
<http://www.example.org/kse/finance#融创中国>  <http://www.example.org/kse/finance#acquire>  <http://www.example.org/kse/finance#乐视网>
<http://www.example.org/kse/finance#万达集团>  <http://www.example.org/kse/finance#main_income>  <http://www.example.org/kse/finance#地产事业>
<http://www.example.org/kse/finance#孙宏斌>  <http://www.example.org/kse/finance#hold_share>  <http://www.example.org/kse/finance#乐视网>
<http://www.example.org/kse/finance#王健林>  <http://www.example.org/kse/finance#control>  <http://www.example.org/kse/finance#万达集团>
<http://www.example.org/kse/finance#贾跃亭>  <http://www.example.org/kse/finance#control>  <http://www.example.org/kse/finance#乐视网>
<http://www.example.org/kse/finance#孙宏斌>  <http://www.example.org/kse/finance#control>  <http://www.example.org/kse/finance#融创中国>
<http://www.example.org/kse/finance#融创中国>  rdf:type  <http://www.example.org/kse/finance#地产事业>
--------------------------------------------------------------------------------
  The number of rows returned: 22
================================================================================
```

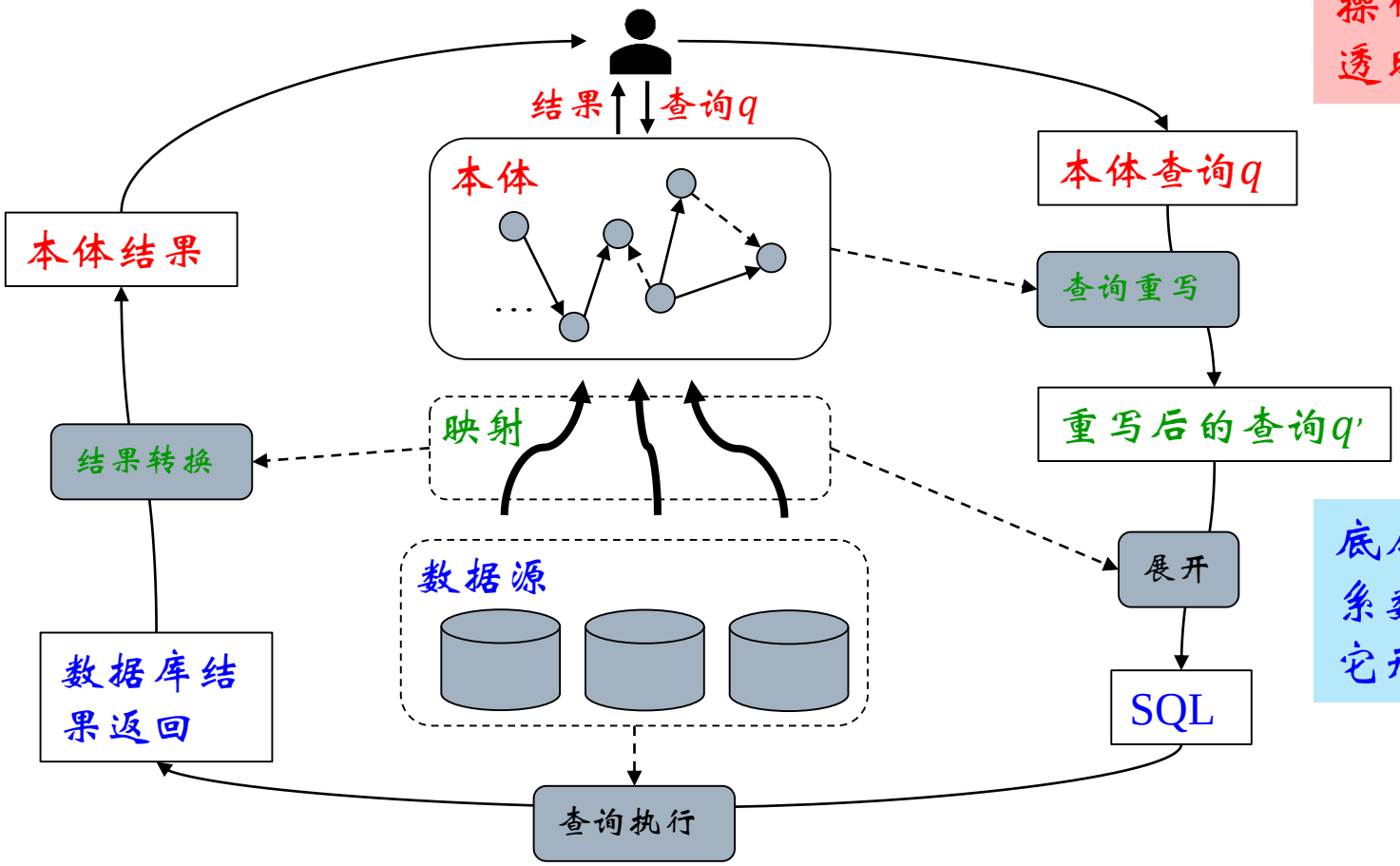
基于一阶查询重写的方法

- 查询重写的目的
  - 高效地结合不同数据格式的数据源
  - 重写方法关联起了不同的查询语言
- 一阶查询

  $q(x) \leftarrow worksFor(x, y), Project(y)$
  - 具有一阶逻辑形式的查询语言
  - Datalog是数据库的一种查询语言，同时具有一阶逻辑形式
- 针对本体基于一阶查询进行重写
  - 以Datalog语言为中间语言，首先重写SPARQL语言为Datalog，再将Datalog重写为SQL查询

基于一阶查询重写的方法

查询重写的基本流程



用户只针对本体层操作；底层对用户透明

通过重写关联

底层数据可以是关系数据库，或者其它形式数据源

基于一阶查询重写的方法

查询重写举例

给定如下本体，查询所有研究人员及其所从事的项目？

Coordinator	⊑	Researcher
∃worksFor	⊑	Researcher
∃worksFor⁻	⊑	Project
Researcher	⊑	∃worksFor
Project	⊑	∃worksFor⁻
∃name⁻	⊑	xsd:String
Researcher	⊑	∃name
Project	⊑	∃name

```
SELECT ?r ?p
WHERE {
        ?r exp:worksFor ?p.
        ?p rdf:type exp:Project.
}
```

用户输入如上SPAQRL语言

基于一阶查询重写的方法

查询重写举例

给定如下本体，查询所有研究人员及其所从事的项目？查询协调专员(coordinator)?

$$
\begin{aligned}
\text{Coordinator} &\sqsubseteq \text{Researcher} \\
\exists \text{worksFor} &\sqsubseteq \text{Researcher} \\
\exists \text{worksFor}^- &\sqsubseteq \text{Project} \\
\text{Researcher} &\sqsubseteq \exists \text{worksFor} \\
\text{Project} &\sqsubseteq \exists \text{worksFor}^- \\
\exists \text{name}^- &\sqsubseteq \text{xsd:String} \\
\text{Researcher} &\sqsubseteq \exists \text{name} \\
\text{Project} &\sqsubseteq \exists \text{name}
\end{aligned}
$$

RESEARCHER

ID	name	project	type
48132	Tom	1	0
61934	Sam	1	0
41257	Lucy	1	1
51943	Mike	2	0

PROJECT

ID	name
1	Optique
2	LOD
3	SemSearch
4	Ontop

底层数据具体为某数据库中如上的两张表

基于一阶查询重写的方法

查询重写举例

步骤一：重写为Datalog查询

```
SELECT ?r ?p
WHERE {
        ?r exp:worksFor ?p.
        ?p rdf:type exp:Project.
}
```

过滤不需要的公理

Coordinator	⊑	Researcher
∃worksFor	⊑	Researcher
∃worksFor⁻	⊑	Project
Researcher	⊑	∃worksFor
Project	⊑	∃worksFor⁻
∃name⁻	⊑	xsd:String
Researcher	⊑	∃name
Project	⊑	∃name

基于一阶查询重写的方法

查询重写举例

步骤一：重写为Datalog查询

$$
\begin{aligned}
\text{Coordinator} &\sqsubseteq \text{Researcher} \\
\exists \text{worksFor} &\sqsubseteq \text{Researcher} \\
\exists \text{worksFor}^- &\sqsubseteq \text{Project} \\
\text{Researcher} &\sqsubseteq \exists \text{worksFor} \\
\text{Project} &\sqsubseteq \exists \text{worksFor}^- \\
\sout{\exists \text{name}^-} &\sout{\sqsubseteq \text{xsd:String}} \\
\sout{\text{Researcher}} &\sout{\sqsubseteq \exists \text{name}} \\
\sout{\text{Project}} &\sout{\sqsubseteq \exists \text{name}}
\end{aligned}
$$

过滤不需要的公理（通过语法层过滤）

```
SELECT ?r ?p
WHERE {
        ?r exp:worksFor ?p.
        ?p rdf:type exp:Project.
}
```

基于一阶查询重写的方法

查询重写举例

步骤一：重写为Datalog查询

