

第三章 谓词逻辑

戴洪良

计算机科学与技术学院/人工智能学院

hongldai@nuaa.edu.cn



命题逻辑的不足

- All men are mortal.
 - Socrates is a man.
 - Socrates is mortal.
-
- 所有计算机专业的学生都要学AI。
 - 小明是计算机专业的学生。
 - 小明要学AI。

命题逻辑的不足

- 我们想表示的世界有不同的对象 (object) , 对象有属性 (properties) , 对象间有关系 (relation)
- 命题逻辑: 表示关于世界的陈述, 不对以上内容建模
 - 原子命题是最基本的单位, 无法对原子命题进行分解
 - 但不同原子命题蕴含个体、群体和关系等内在丰富语义
- 结果: 有些知识用命题逻辑难以表示
 - 其中两种难以表示的情况:
 - 关于相似对象和关系的陈述
 - 对一组对象的陈述

命题逻辑的不足

- 关于相似对象和关系的陈述，需要一一列举
- 例：人的年龄比较

假设有: *John is older than Mary*
Mary is older than Paul

如要得到 *John is older than Paul* 需要:

John is older than Mary \wedge *Mary is older than Paul* \Rightarrow *John is older than Paul*

假设又加了一条事实: *Jane is older than Mary*

如要得到 *Jane is older than Paul* 又需要:

Jane is older than Mary \wedge *Mary is older than Paul* \Rightarrow *Jane is older than Paul*

有什么问题?

命题逻辑的不足

- 关于相似对象和关系的陈述，需要一一列举
- 例：人的年龄比较

假设有: *John is older than Mary*
Mary is older than Paul

如要得到 *John is older than Paul* 需要:

John is older than Mary \wedge *Mary is older than Paul* \Rightarrow *John is older than Paul*

假设又加了一条事实: *Jane is older than Mary*

如要得到 *Jane is older than Paul* 又需要:

Jane is older than Mary \wedge *Mary is older than Paul* \Rightarrow *Jane is older than Paul*

如果有很多人以及关于他们年龄的事实，就需要很多的规则才能进行推理

命题逻辑的不足

- 关于相似对象和关系的陈述，需要一一列举
- 例：人的年龄比较

假设有: *John is older than Mary*
Mary is older than Paul

可能的解决方法：引入变量

PersA is older than PersB \wedge *PersB is older than PersC* \Rightarrow *PersA is older than PersC*

命题逻辑的不足

- 关于一组对象的描述需要列出其中每个对象
- 例：

假设有表达 *Every student likes vacation*

如果使用命题逻辑，需要对每个学生都进行相应陈述：

John likes vacation \wedge
Mary likes vacation \wedge
Jane likes vacation \wedge
...

如何解决？

命题逻辑的不足

- 关于一组对象的描述需要列出其中每个对象
- 例：

假设有表达 *Every student likes vacation*

如果使用命题逻辑，需要对每个学生都进行相应陈述：

John likes vacation \wedge
Mary likes vacation \wedge
Jane likes vacation \wedge
...

如何解决？ 引入量词 (quantifier)

命题逻辑的不足

- 命题逻辑中，原子命题是最基本的单位，无法对原子命题进行分解
- 不同原子命题蕴含个体、群体和关系等内在丰富语义，命题逻辑无法表现内在丰富语义
- 因此，需要分析原子命题，分离其主语（个体或群体）和谓语（关系）

引入更加强大的逻辑表示方法，这就是谓词逻辑

谓词逻辑

- 在谓词逻辑中，将原子命题进一步细化，分解出个体、谓词和量词，来表达个体与总体的内在联系和数量关系。
- 谓词逻辑中三个核心概念：
 - **个体(individual)、谓词(predicate)和量词(quantifier)**

个体 (individual)

- **定义** 个体 (individual) , 指在原子命题中所描述的对象。个体是指可以独立存在的事物, 它可以是某个抽象的概念, 也可以是一个具体的实体, 如汽车、人工智能、实数、品德等。
- **个体常元 (individual constant)** : , 代表特定的个体, 可以理解为特定个体的名字, 如: a, b, c , 或NUAA、Bob等。
- 任意一个个体都有一个讨论范围, 这个变化范围称为**论域 (universe of discourse或domain of discourse)** , 也称为**个体域**, 通常用符号 D 表示, 个体就相当于论域中的元素。
- 以某个论域为变化范围的表示任意个体的形式符号则称为**个体变元 (individual variable)** , 一般以 x, y, z 等表示。

谓词 (Predicate)

定义 谓词 (predicates)，在原子命题中用以描述个体的性质或个体间关系的部分。

谓词当与一个个体相联系时，刻画的是该个体的属性；当与两个或多个个体相联系时，则刻画的是这些个体之间的关系。可用大写英文字母，如 P ， Q ， R ， L ，或直接用英文单词/短语来表示。

例：

张三是学生。

李四是学生。

其中“...是学生”就是谓词，“张三”“李四”是个体。谓词在这里是用来刻画个体的属性。

用 $P(x)$ 表示“ x 是学生”，其中 x 是个体变元。再假设用 a 表示张三， b 表示李四，它们都是个体常元，则上述两个命题又可以表示成 $P(a)$ 、 $P(b)$ 。

谓词 (Predicate)

定义 谓词 (predicates) , 在原子命题中用以描述个体的性质或个体间关系的部分。

谓词当与一个个体相联系时, 刻画的是该个体的属性; 当与两个或多个个体相联系时, 则刻画的是这些个体之间的关系。可用大写英文字母, 如 P , Q , R , L , 或直接用英文单词/短语来表示。

例:

计算机学院的人工智能课是必修类别。

其中“.....学院的.....课是.....类别”是谓词, 是用来刻画**多个个体**之间的关系的。如果用 $L(x, y, z)$ 表示“ x 学院的 y 课是 z 类别”, 假设用 a 表示计算机, b 表示人工智能, c 表示必修, 则上述命题可表示成 $L(a, b, c)$ 。

谓词

$P(x)$ 是一元谓词(包含一个个体), $P(x_1, x_2, \dots, x_n)$ 被称为n元谓词(包含若干个体)。

- 例:

$P(x)$ 表示: $x < x^2$

$father(x, y)$ 表示 x 是 y 的父亲

量词

- 全称量词 (universal quantifier, \forall)
 - 全称量词用符号 \forall 表示, 表示一切的、凡是、所有的、每一个等。
 $\forall x$ 表示论域中的所有个体, $\forall x P(x)$ 表示论域中的所有个体具有性质P
- 存在量词 (existential quantifier, \exists)
 - 存在量词用符号 \exists 表示, 表示存在、有一个、某些等。
 $\exists x$ 表示论域中存在一个或若干个个体, $\exists x P(x)$ 表示论域中存在一个个体或若干个个体具有性质P

量词-例

- 全称量词

南航的学生都是聪明的

- 个体域为南航的学生

$$\forall x \text{ smart}(x)$$

- 个体域为学生

$$\forall x (\text{at}(x, \text{NUAA}) \rightarrow \text{smart}(x))$$

- 个体域为人

$$\forall x (\text{student}(x) \wedge \text{at}(x, \text{NUAA}) \rightarrow \text{smart}(x))$$

全称量词通常与 \rightarrow 蕴含 (implication) 一起使用

量词-例

- 存在量词

南理工有聪明的学生

- 个体域是南理工的学生

$$\exists x \text{ smart}(x)$$

- 个体域是学生

$$\exists x (\text{at}(x, \text{NJUST}) \wedge \text{smart}(x))$$

存在量词通常与 \wedge 合取 (conjunction) 一起使用

量词

- 全称量词通常与蕴含一起使用

- **All $S(x)$ is $P(x)$**
 - $\forall x (S(x) \rightarrow P(x))$
- **No $S(x)$ is $P(x)$**
 - $\forall x (S(x) \rightarrow \neg P(x))$

- 存在量词通常与合取一起使用

- **Some $S(x)$ is $P(x)$**
 - $\exists x (S(x) \wedge P(x))$
- **Some $S(x)$ is not $P(x)$**
 - $\exists x (S(x) \wedge \neg P(x))$

嵌套量词

- 谓词逻辑中，有时只用一个量词不足以表达一个陈述句的含义。

例：

- 命题：** There is a person who loves everybody.
- 翻译：**
 - 用个体变元 x 和 y 表示人
 - 用谓词 $L(x, y)$ 表示 “ x loves y ”