```
SELECT ?r ?p
WHERE {
        ?r exp:worksFor ?p.
        ?p rdf:type exp:Project.
}
```

Coordinator	⊑	Researcher
∃worksFor	⊑	Researcher
∃worksFor⁻	⊑	Project
Researcher	⊑	∃worksFor
Project	⊑	∃worksFor⁻
~~∃name⁻~~	~~⊑~~	~~xsd:String~~
~~Researcher~~	~~⊑~~	~~∃name~~
~~Project~~	~~⊑~~	~~∃name~~

过滤不需要的公理(通过语法层过滤)

生成所有相关的Datalog查询

$q(x) \leftarrow worksFor(x, y), Project(y)$
$q(x) \leftarrow worksFor(x, y), worksFor(\_, y)$
$q(x) \leftarrow worksFor(x, \_)$
$q(x) \leftarrow Researcher(x)$
$q(x) \leftarrow Coordinator(x)$

基于一阶查询重写的方法

查询重写举例

步骤二：将数据库关系表达式映射成Datalog原子

RESEARCHER

ID	name	project	type
48132	Tom	1	0
61934	Sam	1	0
41257	Lucy	1	1
51943	Mike	2	0

$$
\begin{aligned}
\text{Researcher}(x) &\leftarrow \text{RESEARCHER}(x, \_, \_, \_)\\
\text{Coordinator}(x) &\leftarrow \text{RESEARCHER}(x, \_, \_, 1)\\
\text{Project}(x) &\leftarrow \text{PROJECT}(x, \_)\\
\text{worksFor}(x, y) &\leftarrow \text{RESEARCHER}(x, \_, y, \_)\\
\text{name}(x, y) &\leftarrow \text{RESEARCHER}(x, y, \_, \_)\\
\text{name}(x, y) &\leftarrow \text{PROJECT}(x, y)
\end{aligned}
$$

PROJECT

ID	name
1	Optique
2	LOD
3	SemSearch
4	Ontop

基于一阶查询重写的方法

查询重写举例

步骤二：将数据库关系表达式映射成Datalog原子

RESEARCHER

ID	name	project	type
48132	Tom	1	0
61934	Sam	1	0
41257	Lucy	1	1
51943	Mike	2	0

PROJECT

ID	name
1	Optique
2	LOD
3	SemSearch
4	Ontop

$$
\begin{aligned}
Researcher(x) &\leftarrow RESEARCHER(x,\_,\_,\_)\\
Coordinator(x) &\leftarrow RESEARCHER(x,\_,\_,1)\\
Project(x) &\leftarrow PROJECT(x,\_)\\
worksFor(x,y) &\leftarrow RESEARCHER(x,\_,y,\_)\\
name(x,y) &\leftarrow RESEARCHER(x,y,\_,\_)\\
name(x,y) &\leftarrow PROJECT(x,y)
\end{aligned}
$$

Datalog原子

对应数据库关系模式

基于一阶查询重写的方法

查询重写举例

步骤三：将从SPARQL以及数据库重写过来的Datalog规则整合进行查询

q(x) ← worksFor(x, y),Project(y)
q(x) ← worksFor(x, y),worksFor(_, y)
q(x) ← worksFor(x, _)
q(x) ← Researcher(x)
q(x) ← Coordinator(x)

Researcher(x) ← RESEARCHER(x , _ , _ , _)
Coordinator(x) ← RESEARCHER(x , _ , _ , 1)
Project(x) ← PROJECT(x , _)
worksFor(x , y) ← RESEARCHER(x , _ , y , _)
name(x , y) ← RESEARCHER(x , y , _ , _)
name(x , y) ← PROJECT(x , y)

基于一阶查询重写的方法

查询重写举例

步骤三：将从SPARQL以及数据库重写过来的Datalog规则整合进行查询

$$q(x) \leftarrow worksFor(x, y), Project(y)$$
$$q(x) \leftarrow worksFor(x, y), worksFor(\_, y)$$
$$q(x) \leftarrow worksFor(x, \_)$$
$$q(x) \leftarrow Researcher(x)$$
$$q(x) \leftarrow Coordinator(x)$$

$$Researcher(x) \leftarrow RESEARCHER(x, \_, \_, \_)$$
$$Coordinator(x) \leftarrow RESEARCHER(x, \_, \_, 1)$$
$$Project(x) \leftarrow PROJECT(x, \_)$$
$$worksFor(x, y) \leftarrow RESEARCHER(x, \_, y, \_)$$
$$name(x, y) \leftarrow RESEARCHER(x, y, \_, \_)$$
$$name(x, y) \leftarrow PROJECT(x, y)$$

1. 查询所有研究人员及其所从事的项目？

$$q(x) \leftarrow worksFor(x, y), Project(y)$$

基于一阶查询重写的方法

查询重写举例

步骤三：将从SPARQL以及数据库重写过来的Datalog规则整合进行查询

$$
\begin{aligned}
q(x) &\leftarrow worksFor(x, y), Project(y)\\
q(x) &\leftarrow worksFor(x, y), worksFor(\_, y)\\
q(x) &\leftarrow worksFor(x, \_)\\
q(x) &\leftarrow Researcher(x)\\
q(x) &\leftarrow Coordinator(x)
\end{aligned}
$$

$$
\begin{aligned}
Researcher(x) &\leftarrow RESEARCHER(x, \_, \_, \_)\\
Coordinator(x) &\leftarrow RESEARCHER(x, \_, \_, 1)\\
Project(x) &\leftarrow PROJECT(x, \_)\\
worksFor(x, y) &\leftarrow RESEARCHER(x, \_, y, \_)\\
name(x, y) &\leftarrow RESEARCHER(x, y, \_, \_)\\
name(x, y) &\leftarrow PROJECT(x, y)
\end{aligned}
$$

1. 查询所有研究人员及其所从事的项目？

$$q(x) \leftarrow worksFor(x, y), Project(y)$$

查询展开

$$
\begin{aligned}
q(x) \leftarrow\ & RESEARCHER(x, \_, y, \_)\\
& PROJECT(y, \_)
\end{aligned}
$$

基于一阶查询重写的方法

查询重写举例

步骤三：将从SPARQL以及数据库重写过来的Datalog规则整合进行查询

$$q(x) \leftarrow worksFor(x, y), Project(y)$$
$$q(x) \leftarrow worksFor(x, y), worksFor(\_, y)$$
$$q(x) \leftarrow worksFor(x, \_)$$
$$q(x) \leftarrow Researcher(x)$$
$$q(x) \leftarrow Coordinator(x)$$

$$Researcher(x) \leftarrow RESEARCHER(x, \_, \_, \_)$$
$$Coordinator(x) \leftarrow RESEARCHER(x, \_, \_, 1)$$
$$Project(x) \leftarrow PROJECT(x, \_)$$
$$worksFor(x, y) \leftarrow RESEARCHER(x, \_, y, \_)$$
$$name(x, y) \leftarrow RESEARCHER(x, y, \_, \_)$$
$$name(x, y) \leftarrow PROJECT(x, y)$$

1. 查询所有研究人员及其所从事的项目？

$$q(x) \leftarrow worksFor(x, y), Project(y)$$

⇩ 查询展开

$$q(x) \leftarrow RESEARCHER(x, \_, y, \_), PROJECT(y, \_)$$

⇩

进一步去数据库进行查询，很容易翻译成SQL语言

基于一阶查询重写的方法

查询重写举例

步骤三：将从SPARQL以及数据库重写过来的Datalog规则整合进行查询

$q(x) \leftarrow worksFor(x, y), Project(y)$
$q(x) \leftarrow worksFor(x, y), worksFor(\_, y)$
$q(x) \leftarrow worksFor(x, \_)$
$q(x) \leftarrow Researcher(x)$
$q(x) \leftarrow Coordinator(x)$

$Researcher(x) \leftarrow RESEARCHER(x, \_, \_, \_)$
$Coordinator(x) \leftarrow RESEARCHER(x, \_, \_, 1)$
$Project(x) \leftarrow PROJECT(x, \_)$
$worksFor(x, y) \leftarrow RESEARCHER(x, \_, y, \_)$
$name(x, y) \leftarrow RESEARCHER(x, y, \_, \_)$
$name(x, y) \leftarrow PROJECT(x, y)$

1. 查询所有协调专员？

$q(x) \leftarrow Coordinator(x)$

基于一阶查询重写的方法

查询重写举例

步骤三：将从SPARQL以及数据库重写过来的Datalog规则整合进行查询

$q(x) \leftarrow worksFor(x, y), Project(y)$
$q(x) \leftarrow worksFor(x, y), worksFor(\_, y)$
$q(x) \leftarrow worksFor(x, \_)$
$q(x) \leftarrow Researcher(x)$
$q(x) \leftarrow Coordinator(x)$

$Researcher(x) \leftarrow RESEARCHER(x, \_, \_, \_)$
$Coordinator(x) \leftarrow RESEARCHER(x, \_, \_, 1)$
$Project(x) \leftarrow PROJECT(x, \_)$
$worksFor(x, y) \leftarrow RESEARCHER(x, \_, y, \_)$
$name(x, y) \leftarrow RESEARCHER(x, y, \_, \_)$
$name(x, y) \leftarrow PROJECT(x, y)$

1. 查询所有协调专员？

$q(x) \leftarrow Coordinator(x)$

查询展开

$q(x) \leftarrow RESEARCHER(x, \_, \_, 1)$

基于一阶查询重写的方法

查询重写举例

步骤三：将从SPARQL以及数据库重写过来的Datalog规则整合进行查询

$$q(x) \leftarrow worksFor(x, y), Project(y)$$
$$q(x) \leftarrow worksFor(x, y), worksFor(\_, y)$$
$$q(x) \leftarrow worksFor(x, \_)$$
$$q(x) \leftarrow Researcher(x)$$
$$q(x) \leftarrow Coordinator(x)$$

$$Researcher(x) \leftarrow RESEARCHER(x, \_, \_, \_)$$
$$Coordinator(x) \leftarrow RESEARCHER(x, \_, \_, 1)$$
$$Project(x) \leftarrow PROJECT(x, \_)$$
$$worksFor(x, y) \leftarrow RESEARCHER(x, \_, y, \_)$$
$$name(x, y) \leftarrow RESEARCHER(x, y, \_, \_)$$
$$name(x, y) \leftarrow PROJECT(x, y)$$

1. 查询所有协调专员？

$$q(x) \leftarrow Coordinator(x)$$

查询展开

$$q(x) \leftarrow RESEARCHER(x, \_, \_, 1)$$

进一步去数据库进行查询，很容易翻译成SQL语言

相关工具介绍

- Ontop
  - 最先进的OBDA系统
  - 兼容RDFS、OWL 2 QL、R2RML、SPARQL标准
  - 支持主流关系数据库：Oracle、MySQL、SQL Server、Postgres
  - 开源 (Apache License 2.0)

http://obda.inf.unibz.it/

基于产生式规则的方法

- 产生式系统
  - 一种前向推理系统，可以按照一定机制执行规则从而达到某些目标，与一阶逻辑类似，也有区别
  - 应用
    - 自动规划
    - 专家系统

Feigenbaum研制的化学分子结构专家系统DENDRAL
Shortliffe研制的的诊断感染性疾病的专家系统MYCIN
…

- 产生式系统的组成
  - 事实集合 (Working Memory)
  - 产生式/规则集合
  - 推理引擎

基于产生式规则的方法

- 事实集/运行内存 (Working Memory, WM)
  - 事实 (WME)的集合
  - 用于存储当前系统中所有事实
- 事实 (Working Memory Element, WME)
  - 描述对象
    - 形如(type $attr_1$: val_1 $attr_2$: val_2 ... $attr_n$: val_n)，其中type, $attr_i$, val_i均为**原子（常量）**
    - 例如: (student name: Alice age: 24)
  - 描述关系 (Refication)
    - 例如: (basicFact relation: olderThan firstArg: John secondArg: Alice)
      简记为(olderThan John Alice)

类比类和对象

基于产生式规则的方法

- 产生式集合 (Production Memory, PM)
  - 产生式的集合
- 产生式
  - **IF** *conditions* **THEN** *actions*
  - *conditions*是由条件组成的集合，又称为LHS
  - *actions*是由动作组成的序列，又称为RHS

基于产生式规则的方法

- LHS
  - 条件 (condition)的集合，各条件之间是且的关系
  - 当LHS中所有条件均被满足，则该规则触发
  - 每个条件形如(type $attr_1$: $spec_1$ $attr_2$: $spec_2$ … $attr_n$: $spec_n$)
    - 其中$spec_i$ 表示对$attr_i$ 的约束，形式可取下列中的一种
      - 原子，如：Alice
      - 变量，如：x (斜体)
      - 表达式，如：$[n+4]$
      - 布尔测试，如：$\{>10\}$
      - 约束的与、或、非操作

(person name: Alice)

(person name: x)

(person age: $[n+4]$)

(person age: $\{>10\}$)

基于产生式规则的方法

- RHS
  - 动作 (action)的序列，执行时依次执行
  - 动作的种类如下：
    - **ADD *pattern***
      - 向WM中加入形如*pattern*的WME
    - **REMOVE *i***
      - 从WM中移除当前规则第*i*个条件匹配的WME
    - **MODIFY *i* *(attr spec)***
      - 对于当前规则第*i*个条件匹配的WME，将其对应于*attr*属性的值改为*spec*

基于产生式规则的方法

- 产生式 LHS RHS
  - **IF** *conditions* **THEN** *actions*
  - 例如：

    IF (Student name: x)
    Then ADD (Person name: x)

    亦可写作(具体语法因不同系统而异)
    (Student name: x) ⇒ ADD (Person name: x)

如果有一个学生名为?x，那么向事实集中加入一个事实，表示有一个名为?x的人

基于产生式规则的方法

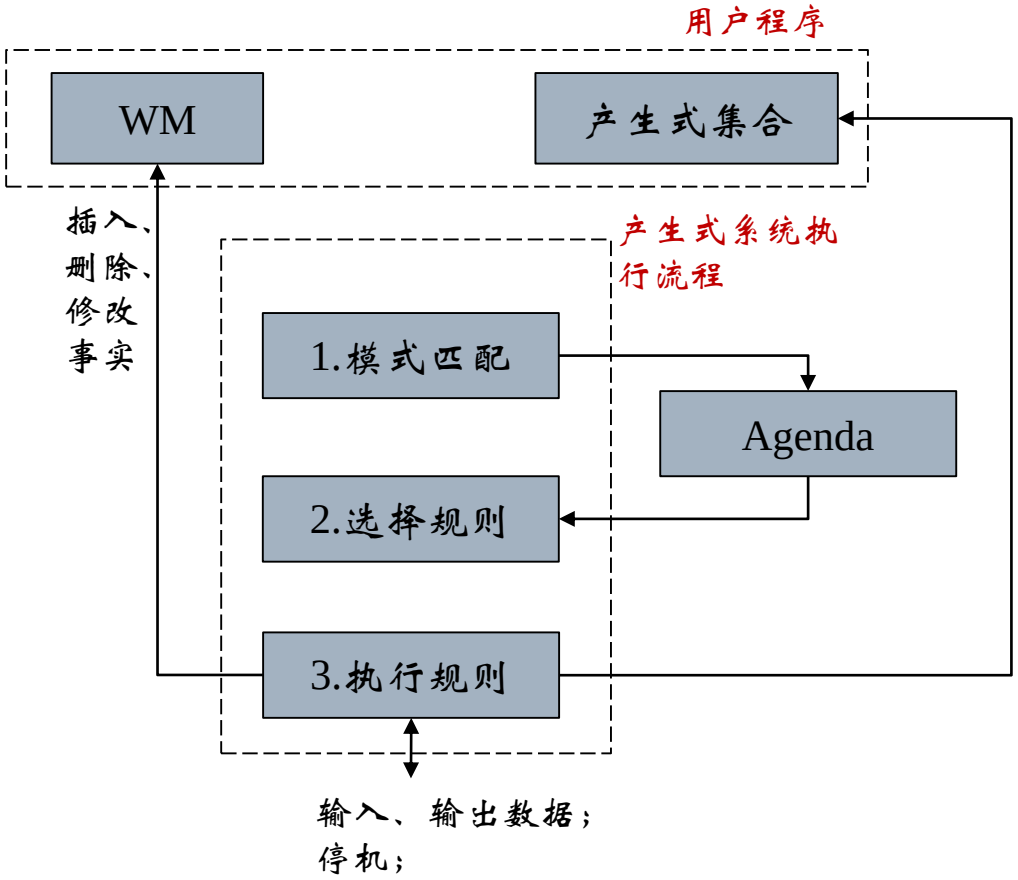
- 推理引擎
  - 控制系统的执行
    - 模式匹配
      - 用规则的条件部分匹配事实集中的事实，整个LHS都被满足的规则被触发，并被加入议程(agenda)
    - 解决冲突
      - 按一定的策略从被触发的多条规则中选择一条
    - 执行动作
      - 执行被选择出来的规则的RHS，从而对WM进行一定的操作

该步骤是产生式系统的核心，具体算法在后面介绍

产生式系统=事实集+产生式集合+推理引擎

基于产生式规则的方法

产生式系统执行流程

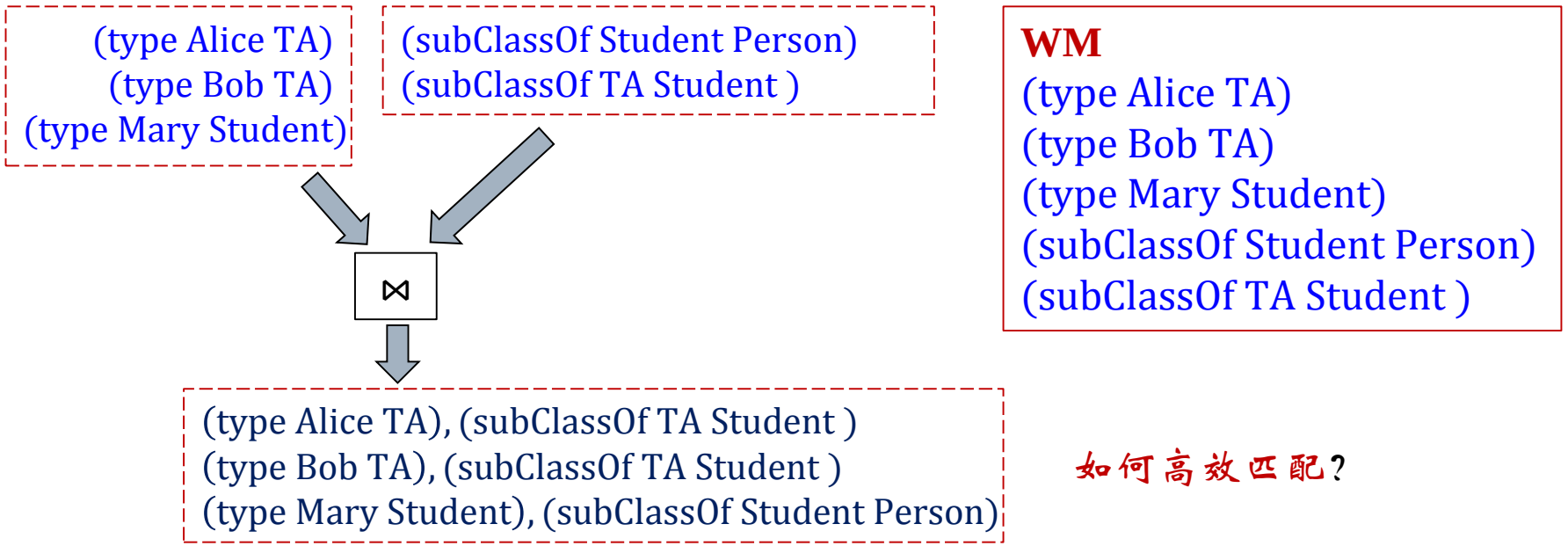


基于产生式规则的方法

模式匹配

- 用每条规则的条件部分匹配当前WM

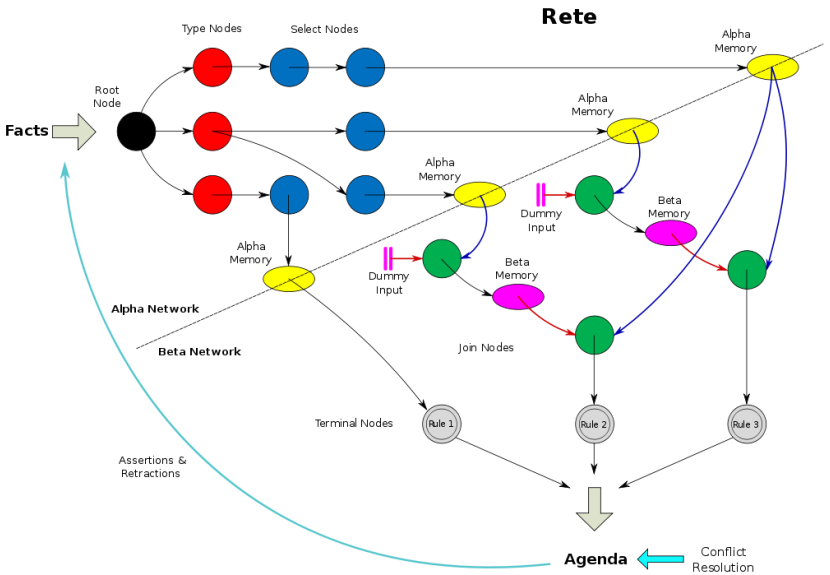
$$(\text{type } x\ y), (\text{subClassOf } y\ z) \Rightarrow \text{ADD } (\text{type } x\ z)$$



如何高效匹配?

基于产生式规则的方法

- 模式匹配
  - RETE算法
    - 1979年由Charles Forgy (CMU)提出
    - 将产生式的LHS组织成判别网络形式
    - 用空间换时间

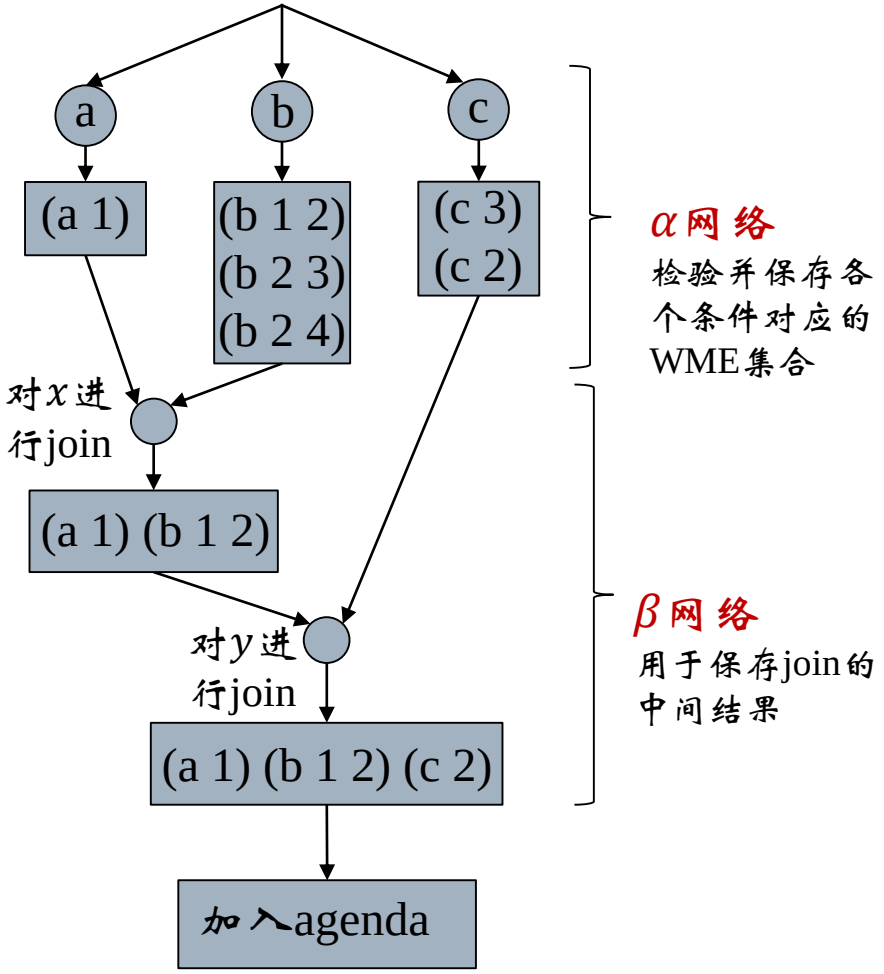


基于产生式规则的方法

- 模式匹配
  - RETE算法

Production Memory
(a x), (b x y), (c y) ⇒ …

Working Memory
(a 1)
(b 1 2)
(b 2 3)
(b 2 4)
(c 3)
(c 2)



基于产生式规则的方法

- 冲突解决
  - 从被触发的多条规则中选择一条
  - 常见策略
    - 随机选择

      在推理的场景下，被触发的多条规则可全被执行
      - 从被触发的规则中随机选择一条执行
    - 具体性 (specificity)
      - 选择最具体的规则

      (Student name: x) ⇒ …

      (Student name: x age: 20) ⇒ …
    - 新近程度 (recency)
      - 选择最近没有被触发的规则执行动作

相关工具介绍

规则格式