那么用谓词逻辑应如何表示？

嵌套量词

- 谓词逻辑中，有时只用一个量词不足以表达一个陈述句的含义。

例：

- 命题：** There is a person who loves everybody.
- 翻译：**
 - 用个体变元 x 和 y 表示人
 - 用谓词 $L(x, y)$ 表示 “ x loves y ”

$$\exists x \forall y L(x, y)$$

翻译

- 令
 - 个体变元 x 和 y 表示人
 - 谓词 $L(x, y)$ 表示 “ x loves y ”
- 翻译
 - Everybody loves Raymond $\forall x L(x, Raymond)$
 - Everybody loves somebody $\forall x \exists y L(x, y)$
 - There is somebody whom everybody loves $\exists y \forall x L(x, y)$
 - There is somebody who Raymond doesn't love $\exists y \neg L(Raymond, y)$
 - There is somebody whom no one loves $\exists y \forall x \neg L(x, y)$

量词的顺序

- 同种量词的顺序改变不影响含义

For all x and y , if x is a parent of y then y is a child of x .

$$\forall x, y \text{ (parent}(x, y) \Rightarrow \text{child}(y, x))$$

$$\forall y, x \text{ (parent}(x, y) \Rightarrow \text{child}(y, x))$$

- 不同量词的顺序改变会影响含义

$$\forall x \exists y \text{ loves } (x, y)$$

Everybody loves somebody

$$\exists y \forall x \text{ loves } (x, y)$$

There is someone who is loved by everyone

量词的关联

Everyone likes ice cream

$\forall x \text{ likes } (x, \text{IceCream})$

- 能否用存在量词来表达同样的意思？

There is no one who does not like ice cream

$\neg \exists x \neg \text{likes } (x, \text{IceCream})$

- 用全称量词表示的句子也可以改为用存在量词表示

量词的关联

Someone likes ice cream

$\exists x \text{ likes } (x, \text{IceCream})$

- 能否用全称量词来表达同样的意思？

Not everyone does not like ice cream

$\neg \forall x \neg \text{likes } (x, \text{IceCream})$

- 用存在量词表示的句子也可以改为用全称量词表示

函词 (Function)

- 定义 **函词 (function)**，表示以论域为定义域和值域的一种映射。通常用小写的字母 f 、 g 、 h 等表示。

- 例：

小王拍了下小明的胳膊。

用 $L(x, y)$ 表示 x 拍了下 y 。 $f(x)$ 表示 x 的胳膊， a 表示小王， b 表示小明，那么这个命题就可以表示为 $L(a, f(b))$ 。

论域可以取包括小明、小王、小明的胳膊的任何集合，它们都是论域中的个体。 a ， b 是个体常元，而 $f(b)$ 比较特殊，它既不是个体常元，也不是个体变元，它使用了一个函词来表示论域上的一个映射。

函词 (Function)

- 函词：用个体（来自论域）代入后结果仍是个体
- 谓词：用个体带入后就变成了命题，如 $car(x)$ （x是车）这个谓词中x用吉普车代替，则 $car(\text{吉普车})$ 是命题
- 函词表示从论域到论域的映射；谓词表示从论域到 $\{True, False\}$ 的映射

项 (Term)

- 定义 **项 (Term)** :

- (1) 个体常元 a 和个体变元 x 是**项**。
- (2) 有函词 $f(a_1, a_2, \dots, a_n)$, 其中 a_1, a_2, \dots, a_n 都是**项**。则 $f(a_1, a_2, \dots, a_n)$ 也是**项**。
- (3) 任何项只能由 (1) (2) 产生。

这里定义的项将个体常元、个体变元和函词的概念统一了起来。
囊括了谓词逻辑中表示对象的方式。

自由变元和约束变元

- 给定一个谓词逻辑的公式（简称“谓词公式”） A ，其中有部分谓词公式形如 $(\forall x)P(x)$ 或 $(\exists x)P(x)$ ，则称其为 A 的 x 约束部分；称 $P(x)$ 为相应量词的辖域（scope）。
- 在辖域中，个体变元 x 的所有出现为约束出现，这种变元称为约束变元（bound variable）；不是约束出现的其他变元出现称为自由出现，这些个体变元称为自由变元（free variable）。

自由变元和约束变元

- 通常，一个量词的辖域是谓词公式 A 的一部分。确定一个量词的辖域即是找出位于该量词之后的相邻接的子公式，一般遵循如下规则。
 - (1) 若量词后有括号，则括号内的子公式就是该量词的辖域。
 - (2) 若量词后无括号，则与该量词邻接的子公式为该量词的辖域。
- 谓词公式 A 中的个体变元在 A 中是约束出现还是自由出现决定了它是约束变元还是自由变元。

自由变元和约束变元

$$(\forall x)(\forall y)(P(x, z) \leftrightarrow Q(x, y)) \vee (\exists x)L(x, y)$$

- $(\forall x)$ 和 $(\forall y)$ 的辖域为 $(P(x, z) \leftrightarrow Q(x, y))$, 其中 x 和 y 为约束出现, z 为自由出现。
- $(\exists x)$ 的辖域为 $L(x, y)$, 其中 x 为约束出现而 y 为自由出现。
 - 严格来说, $(\forall y)$ 只约束了公式的前面部分

如何理解自由变元?

一般, 我们只考虑没有自由变元的公式, 并把这样的公式称为句子。

解释 (Interpretation)

- 谓词公式 A 的一种解释 I
- 指定一个非空集合 D ，称为 I 的论域，并称 I 为 D 上的解释。
- 对 A 中出现的每个个体常元，都指定 D 中的一个元素。
- 对 A 中出现的每个函词，都指定 D 上的一个映射。
- 对 A 中出现的每个 n 元谓词，都指定一个从 D^n 到 $\{T, F\}$ 的映射。
- 这样一来就可以由谓词公式 A 得到 D 上的一个命题。

解释 (Interpretation) - 例

$king(John)$

$\forall x \text{ father}(John, x) \Rightarrow \text{prince}(x)$

$\forall x, y (\text{equal}(\text{father_of}(x), \text{father_of}(y)) \wedge \neg \text{equal}(x, y)) \Rightarrow \text{brother}(x, y)$

$D : \text{John}, \text{Bob}, \text{Aaron}, \dots$

一种解释:

$John \longrightarrow \text{John}$

$king:$

$John$	\longrightarrow	$True$
$Aaron$	\longrightarrow	$False$
Bob	\longrightarrow	$False$
\dots		

$brother:$

$Bob, Aaron$	\longrightarrow	$True$
$John, Aaron$	\longrightarrow	$False$
\dots		

$father_of:$

Bob	\longrightarrow	$John$
$Aaron$	\longrightarrow	$John$
\dots		

$father:$

$John, Bob$	\longrightarrow	$True$
$John, Aaron$	\longrightarrow	$True$
$Bob, Aaron$	\longrightarrow	$False$
\dots		

$equal:$

$Aaron, Aaron$	\longrightarrow	$True$
$John, Aaron$	\longrightarrow	$False$
\dots		

用谓词逻辑表示知识

用谓词逻辑表示知识

- 把一个文字叙述的命题，用谓词公式表示出来，称为谓词逻辑的翻译或符号化。一般来说，符号化的步骤如下
 - (1) 正确理解给定命题。必要时把命题改叙，使其能表达每个原子命题及原子命题之间的关系。
 - (2) 把每个原子命题分解成个体、谓词和量词。
 - (3) 找出适当量词。注意一般全称量词后跟蕴含式，存在量词后跟合取式。
 - (4) 用适当联结词把给定命题表示出来。

用谓词逻辑表示知识

- 例：家谱

Kinship domain

- **Objects:** people
John , Mary , Jane , ...
- **Properties:** gender
Male (x), Female (x)
- **Relations:** parenthood, brotherhood, marriage
Parent (x, y), Brother (x, y), Spouse (x, y)
- **Functions:** mother-of (one for each person x)
MotherOf (x)

用谓词逻辑表示知识

- 写下我们所知道的谓词和函词间的关系

- Male and female are disjoint categories

$$\forall x \text{ Male}(x) \Leftrightarrow \neg \text{Female}(x)$$

- Parent and child relations are inverse

$$\forall x, y \text{ Parent}(x, y) \Leftrightarrow \text{Child}(y, x)$$

- A grandparent is a parent of parent

$$\forall g, c \text{ Grandparent}(g, c) \Leftrightarrow \exists p \text{ Parent}(g, p) \wedge \text{Parent}(p, c)$$

- A sibling is another child of one's parents

$$\forall x, y \text{ Sibling}(x, y) \Leftrightarrow \neg \text{equal}(x, y) \wedge \exists p \text{ Parent}(p, x) \wedge \text{Parent}(p, y)$$

- 等等 ...

然后

- 再加入一些事实，可以进行推理

KB:

$Parent(John, Mary)$

$Parent(Mary, Jane)$

$\forall x, y \text{ Grandparent}(x, y) \Leftrightarrow \exists z \text{ Parent}(x, z) \wedge \text{Parent}(z, y)$

...

$\alpha:$ $Grandparent(John, Jane)$

谓词逻辑中的推理

谓词逻辑中的推理

给定：

- 一个KB (a set of sentences)
- 一个句子 α

问：KB是否语义蕴含 α ? $KB \models \alpha$?

即：所有令KB为真的解释是否也令 α 为真

Logical inference problem in the first-order logic is semidecidable.

Algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non-entailed sentence.

求解逻辑推理问题

- 如何设计流程来回答

$$KB \models \alpha \text{ ?}$$

- 方法：
 - 真值表法 (Truth-table approach)
 - 推理规则法 (Inference rules)

真值表法

- 真值表法在谓词逻辑中可行吗?
- NO!
- 为什么

推理规则法

- 推理规则法对谓词逻辑是否可行？
- 可行
- 推理规则是可应用于KB中句子的可靠推理模式
- What is derived follows from the KB
- 但是
 - 需要增加规则来处理量词

推理规则

- 主要考虑两条规则
 - 肯定前件 (Modus ponens)
 - 归结 (Resolution)

$$\frac{A \Rightarrow B, \quad A}{B}$$

$$\frac{A \vee B, \quad \neg B \vee C}{A \vee C}$$

- 由于有量词，需要进行变元替换

变元替换

- 可以把句子中的变元用项 (term) 替换
 - 项：常元、变元、函词

- 替换：
 - 即将变元映射到项

- 例：

$$SUBST(\{x/Sam, y/Pam\}, Likes(x, y)) = Likes(Sam, Pam)$$

$$SUBST(\{x/z, y/fatherof(John)\}, Likes(x, y)) = Likes(z, fatherof(John))$$

量词的推理规则

- 通过替换消除量词后推理
- 全称量词消除
 - 可考虑将变量替换为一个个体常元符号

$$\frac{\forall x \phi(x)}{\phi(a)}$$

其中a是个体常元符号

- 例:

$\forall x Likes(x, IceCream)$



$Likes(Ben, IceCream)$

量词的推理规则

- 存在量词消除
 - 将变量替换为一个在KB中其他地方没有用过的个体常元符号

$$\frac{\exists x \phi(x)}{\phi(a)}$$

- 例：

这里的k是一个特殊的常元，称为Skolem常元

$$\exists x \text{ Kill}(x, \text{Victim}) \longrightarrow \text{Kill}(k, \text{Victim})$$

$$\exists x (\text{Crown}(x) \wedge \text{OnHead}(x, \text{John})) \longrightarrow \text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$$

转化为命题推理

KB: $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
King(John)
Greedy(John)
Brother(Richard, John)

- 用所有可能的个体常元替换全称量词变元:

King(John) \wedge Greedy(John) \Rightarrow Evil(John)
King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)
King(John)
Greedy(John)
Brother(Richard, John)

这样, KB就被命题化 (propositionalized), 命题符号为:

King(John), Greedy(John), Evil(John), King(Richard)等

命题化的问题

- 命题化可能产生很多不相关或意义不大的句子

- 例如, 根据

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$

$\text{King}(\text{John})$

$\forall y \text{ Greedy}(y)$

$\text{Brother}(\text{Richard}, \text{John})$

- $\text{Evil}(\text{John})$ 应是明显被蕴含的, 但命题化产生了很多像 $\text{Greedy}(\text{Richard})$ 这样的不相关句子
- 对全称量词, 如果有 p 个 k 元谓词, n 个常元, 就有 $p \cdot n^k$ 种可能
- 如果有函词, 甚至可以产生无限多命题

Unification (合一)

- 尝试可能有助于使用规则进行推理的替换
 - 考虑KB中相似的句子

例: $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
 $\text{King}(\text{John})$
 $\forall y \text{ Greedy}(y)$

如果使用替换 $\sigma = \{x/\text{John}, y/\text{John}\}$
就可以用肯定前件一步推导出 $\text{Evil}(\text{John})$

Unification (合一)

- Unification考虑两个句子，通过替换使它们相同

$$UNIFY(p, q) = \sigma \text{ s.t. } SUBST(\sigma, p) = SUBST(\sigma, q)$$

例:

$$UNIFY(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$$

$$UNIFY(Knows(John, x), Knows(y, Ann)) = \{x/Ann, y/John\}$$

$$\begin{aligned} UNIFY(Knows(John, x), Knows(y, MotherOf(y))) \\ = \{x/MotherOf(John), y/John\} \end{aligned}$$

$$UNIFY(Knows(John, x), Knows(x, Elizabeth)) \text{ fail}$$

Unification (合一)

- 如何Unify $Knows(John, x)$ 和 $Knows(y, z)$?

$\sigma = \{y/John, x/z\}$ 还是 $\sigma = \{y/John, x/John, z/John\}$

- 前一种比后一种更一般，后一种可以在前一种执行后再进行一次替换得到
- 对每对可合一的句子，都存在一个most general unifier (MGU)

$$MGU = \{ y/John, x/z \}$$

Unification (合一)算法

```
function UNIFY( $x, y, \theta$ ) returns a substitution to make  $x$  and  $y$  identical
  inputs:  $x$ , a variable, constant, list, or compound
            $y$ , a variable, constant, list, or compound
            $\theta$ , the substitution built up so far (optional, defaults to empty)

  if  $\theta = \text{failure}$  then return failure
  else if  $x = y$  then return  $\theta$ 
  else if VARIABLE?( $x$ ) then return UNIFY-VAR( $x, y, \theta$ )
  else if VARIABLE?( $y$ ) then return UNIFY-VAR( $y, x, \theta$ )
  else if COMPOUND?( $x$ ) and COMPOUND?( $y$ ) then
    return UNIFY(ARGS[ $x$ ], ARGS[ $y$ ], UNIFY(OP[ $x$ ], OP[ $y$ ],  $\theta$ ))
  else if LIST?( $x$ ) and LIST?( $y$ ) then
    return UNIFY(REST[ $x$ ], REST[ $y$ ], UNIFY(FIRST[ $x$ ], FIRST[ $y$ ],  $\theta$ ))
  else return failure
```

Unification (合一)算法

function UNIFY-VAR(var, x, θ) **returns** a substitution

inputs: var , a variable

x , any expression

θ , the substitution built up so far

if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)

else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)

else if OCCUR-CHECK?(var, x) **then return** failure

else return add $\{var/x\}$ to θ

一般化的推理规则

- 可基于合一（Unification）得到更一般的推理规则

$$\frac{A \vee B, \quad \neg A \vee C}{B \vee C} \quad \Rightarrow \quad \frac{\sigma = \text{UNIFY}(\phi_i, \neg \psi_j) \neq \text{fail} \quad \phi_1 \vee \phi_2 \dots \vee \phi_k, \quad \psi_1 \vee \psi_2 \vee \dots \vee \psi_n}{\text{SUBST}(\sigma, \phi_1 \vee \dots \vee \phi_{i-1} \vee \phi_{i+1} \dots \vee \phi_k \vee \psi_1 \vee \dots \vee \psi_{j-1} \vee \psi_{j+1} \dots \vee \psi_n)}$$

- 以上为广义归结（Generalized Resolution）
- 使用一般化的推理规则的优点：
 - 只进行有利于推理的替换
 - 无需将句子命题化

广义归结 (Generalized Resolution)

$$\frac{\sigma = UNIFY(\phi_i, \neg \psi_j) \neq fail \quad \phi_1 \vee \phi_2 \dots \vee \phi_k, \quad \psi_1 \vee \psi_2 \vee \dots \vee \psi_n}{SUBST(\sigma, \phi_1 \vee \dots \vee \phi_{i-1} \vee \phi_{i+1} \dots \vee \phi_k \vee \psi_1 \vee \dots \vee \psi_{j-1} \vee \psi_{j+1} \dots \vee \psi_n)}$$