```
rule "name"
    attributes
    when
        LHS
    then
        RHS
end
```

Drools

- 商用规则管理系统，其中提供了一个规则推理引擎
- 核心算法基于RETE算法改进
- 提供规则定义语言，支持嵌入Java代码

使用举例

https://www.drools.org/

创建容器与会话

```
KieServices ks = KieServices.Factory.get();
KieContainer kContainer = ks.getKieClasspathContainer();
KieSession kSession = kContainer.newKieSession("ksession-rules");
```

触发规则

```
kSession.fireAllRules();
```

相关工具介绍

Jena

- 用于构建语义网应用的Java框架
- 提供了处理RDF、RDFS、OWL数据的接口，还提供了一个规则引擎
- 提供三元组的内存存储于查询

http://jena.apache.org/

```
[ruleHasStudent: (?s :hasClass ?c) (?p :teaches ?c) -> (?p :hasStudent ?s)]
```

规则格式

使用举例

```
Model m = ModelFactory.createDefaultModel();
```

创建模型

```
Reasoner reasoner = new
GenericRuleReasoner(Rule.rulesFromURL("file:rule.txt"));
InfModel inf = ModelFactory.createInfModel(reasoner, m);
```

创建规则推理机

相关工具介绍

```
CONSTRUCT { ?p :relatesTo
:Cryptography } WHERE { { :Bob ?p
:Alice } UNION { :Alice ?p :Bob } }
```

规则格式
(SPARQL)

- RDF4J (原Sesame)
  - 一个处理RDF数据的开源框架
  - 支持语义数据的解析、存储、推理和查询
  - 能够关联几乎所有RDF存储系统
  - 能够用于访问远程RDF存储

http://rdf4j.org/

- 使用举例

创建规则