例:

$$\frac{P(x) \vee Q(x), \quad \neg Q(John) \vee S(y)}{P(John) \vee S(y)}$$

其中 $\sigma = UNIFY(Q(x), \neg(\neg Q(John))) = \{x/John\}$

广义归结 (Generalized Resolution)

$$\frac{\sigma = UNIFY(\phi_i, \neg \psi_j) \neq fail \quad \phi_1 \vee \phi_2 \dots \vee \phi_k, \quad \psi_1 \vee \psi_2 \vee \dots \vee \psi_n}{SUBST(\sigma, \phi_1 \vee \dots \vee \phi_{i-1} \vee \phi_{i+1} \dots \vee \phi_k \vee \psi_1 \vee \dots \vee \psi_{j-1} \vee \psi_{j+1} \dots \vee \psi_n)}$$

- 广义归结规则对谓词逻辑是可靠且完备的

使用归结推理

使用归结推理

- Proof by refutation
 - 证明 $KB, \neg\alpha$ 不可满足
- 主要流程
 - 将 $KB, \neg\alpha$ 转化, 使其只带基本项和“全称”变元 (原本被全称量词约束)
 - 基本项 (ground term) : 不带变元的项
 - 重复使用归结
 - 如得到矛盾则停止 (CNF 不可满足, $KB \models \alpha$)

注: 如 $KB \models \alpha$ 不成立, 过程可能无法终止

转化为CNF

1. 消除蕴含 (implication)

$$p \Rightarrow q \longrightarrow \neg p \vee q$$

2. 移入否定符号

$$\neg(p \wedge q) \longrightarrow \neg p \vee \neg q$$

$$\neg(p \vee q) \longrightarrow \neg p \wedge \neg q$$

$$\neg\forall x p \longrightarrow \exists x \neg p$$

$$\neg\exists x p \longrightarrow \forall x \neg p$$

$$\neg\neg p \longrightarrow p$$

3. 将重名的变元重新命名

$$(\forall x P(x)) \vee (\exists x Q(x)) \longrightarrow (\forall x P(x)) \vee (\exists y Q(y))$$

转化为CNF

4. 将量词移到左边

$$(\forall x P(x)) \vee (\exists y Q(y)) \longrightarrow \forall x \exists y (P(x) \vee Q(y))$$

5. 去除存在量词

- 如果前面没有全称量词，则将变元用一个新的常元替换

$$\exists y (P(A) \vee Q(y)) \longrightarrow P(A) \vee Q(B)$$

- 如果前面有全称量词，则用一个“全称”变元的函词替换

$$\forall x \exists y (P(x) \vee Q(y)) \longrightarrow \forall x (P(x) \vee Q(f(x)))$$

将该函词 $f(x)$ 称为Skolem function

转化为CNF

6. (直接) 去掉全称量词

$$\forall x (P(x) \vee Q(f(x))) \longrightarrow P(x) \vee Q(f(x))$$

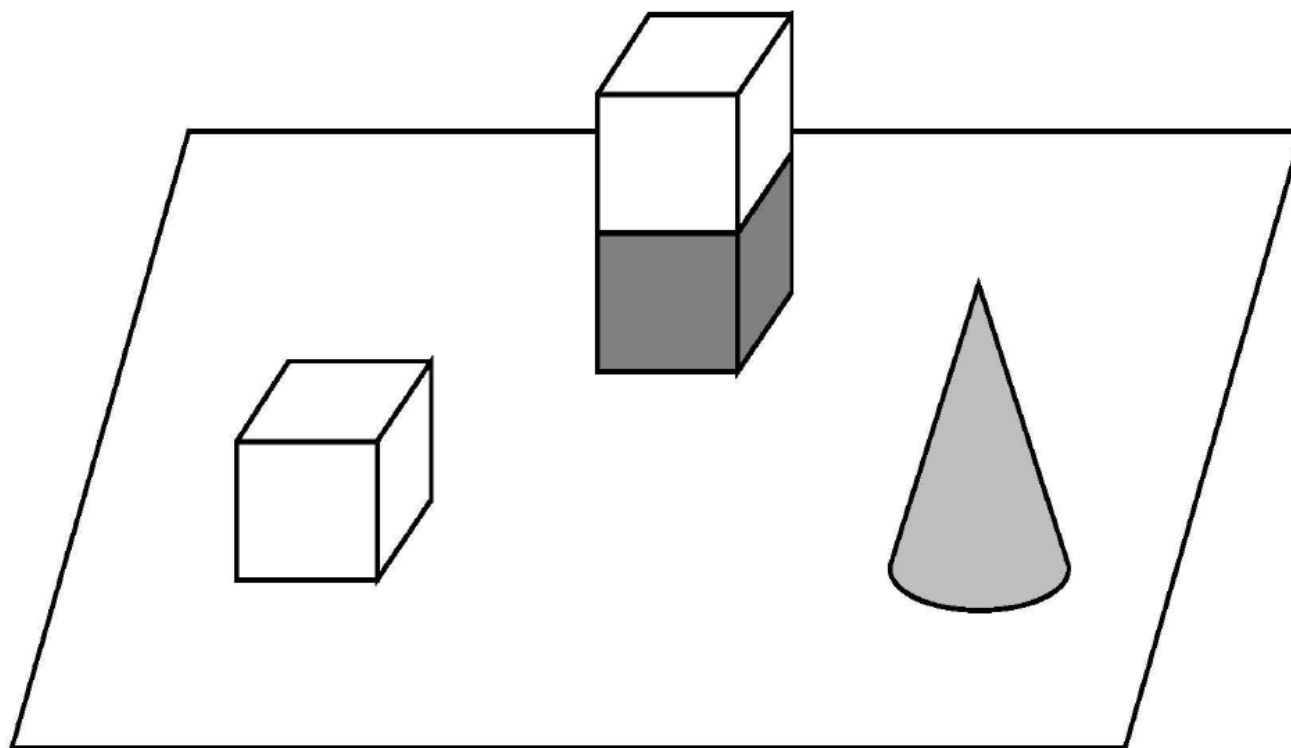
7. 使用分配律转化为CNF

$$p \vee (q \wedge r) \longrightarrow (p \vee q) \wedge (p \vee r)$$

最后得到一个带变元、常元和函词的CNF

转化为CNF-例

Blocksworld



转化为CNF-例

句子:

$$\begin{aligned}\forall x : brick(x) \rightarrow & ((\exists y : on(x, y) \wedge \neg pyramid(y)) \\ & \wedge (\neg \exists y : on(x, y) \wedge on(y, x)) \\ & \wedge (\forall y : \neg brick(y) \rightarrow \neg equal(x, y)))\end{aligned}$$

First, bricks are on something else that is not a pyramid;

Second, there is nothing that a brick is on and that is on the brick as well.

Third, there is nothing that is not a brick and also is the same thing as the brick.

CNF:

$$\neg brick(x) \vee on(x, support(x))$$

$$\neg brick(w) \vee \neg pyramid(support(w))$$

$$\neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$$

$$\neg brick(v) \vee brick(z) \vee \neg equal(v, z)$$

转化为CNF-例

- 1. 消除蕴含 (implication)

$$\begin{aligned}\forall x : brick(x) \rightarrow & ((\exists y : on(x, y) \wedge \neg pyramid(y)) \\ & \wedge (\neg \exists y : on(x, y) \wedge on(y, x)) \\ & \wedge (\forall y : \neg brick(y) \rightarrow \neg equal(x, y)))\end{aligned}$$

$$\begin{aligned}\forall x : \neg brick(x) \vee & ((\exists y : on(x, y) \wedge \neg pyramid(y)) \\ & \wedge (\neg \exists y : on(x, y) \wedge on(y, x)) \\ & \wedge (\forall y : \neg(\neg brick(y)) \vee \neg equal(x, y)))\end{aligned}$$

转化为CNF-例

• 2. 移入否定符号

$$\begin{aligned} \forall x : \neg brick(x) \vee & ((\exists y : on(x, y) \wedge \neg pyramid(y)) \\ & \wedge (\neg \exists y : on(x, y) \wedge on(y, x)) \\ & \wedge (\forall y : \neg(\neg brick(y)) \vee \neg equal(x, y)) \end{aligned}$$

$$\begin{aligned} \forall x : \neg brick(x) \vee & ((\exists y : on(x, y) \wedge \neg pyramid(y)) \\ & \wedge (\forall y : (\neg on(x, y) \vee \neg on(y, x))) \\ & \wedge (\forall y : (brick(y) \vee \neg equal(x, y))) \end{aligned}$$

$\neg(E_1 \wedge E_2) \iff (\neg E_1) \vee (\neg E_2)$
$\neg(E_1 \vee E_2) \iff (\neg E_1) \wedge (\neg E_2)$
$\neg(\neg E_1) \iff E_1$
$\neg \forall x : E_1(x) \iff \exists x : \neg E_1(x)$
$\neg \exists x : E_1(x) \iff \forall x : \neg E_1(x)$

转化为CNF--例

• 3. 消除存在量词

$$\begin{aligned} & \forall x : \neg brick(x) \vee \\ & ((\exists y : on(x, y) \wedge \neg pyramid(y)) \\ & \wedge (\forall y : (\neg on(x, y) \vee \neg on(y, x))) \\ & \wedge (\forall y : (brick(y) \vee \neg equal(x, y))) \end{aligned}$$

- 如果前面没有全称量词，则将变元用一个新的常元替换
- 如果前面有全称量词，则用一个“全称”变元的函词替换

$$\begin{aligned} \forall x : \neg brick(x) \vee & (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\forall y : (\neg on(x, y) \vee \neg on(y, x))) \\ & \wedge (\forall y : (brick(y) \vee \neg equal(x, y))) \end{aligned}$$

转化为CNF-例

- 4. 将重名的变元重新命名 (Standardize variables)

$$\begin{aligned} \forall x : \neg brick(x) \vee & (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\forall y : (\neg on(x, y) \vee \neg on(y, x))) \\ & \wedge (\forall y : (brick(y) \vee \neg equal(x, y))) \end{aligned}$$

$$\begin{aligned} \forall x : \neg brick(x) \vee & (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\forall y : (\neg on(x, y) \vee \neg on(y, x))) \\ & \wedge (\forall z : (brick(z) \vee \neg equal(x, z))) \end{aligned}$$

转化为CNF-例

- 5. 将量词移至左边

$$\begin{aligned}\forall x : & \neg brick(x) \vee (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\forall y : (\neg on(x, y) \vee \neg on(y, x))) \\ & \wedge (\forall z : (brick(z) \vee \neg equal(x, z)))\end{aligned}$$

$$\begin{aligned}\forall x \forall y \forall z : & \neg brick(x) \vee (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\neg on(x, y) \vee \neg on(y, x)) \\ & \wedge (brick(z) \vee \neg equal(x, z))\end{aligned}$$

转化为CNF-例

- 6. 去除全称量词

$$\begin{aligned} \forall x \forall y \forall z : & \neg brick(x) \vee (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\neg on(x, y) \vee \neg on(y, x)) \\ & \wedge (brick(z) \vee \neg equal(x, z)) \end{aligned}$$

$$\begin{aligned} & \neg brick(x) \vee (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\neg on(x, y) \vee \neg on(y, x)) \\ & \wedge (brick(z) \vee \neg equal(x, z)) \end{aligned}$$

转化为CNF-例

- 7. 使用分配律转为CNF

$$\begin{aligned} &\neg brick(x) \vee (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ &\quad \wedge (\neg on(x, y) \vee \neg on(y, x)) \\ &\quad \wedge (brick(z) \vee \neg equal(x, z)) \end{aligned}$$

$$\begin{aligned} &\neg brick(x) \vee on(x, S1(x)) \\ &\neg brick(x) \vee \neg pyramid(S1(x)) \\ &\neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x) \\ &\neg brick(x) \vee brick(z) \vee \neg equal(x, z) \end{aligned}$$

归结—例

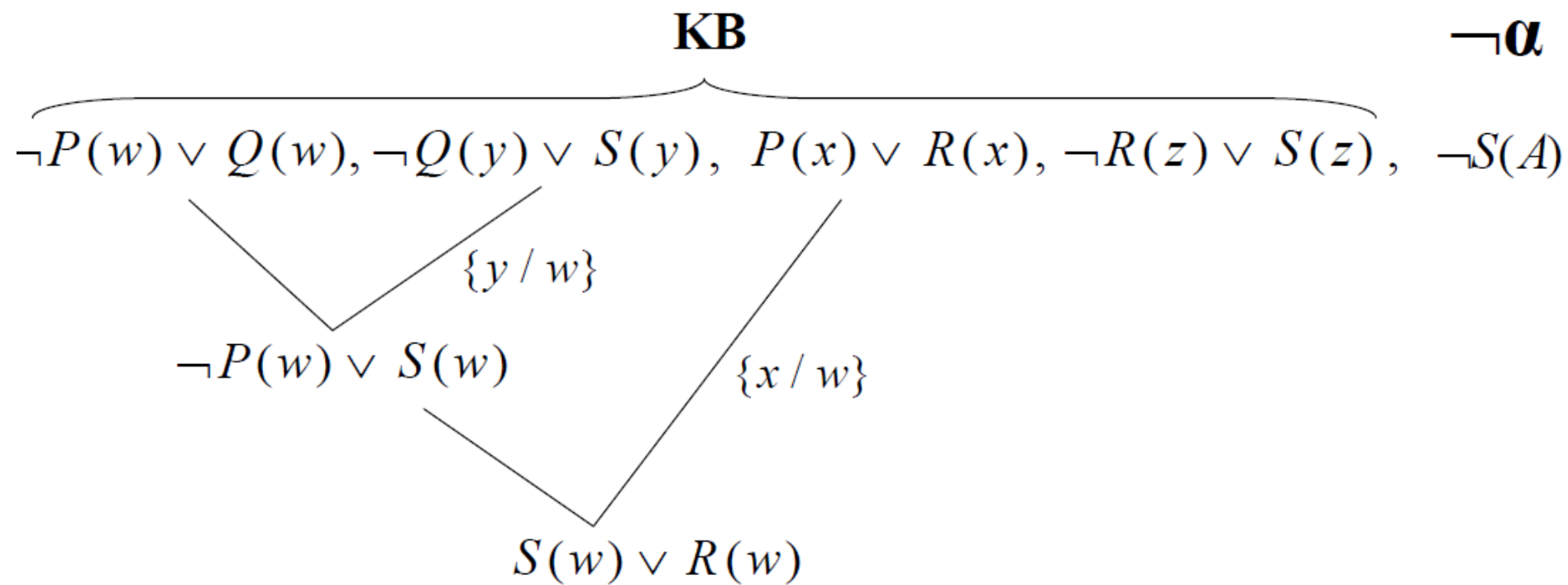
$$\begin{array}{ccc} & \text{KB} & \neg \alpha \\ \hline \overbrace{\neg P(w) \vee Q(w), \neg Q(y) \vee S(y), P(x) \vee R(x), \neg R(z) \vee S(z)} & & \neg S(A) \end{array}$$