```
String pre = "PREFIX : <http://foo.org/bar#>\n";
String rule = pre + "CONSTRUCT { ?p :relatesTo :Cryptography }
WHERE " + "{ { :Bob ?p :Alice } UNION { :Alice ?p :Bob } }";
```

创建推理知识库

```
Repository repo = new SailRepository(new
CustomGraphQueryInferencer(new MemoryStore(),
QueryLanguage.SPARQL, rule, “”));
```

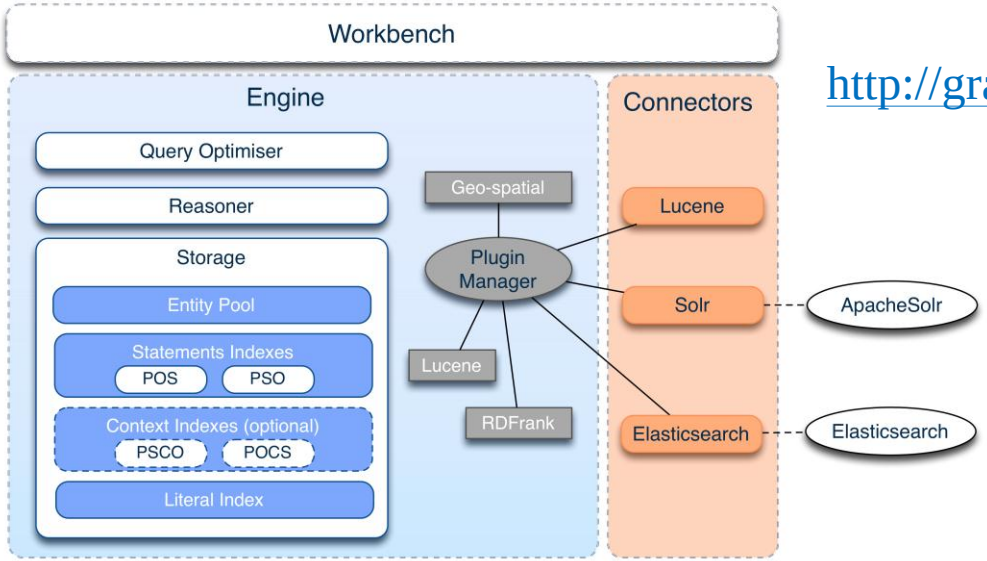
相关工具介绍

```
Id: rdfs2
  x a y [Constraint a != <rdf:type>]
  a <rdfs:domain> z [Constraint z != <rdfs:Resource>]
  ------------------------------
  x  <rdf:type>  z
```

规则格式

GraphDB (原OWLIM)

- 一个可扩展的语义数据存储系统(基于RDF4J)
- 包含三元组存储、推理引擎、查询引擎
- 支持RDFS，OWL DLP，OWL Horst，OWL 2 RL推理



http://graphdb.ontotext.com/

相关工具简介

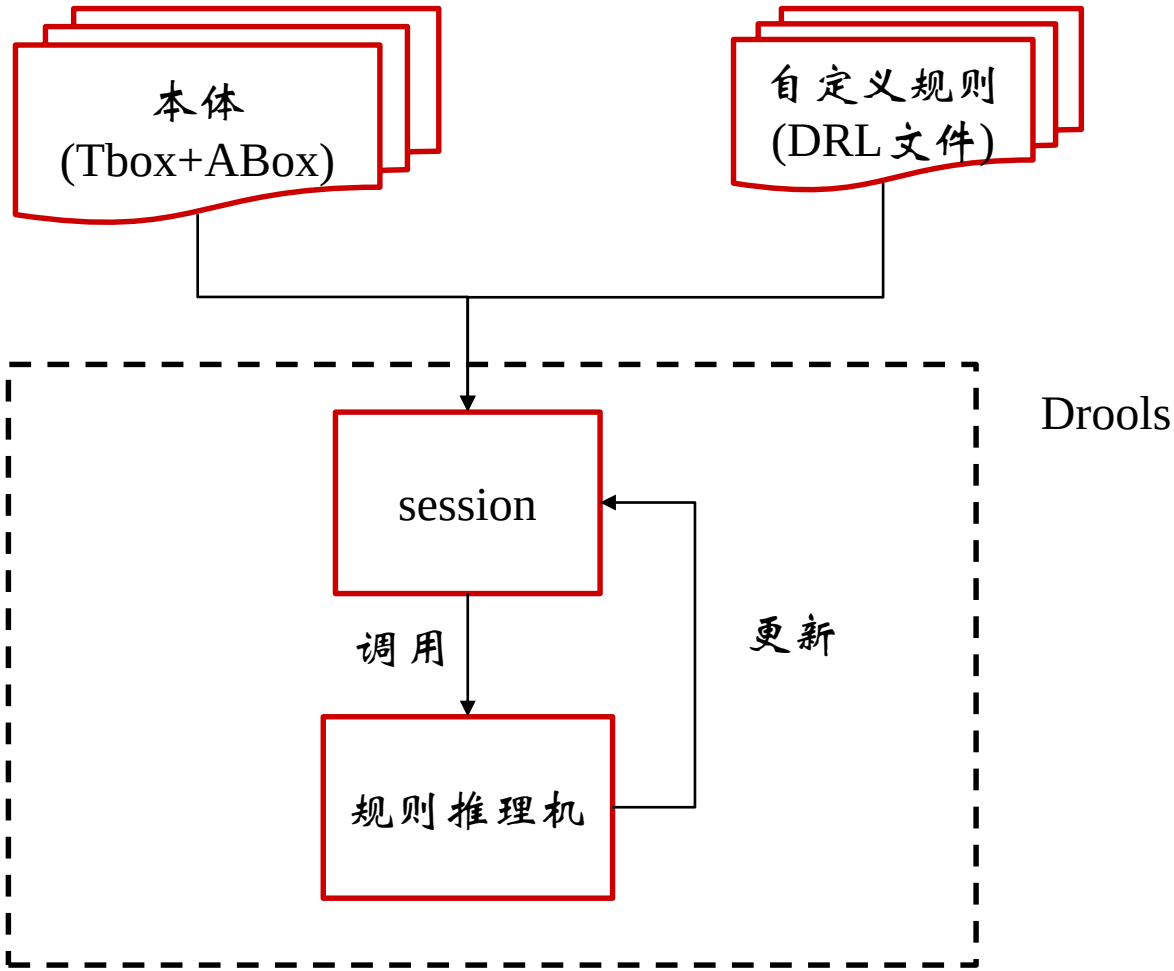
相关工具总结

工具名称	实现语言	支持编程语言	是否开源	算法
Drools	Java	Java	✓	RETEOO/PHREAK
Jena	Java	Java	✓	RETE
RDF4J	Java	Java/PHP/Python	✓	RETE
GraphDB	Java	Java/PHP/Python/Scala…	✗	TRREE

其中，Drools是通用规则管理系统，而另外三个均是图数据库

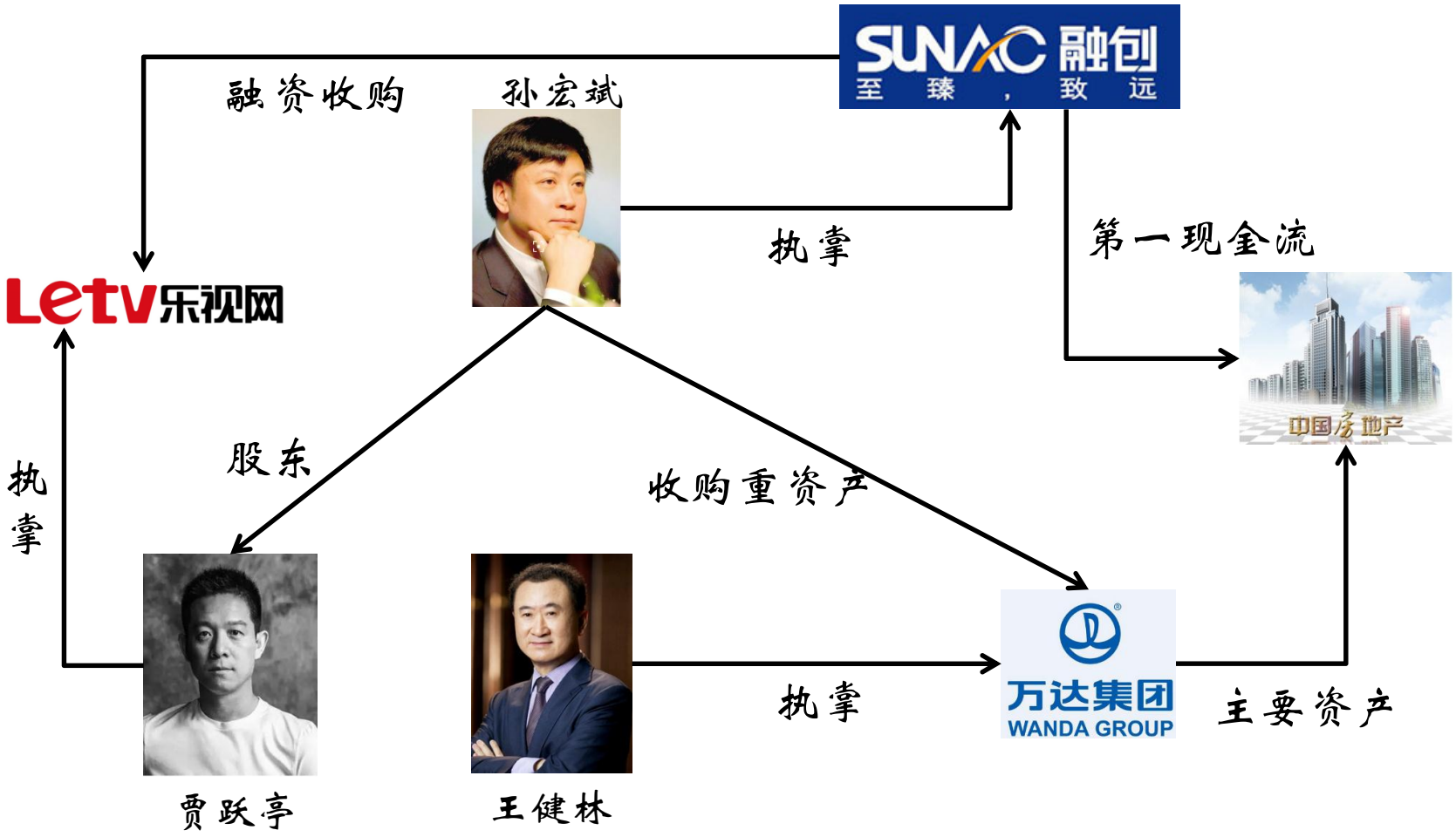
TRREE stands for ‘Triple Reasoning and Rule Entailment Engine’.

Drools实践



注意：
由于Drools不仅仅面向语义数据，因此所有三元组均需以对象形式输入

Drools实践


融资收购
孙宏斌
SUNAC 融创
至臻，致远
执掌
第一现金流
Letv乐视网
中国房地产
股东
执掌
收购重资产
执掌
万达集团
WANDA GROUP
主要资产
贾跃亭
王健林

Drools实践

- 输入
  - TBox与ABox均以三元组的形式输入
  - 创建Triple类，每个三元组都以该类的对象输入

本体

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY finance "http://www.example.org/kse/finance#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&finance;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about=""/>

<owl:Class rdf:ID="PublicCompany">
  <rdfs:subClassOf rdf:resource="Company"/>
</owl:Class>


<owl:ObjectProperty rdf:ID="control">
  <rdfs:domain rdf:resource="Person"/>
  <rdfs:range rdf:resource="Company"/>
</owl:ObjectProperty>

</rdf:RDF>
```

```
public class Triple {
    private String subject;
    private String predicate;
    private String object;

    public Triple(String subject, String predicate, String object) {
        this.subject = subject;
        this.predicate = predicate;
        this.object = object;
    }
```