其中 w, x, y, z 都是变元

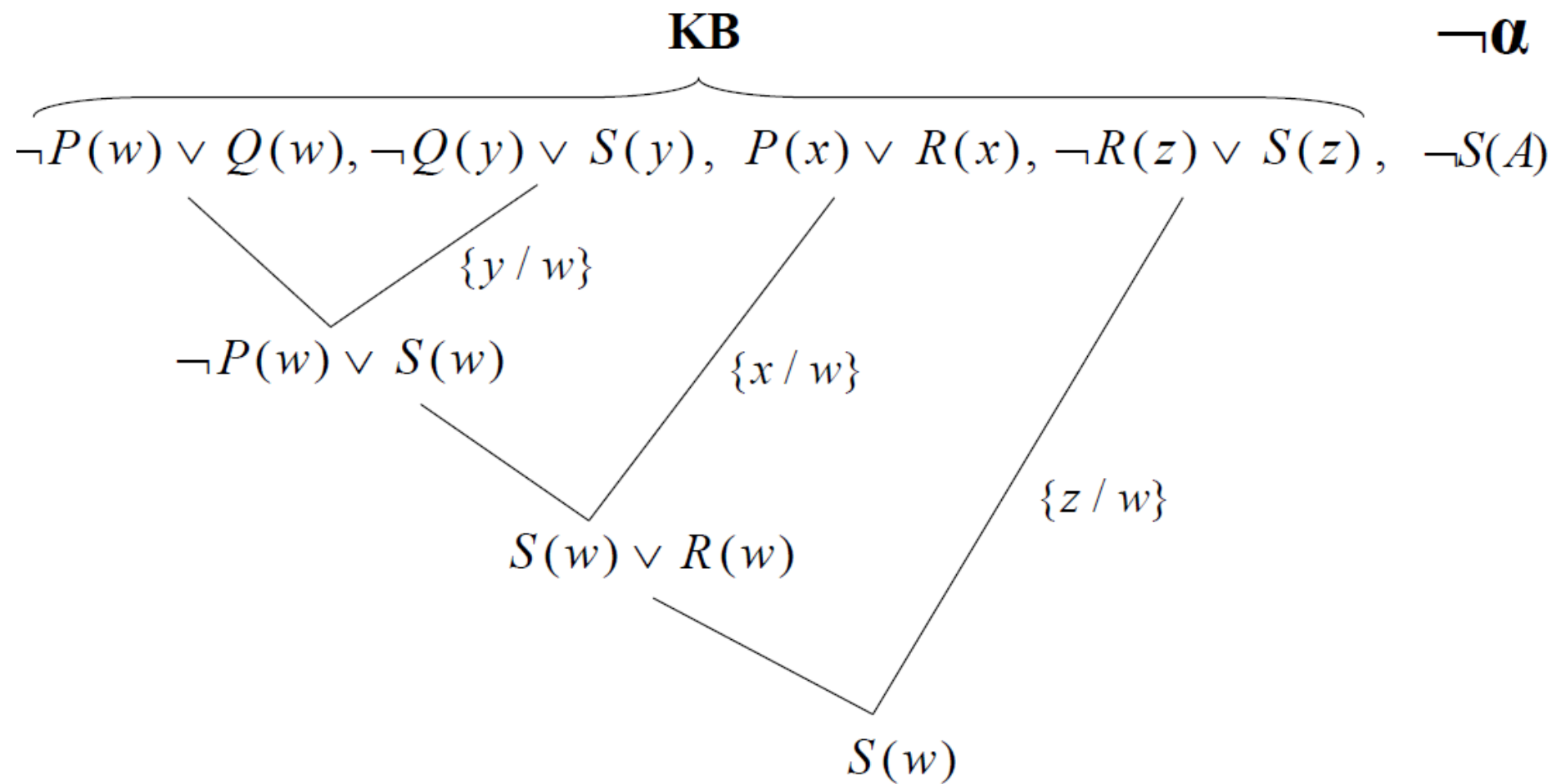
归结—例

$$\begin{array}{c} \text{KB} \qquad \qquad \qquad \neg \alpha \\ \hline \neg P(w) \vee Q(w), \neg Q(y) \vee S(y), P(x) \vee R(x), \neg R(z) \vee S(z), \neg S(A) \\ \swarrow \quad \searrow \{y/w\} \\ \neg P(w) \vee S(w) \end{array}$$

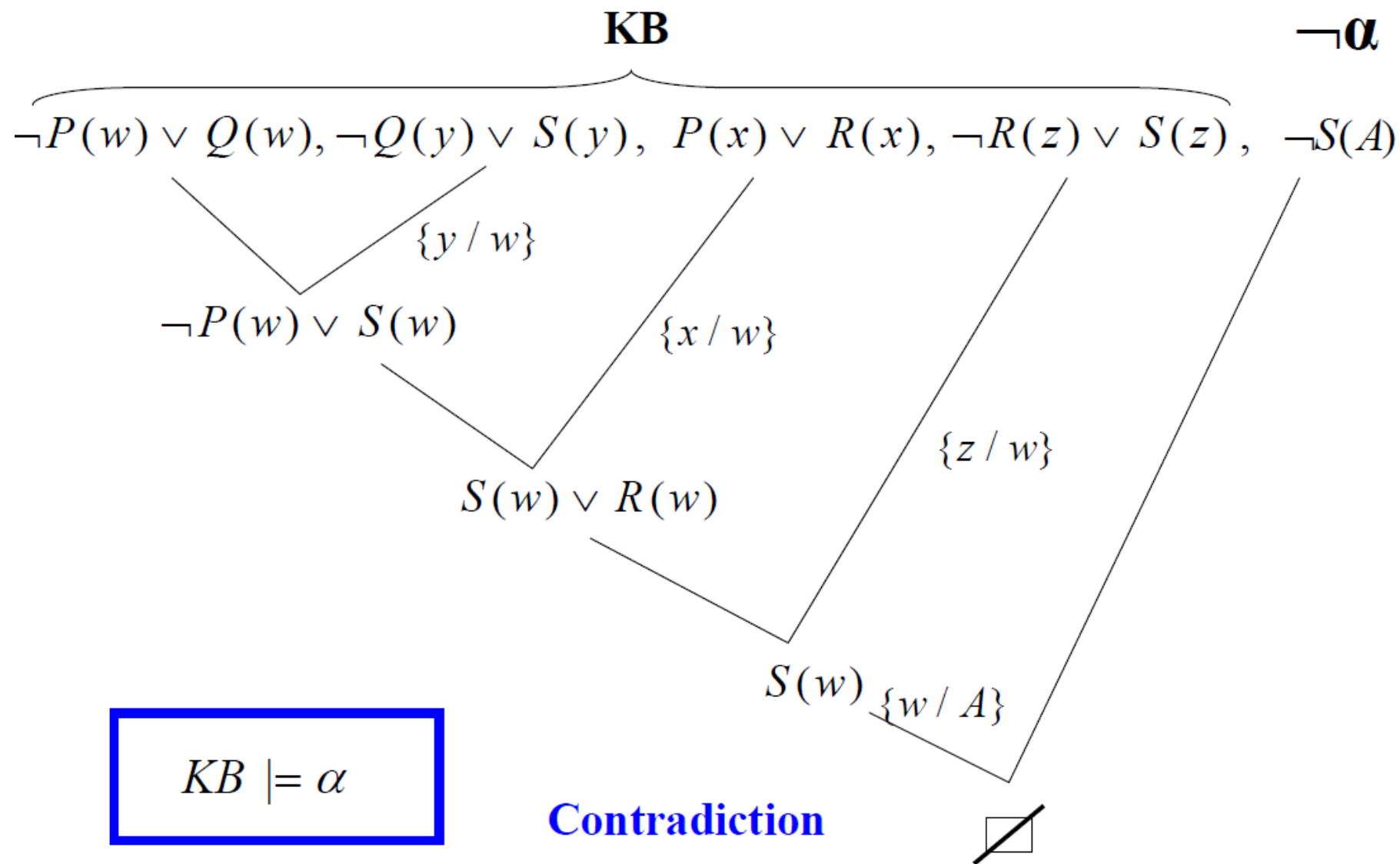
归结—例



归结—例



归结—例



无法停止的例子

假设有两个句子：

$$\begin{array}{l} 1. \quad q(y) \vee \neg q(g(y)) \\ 2. \quad \neg q(x) \vee \neg p(x) \end{array}$$

推理：

$$\begin{array}{ll} 3. \quad \neg q(g(x)) \vee \neg p(x) & [1, 2, \{y \mapsto x\}] \\ 4. \quad \neg q(g(g(x))) \vee \neg p(x) & [1, 3, \{y \mapsto g(x)\}] \\ 5. \quad \neg q(g(g(g(x)))) \vee \neg p(x) & [1, 4, \{y \mapsto g(g(x))\}] \\ \dots & \end{array}$$

例

已知：

- $(\forall x)(P(x) \rightarrow Q(x))$
- $(\forall x)(Q(x) \rightarrow R(x))$

试证明： $(\forall x)(P(x) \rightarrow R(x))$

练习

• 翻译

已知： 1) 每架飞机或者停在地面或者飞在天空;
2) 并非每架飞机都飞在天空

证明： 有些飞机停在地面

形式化： $plane(x)$: x 是飞机; $in_ground(x)$: x 停在地面; $on_fly(x)$: x 飞在天空

已知： $\forall x (plane(x) \Rightarrow in_ground(x) \vee on_fly(x))$
 $\neg \forall x (plane(x) \Rightarrow on_fly(x))$

注：可加上 $\neg(in_ground(x) \wedge on_fly(x))$ 更准确地表示语义，此处省略

$\forall x (plane(x) \Rightarrow ((in_ground(x) \vee on_fly(x)) \wedge \neg(in_ground(x) \wedge on_fly(x))))$