Triple类

三元组

```
<http://www.example.org/kse/finance> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#Ontology> .
<http://www.example.org/kse/finance#control> <http://www.w3.org/2000/01/rdf-schema#range> <http://www.example.org/kse/Company> .
<http://www.example.org/kse/finance#control> <http://www.w3.org/2000/01/rdf-schema#domain> <http://www.example.org/kse/Person> .
<http://www.example.org/kse/finance#control> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#ObjectProperty> .
<http://www.example.org/kse/finance#PublicCompany> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://www.example.org/kse/Company> .
<http://www.example.org/kse/finance#PublicCompany> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#Class> .
```

Drools实践

- 输入
  - TBox与ABox均以三元组的形式输入
  - 创建Triple类，每个三元组都以该类的对象输入

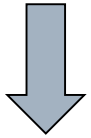
实例

```
<http://www.example.org/kse/finance#融创中国> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.example.org/kse/finance#地产事业> .
<http://www.example.org/kse/finance#孙宏斌> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#融创中国> .
<http://www.example.org/kse/finance#贾跃亭> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#乐视网> .
<http://www.example.org/kse/finance#王健林> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#万达集团> .
<http://www.example.org/kse/finance#孙宏斌> <http://www.example.org/kse/finance#hold_share> <http://www.example.org/kse/finance#乐视网> .
<http://www.example.org/kse/finance#万达集团> <http://www.example.org/kse/finance#main_income> <http://www.example.org/kse/finance#地产事业> .
<http://www.example.org/kse/finance#融创中国> <http://www.example.org/kse/finance#acquire> <http://www.example.org/kse/finance#乐视网> .
<http://www.example.org/kse/finance#融创中国> <http://www.example.org/kse/finance#acquire> <http://www.example.org/kse/finance#万达集团> .
```

Drools实践

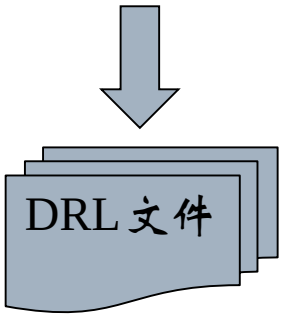
自定义规则

1) 执掌一家公司就一定是这家公司的股东；
2) 某人同时是两家公司的股东，那么这两家公司一定有关联交易；



形式化

```
finance:hold_share(X,Y) :- finance:control(X,Y).
finance:conn_trans(Y,Z) :- finance:hold_share(X,Y), finance:hold_share(X,Z).
```



Drools实践

自定义规则

DRL文件示例

```
1  package data.drools.rules;
2  dialect  "mvel"
3
4  import drools.Triple
5
6  rule "finance1"
7      when
8          c1: Triple($X: subject, predicate == "<http://www.example.org/kse/finance#control>", $Y: object)
9      then
10         insert(new Triple($X, "<http://www.example.org/kse/finance#hold_share>", $Y));
11 end
12
13 rule "finance2"
14     when
15         c1: Triple($X: subject, predicate == "<http://www.example.org/kse/finance#hold_share>", $Y: object)
16         c2: Triple(subject == $X, predicate == "<http://www.example.org/kse/finance#hold_share>", $Z: object)
17     then
18         insert(new Triple($Y, "<http://www.example.org/kse/finance#conn_trans>", $Z))
19 end
```

finance:hold_share(X,Y) :- finance:control(X,Y).

finance:conn_trans(Y,Z) :- finance:hold_share(X,Y), finance:hold_share(X,Z).

Drools实践

Drools工程结构

```
src
  |--main
  |  |--java
  |  |  |--drools
  |  |  |  |--Drools_tutorial.java
  |  |--resources
  |  |  |--META-INF
  |  |  |  |--kmodule.xml
  |  |  |--data
  |  |  |  |--finance.drl
  |  |  |  |--finance-onto.owl
  |  |  |  |--finance-onto.nt
  |  |  |  |--finance-data.nt
```

配置文件，用于指定规则路径

数据

规则在classpath下的路径

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
    <kbase name="rules" packages="data">
        <ksession name="ksession-rules"/>
    </kbase>
</kmodule>
```

配置文件样例

在程序中引用此ksession的名称

- 定义了kmodule、kbase、ksession从上到下的包含关系
- 项目运行时会自动解析classpath中META-INF/kmodule.xml文件，构造相应的对象供Drools引擎使用

Drools实践

代码示例 (Java)

初始化服务及会话

```java
// 获取drools实现的 KieServices 实例
KieServices ks = KieServices.Factory.get();
// kieServices默认加载 classpath:META-INF/kmodule.xml 得到 KieContainer
KieContainer kContainer = ks.getKieClasspathContainer();
// 通过 kContainer获取 kmodule.xml 中定义的 ksession
KieSession kSession = kContainer.newKieSession("ksession-rules");
```

读取本体中的三元组

```java
// 读取本体数据
BufferedReader ontoReader = new BufferedReader(new FileReader(new File(Drools_tutorial.class.getResource("/data/finance-onto.nt").
    toURI())));
String ontoLine = null;
while((ontoLine = ontoReader.readLine()) != null){
    if(ontoLine.isEmpty())
        continue;
    else {
        String[] lineArray = ontoLine.split(" ");
        // 向WorkingMemory插入三元组
        kSession.insert(new Triple(lineArray[0], lineArray[1], lineArray[2]));
    }
}
```

读取数据中的三元组

```java
// 读取实例数据
BufferedReader dataReader = new BufferedReader(new FileReader(new File(Drools_tutorial.class.getResource("/data/finance-data.nt").
    toURI())));
String dataLine = null;
while((dataLine = dataReader.readLine()) != null){
    if(dataLine.isEmpty())
        continue;
    else{
        String[] lineArray = dataLine.split(" ");
        // 向WorkingMemory插入三元组
        kSession.insert(new Triple(lineArray[0], lineArray[1], lineArray[2]));
    }
}
```

Drools实践

代码示例 (Java)

```
System.out.println("Facts num before reasoning: " + kSession.getObjects().toArray().length);
System.out.println("Facts Before Reasoning:");
Object[] array = kSession.getObjects().toArray();
for(int i = 0; i < array.length; i++){
    System.out.println(i + ": " + array[i]);
}

long startTime = System.currentTimeMillis();
System.out.println("Execute...");
kSession.fireAllRules();
long endTime = System.currentTimeMillis();
long runningTime = endTime - startTime;