证明： $\exists x (plane(x) \wedge in_ground(x))$

例

1. $(\neg \forall x) \text{Plane}(x) \rightarrow \text{on_fly}(x)$
2. $(\exists x \neg (\text{Plane}(x) \rightarrow \text{on_fly}(x)))$
3. $(\exists x \neg (\neg \text{Plane}(x) \vee \text{on_fly}(x)))$
4. $(\exists x (\text{Plane}(x) \wedge \neg \text{on_fly}(x)))$
5. $\text{Plane}(a) \wedge \neg \text{on_fly}(a)$
6. $\text{Plane}(a)$
7. $\neg \text{on_fly}(a)$
8. $(\forall x) (\text{plane}(x) \rightarrow \text{in_ground}(x) \vee \text{on_fly}(x))$
9. $\text{plane}(a) \rightarrow \text{in_ground}(a) \vee \text{on_fly}(a)$
10. $\text{in_ground}(a) \vee \text{on_fly}(a)$
11. $\text{in_ground}(a)$
12. $\text{Plane}(a) \wedge \text{in_ground}(a)$
13. $(\exists x) (\text{plane}(x) \wedge \text{in_ground}(x))$

Try ChatGPT

HL **You**

Given "A plane either lands on the ground or flies in the sky. Not every plane flies in the sky." Use first order logic to prove that "some planes land on the ground."



You

Given "A plane either lands on the ground or flies in the sky. Not every plane flies in the sky." Use first order logic to prove that "some planes land on the ground."

ChatGPT

To prove "some planes land on the ground" using first-order logic given the premises "A plane either lands on the ground or flies in the sky" and "Not every plane flies in the sky," we can use existential instantiation and the law of excluded middle.

Let's denote the following predicates:

- $L(x)$ represents "x lands on the ground."
- $F(x)$ represents "x flies in the sky."
- $P(x)$ represents "x is a plane."

Given the premises:

1. $\forall x(P(x) \rightarrow (L(x) \vee F(x)))$ - A plane either lands on the ground or flies in the sky.
2. $\neg \forall x(P(x) \rightarrow F(x))$ - Not every plane flies in the sky.

We need to prove:

$\exists x(P(x) \wedge L(x))$ - Some planes land on the ground.

Given the premises:

1. $\forall x(P(x) \rightarrow (L(x) \vee F(x)))$ - A plane either lands on the ground or flies in the sky.
2. $\neg \forall x(P(x) \rightarrow F(x))$ - Not every plane flies in the sky.

We need to prove:

$\exists x(P(x) \wedge L(x))$ - Some planes land on the ground.

Proof:

1. $\neg \forall x(P(x) \rightarrow F(x))$ - Given (from premise 2).
2. $\exists x \neg(P(x) \rightarrow F(x))$ - Negation of universal quantifier.
3. $\exists x(P(x) \wedge \neg F(x))$ - Implication law.
4. Let y be a plane such that $P(y) \wedge \neg F(y)$.
5. $P(y) \rightarrow (L(y) \vee F(y))$ - Universal instantiation from premise 1.
6. $\neg F(y)$ - From step 4.
7. $L(y) \vee F(y)$ - Modus Ponens from steps 5 and 6.
8. $L(y)$ - Disjunctive syllogism from steps 7 and 6.
9. $P(y) \wedge L(y)$ - Conjunction of steps 4 and 8.

Therefore, we have proved that "some planes land on the ground" using first-order logic.

广义肯定前件 (GMP)

$$\frac{A_1 \wedge A_2 \wedge \dots A_n \Rightarrow B, \quad A_1', A_2', \dots A_n'}{SUBST(\sigma, B)}$$

其中对所有 $i = 1, \dots, n$, $SUBST(\sigma, A_i) = SUBST(\sigma, A_i')$

例:

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
 $\text{King}(\text{John})$
 $\forall y \text{ Greedy}(y)$
 $\text{Brother}(\text{Richard}, \text{John})$

$A_1 : \text{King}(x)$ $A_1' : \text{King}(\text{John})$ $B : \text{Evil}(x)$
 $A_2 : \text{Greedy}(x)$ $A_2' : \text{Greedy}(y)$

$\sigma = \{x/\text{John}, y/\text{John}\}$

$SUBST(\sigma, B) = \text{Evil}(\text{John})$

广义肯定前件 (GMP)

$$\frac{A_1 \wedge A_2 \wedge \dots A_n \Rightarrow B, \quad A_1', A_2', \dots A_n'}{SUBST (\sigma, B)}$$

其中对所有 $i = 1, \dots, n$, $SUBST (\sigma, A_i) = SUBST (\sigma, A_i')$

- GMP可用于 (在去掉量词后) 以Horn clause表示的KB

Forward Chaining

```
function FOL-FC-ASK( $KB, \alpha$ ) returns a substitution or false
  inputs:  $KB$ , the knowledge base, a set of first-order definite clauses
            $\alpha$ , the query, an atomic sentence
  local variables:  $new$ , the new sentences inferred on each iteration

  repeat until  $new$  is empty
     $new \leftarrow \{ \}$ 
    for each sentence  $r$  in  $KB$  do
       $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-APART}(r)$ 
      for each  $\theta$  such that  $\text{SUBST}(\theta, p_1 \wedge \dots \wedge p_n) = \text{SUBST}(\theta, p'_1 \wedge \dots \wedge p'_n)$ 
        for some  $p'_1, \dots, p'_n$  in  $KB$ 
           $q' \leftarrow \text{SUBST}(\theta, q)$ 
          if  $q'$  is not a renaming of some sentence already in  $KB$  or  $new$  then do
            add  $q'$  to  $new$ 
             $\phi \leftarrow \text{UNIFY}(q', \alpha)$ 
            if  $\phi$  is not fail then return  $\phi$ 
    add  $new$  to  $KB$ 
  return false
```

Forward Chaining

- 可以达到一个不动点 (fixed point)
 - 能推导出的句子都已被推导出
- 对用Horn clause表示的KB是可靠且完备的
 - Horn clause: clauses that have at most one positive literal
 - 完备: 对推理positive literal
- 算法效率仍可提升

Forward Chaining – 例

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

请证明Colonel West是罪犯

Forward Chaining – 例

“It is a crime for an American to sell weapons to hostile nations”

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

“The country Nono is an enemy of America”

$$Enemy(Nono, America)$$

“Nono has some missiles”

$$\exists x \text{ Owns}(\text{Nono}, x) \wedge \text{Missile}(x) \quad \Rightarrow \quad \text{Owns}(\text{Nono}, M_1) \wedge \text{Missile}(M_1)$$

“All of its missiles were sold to it by Colonel West”

$$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$$

“West is an American”

$$American(\text{West})$$

Missiles are weapons

$$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

An enemy of America counts as “hostile”