System.out.println("Facts num after reasoning: " + kSession.getObjects().toArray().length);
System.out.println("Facts After Reasoning:");
Object[] array2 = kSession.getObjects().toArray();
for(int i = 0; i < array2.length; i++){
    System.out.println(i + ": " + array2[i]);
}

System.out.println("Total time cost: " + runningTime + "ms");
```

输出推理前的三元组

进行推理

输出推理后的三元组及推理时间

Drools实践

结果输出

注：此处推理结果与RDFox略有不同是因为Drools只进行规则推理

```
Triples num before reasoning: 14
Triples Before Reasoning:
1: <http://www.example.org/kse/finance#融创中国> <http://www.example.org/kse/finance#acquire> <http://www.example.org/kse/finance#万达集团> .
2: <http://www.example.org/kse/finance#control> <http://www.w3.org/2000/01/rdf-schema#range> <http://www.example.org/kse/Company> .
3: <http://www.example.org/kse/finance#孙宏斌> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#融创中国> .
4: <http://www.example.org/kse/finance#control> <http://www.w3.org/2000/01/rdf-schema#domain> <http://www.example.org/kse/Person> .
5: <http://www.example.org/kse/finance#万达集团> <http://www.example.org/kse/finance#main_income> <http://www.example.org/kse/finance#地产事业> .
6: <http://www.example.org/kse/finance#PublicCompany> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://www.example.org/kse/Company> .
7: <http://www.example.org/kse/finance#融创中国> <http://www.example.org/kse/finance#acquire> <http://www.example.org/kse/finance#乐视网> .
8: <http://www.example.org/kse/finance#control> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#ObjectProperty> .
9: <http://www.example.org/kse/finance#贾跃亭> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#乐视网> .
10: <http://www.example.org/kse/finance#孙宏斌> <http://www.example.org/kse/finance#hold_share> <http://www.example.org/kse/finance#乐视网> .
11: <http://www.example.org/kse/finance#PublicCompany> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#Class> .
12: <http://www.example.org/kse/finance> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#Ontology> .
13: <http://www.example.org/kse/finance#融创中国> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.example.org/kse/finance#地产事业> .
14: <http://www.example.org/kse/finance#王健林> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#万达集团> .
Execute...
Facts num after reasoning: 23
Facts After Reasoning:
1: <http://www.example.org/kse/finance#融创中国> <http://www.example.org/kse/finance#acquire> <http://www.example.org/kse/finance#万达集团> .
2: <http://www.example.org/kse/finance#control> <http://www.w3.org/2000/01/rdf-schema#range> <http://www.example.org/kse/Company> .
3: <http://www.example.org/kse/finance#孙宏斌> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#融创中国> .
4: <http://www.example.org/kse/finance#融创中国> <http://www.example.org/kse/finance#conn_trans> <http://www.example.org/kse/finance#融创中国> .
5: <http://www.example.org/kse/finance#control> <http://www.w3.org/2000/01/rdf-schema#domain> <http://www.example.org/kse/Person> .
6: <http://www.example.org/kse/finance#乐视网> <http://www.example.org/kse/finance#conn_trans> <http://www.example.org/kse/finance#乐视网> .
7: <http://www.example.org/kse/finance#万达集团> <http://www.example.org/kse/finance#main_income> <http://www.example.org/kse/finance#地产事业> .
8: <http://www.example.org/kse/finance#孙宏斌> <http://www.example.org/kse/finance#hold_share> <http://www.example.org/kse/finance#融创中国> .
9: <http://www.example.org/kse/finance#融创中国> <http://www.example.org/kse/finance#conn_trans> <http://www.example.org/kse/finance#乐视网> .
10: <http://www.example.org/kse/finance#PublicCompany> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://www.example.org/kse/Company> .
11: <http://www.example.org/kse/finance#王健林> <http://www.example.org/kse/finance#hold_share> <http://www.example.org/kse/finance#万达集团> .
12: <http://www.example.org/kse/finance#融创中国> <http://www.example.org/kse/finance#acquire> <http://www.example.org/kse/finance#乐视网> .
13: <http://www.example.org/kse/finance#control> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#ObjectProperty> .
14: <http://www.example.org/kse/finance#贾跃亭> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#乐视网> .
15: <http://www.example.org/kse/finance#孙宏斌> <http://www.example.org/kse/finance#hold_share> <http://www.example.org/kse/finance#乐视网> .
16: <http://www.example.org/kse/finance#乐视网> <http://www.example.org/kse/finance#conn_trans> <http://www.example.org/kse/finance#融创中国> .
17: <http://www.example.org/kse/finance#PublicCompany> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#Class> .
18: <http://www.example.org/kse/finance> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#Ontology> .
19: <http://www.example.org/kse/finance#万达集团> <http://www.example.org/kse/finance#conn_trans> <http://www.example.org/kse/finance#万达集团> .
20: <http://www.example.org/kse/finance#贾跃亭> <http://www.example.org/kse/finance#hold_share> <http://www.example.org/kse/finance#乐视网> .
21: <http://www.example.org/kse/finance#乐视网> <http://www.example.org/kse/finance#conn_trans> <http://www.example.org/kse/finance#乐视网> .
22: <http://www.example.org/kse/finance#融创中国> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.example.org/kse/finance#地产事业> .
23: <http://www.example.org/kse/finance#王健林> <http://www.example.org/kse/finance#control> <http://www.example.org/kse/finance#万达集团> .
Total time cost: 58ms
```

基于并行技术的方法

- 单机环境下多核、多处理器技术
  - 多线程
    - [Kazakov, 2011] 利用多线程实现了OWL EL分类
  - GPU
    - [Heino, 2012] 利用GPU技术实现RDFS的大规模推理
- 多机环境下基于网络通信的分布式技术
  - WebPIE [Urbani, 2010]在大集群上可以完成上百亿的RDF三元组的推理
  - [Zhou, 2013]利用MapReduce实现大规模的EL本体推理

相关工具简介

工具名称	实现语言	支持本体语言	并行技术	功能
RDFox	C++	OWL 2 RL	多线程	实例化、查询
DistEL	Java	OWL 2 EL	分布式	分类
DRAOn	Java	OWL	分布式	本体推理、一致性检测
WebPIE	Java	RDFS/OWL ter Horst	分布式	本体推理

WebPIE: http://www.few.vu.nl/~jui200/webpie.html

参考文献

- [Urbani, 2010] Jacopo Urbani , Spyros Kotoulas, Jason Maassen, Frank van Harmelen, Henri E. Bal: OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples. In Proceedings of the Extended Semantic Web Conference, 180-195, 2010.
- [Kazakov, 2011] Yevgeny Kazakov, Markus Krötzsch, Frantisek Simancik: Concurrent Classification of EL Ontologies. In Proceedings of the International Semantic Web Conference, 305-320, 2011.
- [Heino, 2012] Norman Heino, Jeff Z. Pan: RDFS Reasoning on Massively Parallel Hardware. In Proceedings of the International Conference on The Semantic Web, 133-1129, 2012.
- [Zhou, 2013] Zhangquan Zhou, Guilin Qi, Chang Liu, Pascal Hitzler, Raghava Mutharaju: Scale reasoning with fuzzy-EL+ ontologies based on MapReduce. In Proceedings of the IJCAI-2013 Workshop on Weighted Logics for Artificial Intelligence, 87-93, 2013.

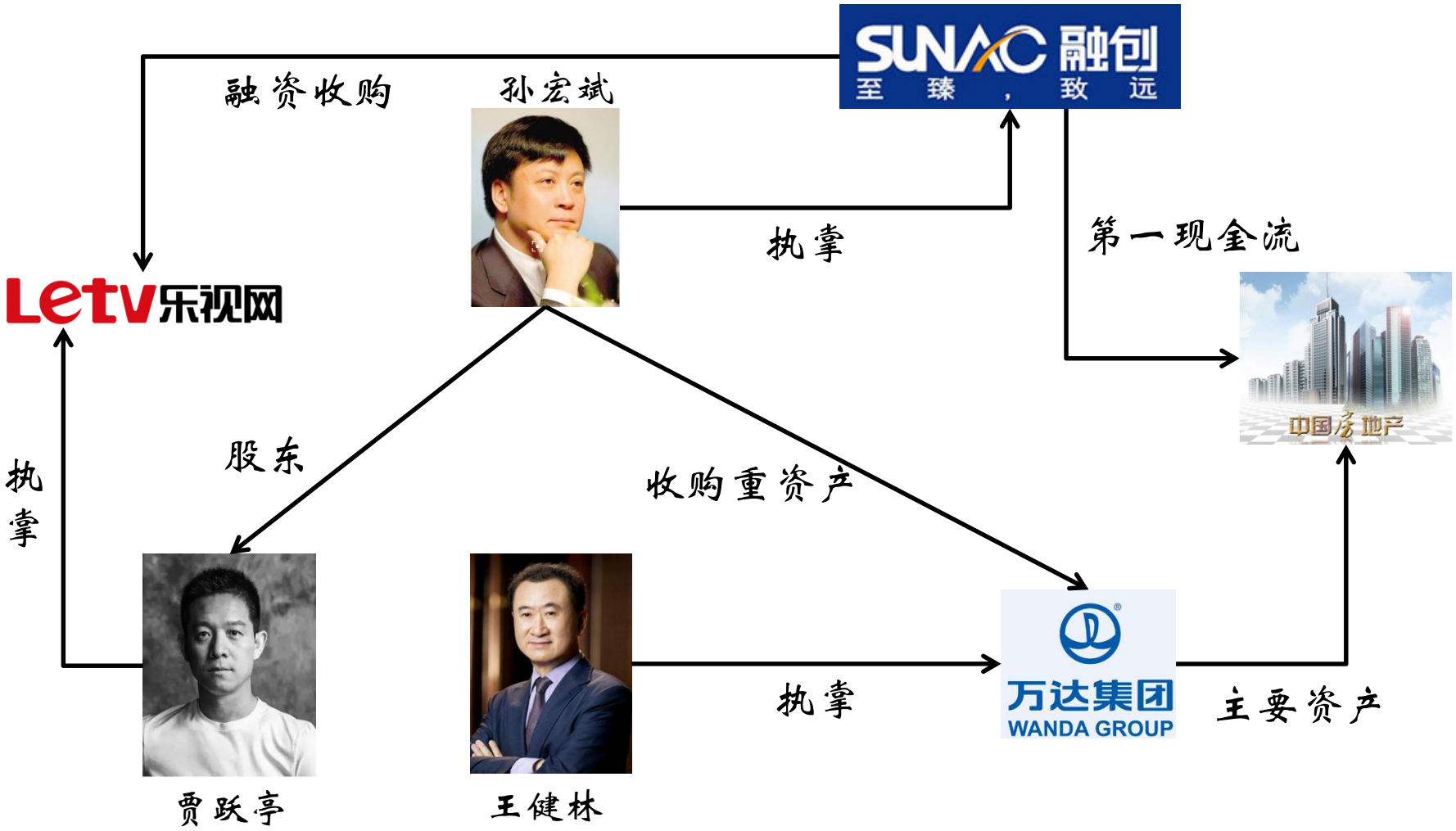
大纲

- 知识推理简介与任务分类
- 本体推理方法与工具介绍
- 实践展示：使用Jena进行知识推理

实践展示

- 使用Jena完成示例图谱知识上的上下位推理、缺失类别补全和一致性检测等

实践展示


SUNAC 融创
至臻，致远
融资收购
孙宏斌
执掌
第一现金流
Letv乐视网
中国房地产
执掌
股东
收购重资产
贾跃亭
王健林
执掌
万达集团
WANDA GROUP
主要资产

实践展示

构建本体

其本质上就是Jena中的知识库结构

- Model: Jena最核心的数据结构

构建一个最简单的Model

```
Model myMod = ModelFactory.createDefaultModel();
```

实践展示

构建本体

其本质上就是Jena中的知识库结构

- Model: Jena最核心的数据结构

构建一个最简单的Model

```
Model myMod = ModelFactory.createDefaultModel();
```

定义我们这个例子的命名空间

```
String finance = “http://www.example.org/kse/finance#”;
```

实践展示

构建本体

其本质上就是Jena中的知识库结构

- Model: Jena最核心的数据结构

构建一个最简单的Model

```
Model myMod = ModelFactory.createDefaultModel();
```

定义我们这个例子的命名空间

```
String finance = “http://www.example.org/kse/finance#”;
```

定义一个个体

```
Resource shb = myMod.createResource(finance + "孙宏斌");
```

实践展示

构建本体

其本质上就是Jena中的知识库结构

- Model: Jena最核心的数据结构

构建一个最简单的Model

```
Model myMod = ModelFactory.createDefaultModel();
```

定义我们这个例子的命名空间

```
String finance = “http://www.example.org/kse/finance#”;
```

定义一个个体

```
Resource rczg = myMod.createResource(finance + "融创中国");
```

实践展示

构建本体

其本质上就是Jena中的知识库结构

- Model: Jena最核心的数据结构

构建一个最简单的Model

```
Model myMod = ModelFactory.createDefaultModel();
```

定义我们这个例子的命名空间

```
String finance = “http://www.example.org/kse/finance#”;
```

定义一个关系

```
Property control = myMod.createProperty(finance + "执掌");
```

实践展示

构建本体

其本质上就是Jena中的知识库结构

- Model: Jena最核心的数据结构

构建一个最简单的Model

```
Model myMod = ModelFactory.createDefaultModel();
```

定义我们这个例子的命名空间

```
String finance = “http://www.example.org/kse/finance#”;
```

往知识库中加入三元组

```
myMod.add(shb, control, rczg);
```

执掌

实践展示

知识库中的部分三元组:

finance :孙宏斌 finance :control finance :融创中国

finance :贾跃亭 finance :control finance :乐视网

finance :融创中国 rdf:type finance :地产公司

finance :地产公司 rdfs:subclassOf finance:公司

finance:公司 rdfs:subclassOf finance:法人实体

finance:孙宏斌 rdf:type finance:公司

finance:孙宏斌 rdf:type finance:人

finance :人 owl:disjointWith finance:公司

实践展示

添加推理机

- Model: 最核心的数据结构

构建一个含推理功能的Model

```
Model myMod = ModelFactory.createDefaultModel();
String finance = “http://www.example.org/kse/finance#”;
Resource shb = myMod.createResource(finance, “孙宏斌");
Resource rczg = myMod.createResource(finance, “融创中国");
Property control = myMod.createProperty(finance, “执掌");
… // 添加三元组，代码省略

InfModel inf_rdfs = ModelFactory.createRDFSModel(myMod);
```

实际上在原来的Model之上加了个RDFS推理机

实践展示

上下位推理

查询的同时已经做出了推理！

查询触发推理

查询输入类别s和o之间有无上下位关系

```
subClassOf(inf_rdfs, myMod.getResource(finance+"地产公司"),
myMod.getResource(finance+"法人实体"));
```

```
public static void subClassOf(Model m, Resource s, Resource o) {
    for (StmtIterator i = m.listStatements(s, RDFS.subClassOf, o); i.hasNext(); ) {
        Statement stmt = i.nextStatement();
        System.out.println(" yes! " );
        break;
    }
}
```

finance:地产公司 rdfs:subclassOf
finance:公司
finance:公司 rdfs:subclassOf
finance:法人实体

→

finance:地产公司 rdfs:subclassOf
finance:法人实体

实践展示

添加推理机

- OWL推理：构建OWL推理机

构建一个含OWL推理功能的Model

```
Model myMod = ModelFactory.createDefaultModel();
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
InfModel inf_owl = ModelFactory.createInfModel(reasoner, myMod);
```

在普通的Model之上加了个OWL推理机

实践展示

针对类别的推理

- 类别补全

OWL推理机可以针对个体类别做出完备推理，即补充完整该个体的所有类别；在查询的时候，可以直接打印出所有类别！

```
printStatements(inf_owl, rczg, RDF.type, null);
```

```
public static void printStatements(Model m, Resource s, Property p, Resource o)
{
    for (StmtIterator i = m.listStatements(s,p,o); i.hasNext(); ) {
        Statement stmt = i.nextStatement();
        System.out.println(" - " + PrintUtil.print(stmt));
    }
}
```

打印个体finance:融创中国所有类别

```
finance:融创中国  rdf:type finance:地产公司 .
finance:地产公司 rdfs:subclassOf finance:公司 .
finance:公司 rdfs:subclassOf finance:法人实体 .
```

→

```
finance:地产公司 .
finance:公司 .
finance:法人实体 .
```

实践展示

不一致检测

- 通过validate接口检测不一致

生成data的不一致检测报告

```
Model data = FileManager.get().loadModel(fname);
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
InfModel inf_owl = ModelFactory.createInfModel(reasoner, myMod);
ValidityReport validity = inf_owl.validate();
```

这里检测不一致基于OWL推理机

实践展示

不一致检测

- 通过validate接口检测不一致

打印不一致具体信息

```
if (validity.isValid()) {
    System.out.println("没有不一致");
} else {
    System.out.println("存在不一致，如下：");
    for (Iterator i = validity.getReports(); i.hasNext(); ) {
        System.out.println(" - " + i.next());
    }
}
```

实践展示

不一致检测举例

- 通过我们的例子找到的不一致

因为用到了OWL词汇，一定要配置OWL推理机！

finance:孙宏斌 rdf:type finance:公司

finance:孙宏斌 rdf:type finance:人

finance:人 owl:disjointWith finance:公司

打印不一致具体信息

存在不一致，如下：
“ finance:孙宏斌 rdf:type finance:公司.
finance:孙宏斌 rdf:type finance:人. ”
are not compatible with
“finance :人 owl:disjointWith finance:公司”

谢谢大家！

本课程课件由OpenKG提供支持

联系我们

小象学院：互联网新技术在线教育领航者

- 微信公众号：小象
- 新浪微博：ChinaHadoop