$$Enemy(x, America) \Rightarrow \text{Hostile}(x)$$

Forward Chaining – 例

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

$$Missile(x) \Rightarrow Weapon(x)$$

$$Enemy(x, America) \Rightarrow Hostile(x)$$

$American(West)$

$Missile(M_1)$

$Owns(Nono, M_1)$

$Enemy(Nono, America)$

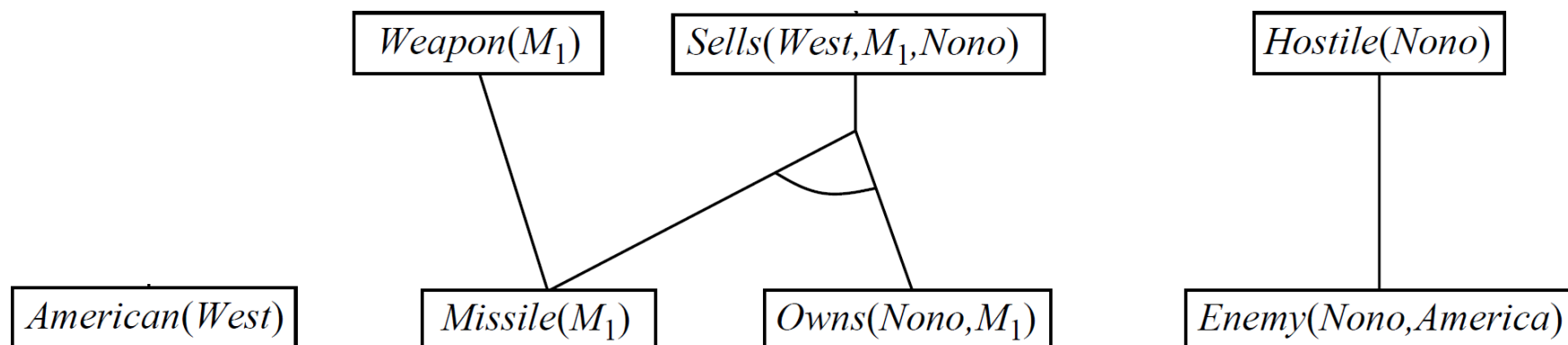
Forward Chaining – 例

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

➡ $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

➡ $Missile(x) \Rightarrow Weapon(x)$

➡ $Enemy(x, America) \Rightarrow Hostile(x)$



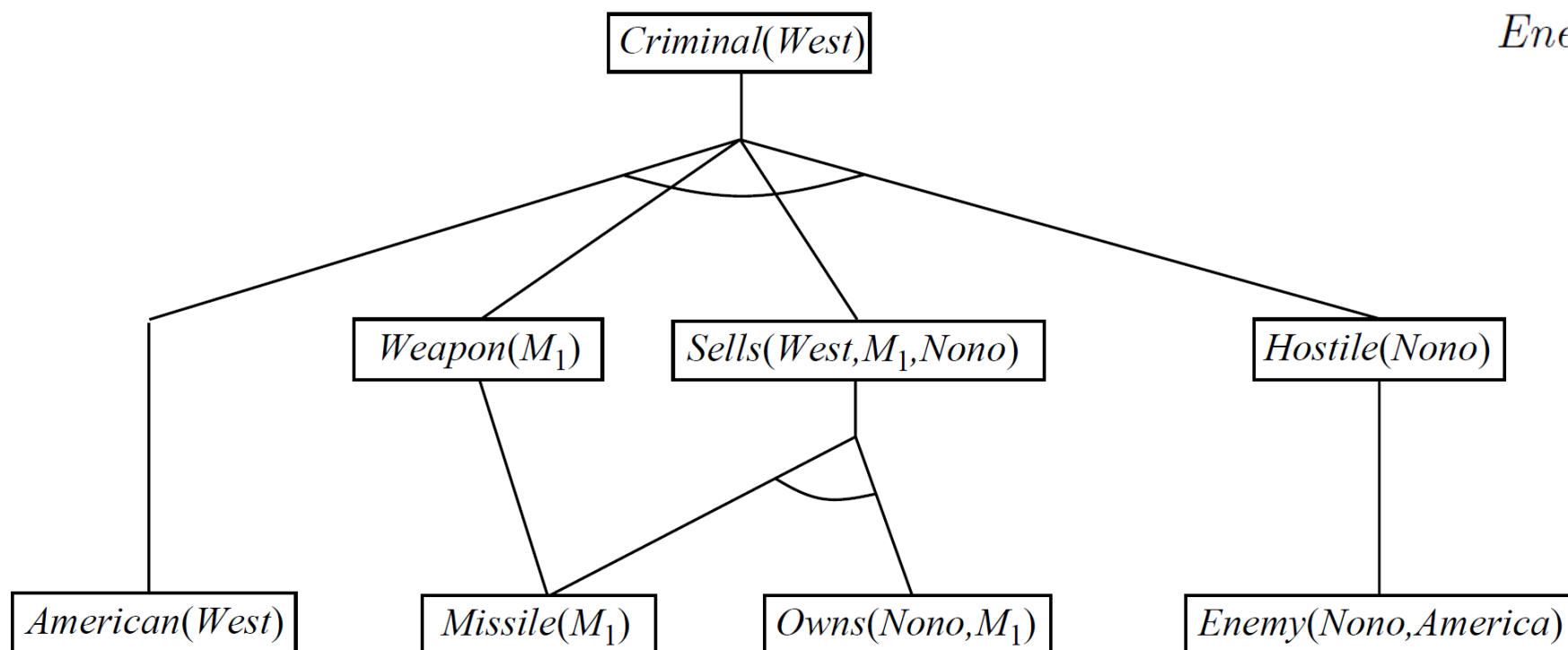
Forward Chaining – 例

➡ $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$



Backward Chaining

- 从目标开始，反向根据规则找证据证明它，递归地尝试证明肯定前件的前提

```
function FOL-BC-ASK(KB, goals,  $\theta$ ) returns a set of substitutions
  inputs: KB, a knowledge base
           goals, a list of conjuncts forming a query ( $\theta$  already applied)
            $\theta$ , the current substitution, initially the empty substitution  $\{ \}$ 
  local variables: answers, a set of substitutions, initially empty

  if goals is empty then return  $\{ \theta \}$ 
   $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(\text{goals}))$ 
  for each sentence r in KB where  $\text{STANDARDIZE-APART}(r) = (p_1 \wedge \dots \wedge p_n \Rightarrow q)$ 
    and  $\theta' \leftarrow \text{UNIFY}(q, q')$  succeeds
     $\text{new\_goals} \leftarrow [p_1, \dots, p_n | \text{REST}(\text{goals})]$ 
     $\text{answers} \leftarrow \text{FOL-BC-ASK}(\text{KB}, \text{new\_goals}, \text{COMPOSE}(\theta', \theta)) \cup \text{answers}$ 
  return answers
```

Coding

- 使用NLTK和prover9进行谓词逻辑推理
- NLTK自然语言处理库，是最受欢迎的自然语言处理工具包之一，可用于分词、词性标注、命名实体识别（NER）及句法分析等。本实验中将使用其LogicParser类将字符串转化为计算机可以理解的逻辑解析式。
- Prover9是一个成熟的逻辑推理工具软件，在NLTK中内置了调用Prover9的接口，可以调用其对输入逻辑解析式进行验证。

过程

- 三个命题：
 - P: All men are mortal.
 - Q: Socrates is a man.
 - R: Socrates is mortal.
- 将三个命题输入计算机；将P和Q作为前提，看R是否成立

环境配置

- 安装NLTK库
 - <https://www.nltk.org/>
 - 或直接: `pip install nltk`
- 下载Prover9并编译
 - <https://www.cs.unm.edu/~mccune/prover9/>
 - 配置PROVER9环境变量为 "安装地址/bin"

可能的编译错误: `search.c:(.text+0x670b): undefined reference to `round'`

解决办法:

```
gcc -O -Wall -o prover9 prover9.o index_lits.o forward_subsume.o demodulate.o pred_elim.o unfold.o  
semantics.o giv_select.o white_black.o actions.o search.o utilities.o provers.o foffer.o ../ladr/libladr.a -lm  
执行得到prover9放到 安装地址/bin/ 目录下
```

代码

```
import nltk

lp = nltk.sem.logic.LogicParser()           # 调用LogicParser()类
P = lp.parse('man(Socrates)')              # 苏格拉底是人
Q = lp.parse('all x (man(x)->mortal(x))')  # 是人都会死
R = lp.parse('mortal(Socrates)')           # 苏格拉底会死

prover1= nltk.inference.prover9.Prover9()   # 调用prover9类接口
prover1.prove(R,[P,Q])                      # P和Q作为前提, R为结论输入验证结果
```


执行

```
>>> import nltk
>>>
>>> lp = nltk.sem.logic.LogicParser()
>>> P = lp.parse('man(Socrates)')
>>> Q = lp.parse('all x (man(x)->mortal(x))')
>>> R = lp.parse('mortal(Socrates)')
>>>
>>> prover1 = nltk.inference.prover9.Prover9()
>>> prover1.prove(R, [P, Q])
True
>>> |
```

例

- 翻译

每个奇数均存在一个大于它的奇数

- $\text{odd}(x)$: x 是奇数
- $\text{Great}(x, y)$: x 大于 y
- $(\forall x)(\text{odd}(x) \rightarrow (\exists y)(\text{odd}(y) \wedge \text{Great}(y, x)))$

例

已知：

- $(\forall x)(F(x) \rightarrow (G(x) \wedge H(x)))$
- $(\exists x)(F(x) \wedge P(x))$

试证明： $(\exists x)(P(x) \wedge H(x))$

例

- 已知：**
- 1) Jack owns a dog.
 - 2) Every dog owner is an animal lover.
 - 3) No animal lover kills an animal.
 - 4) Either Jack or Curiosity killed the cat, who is named Tuna.

证明： Curiosity killed the cat.

例

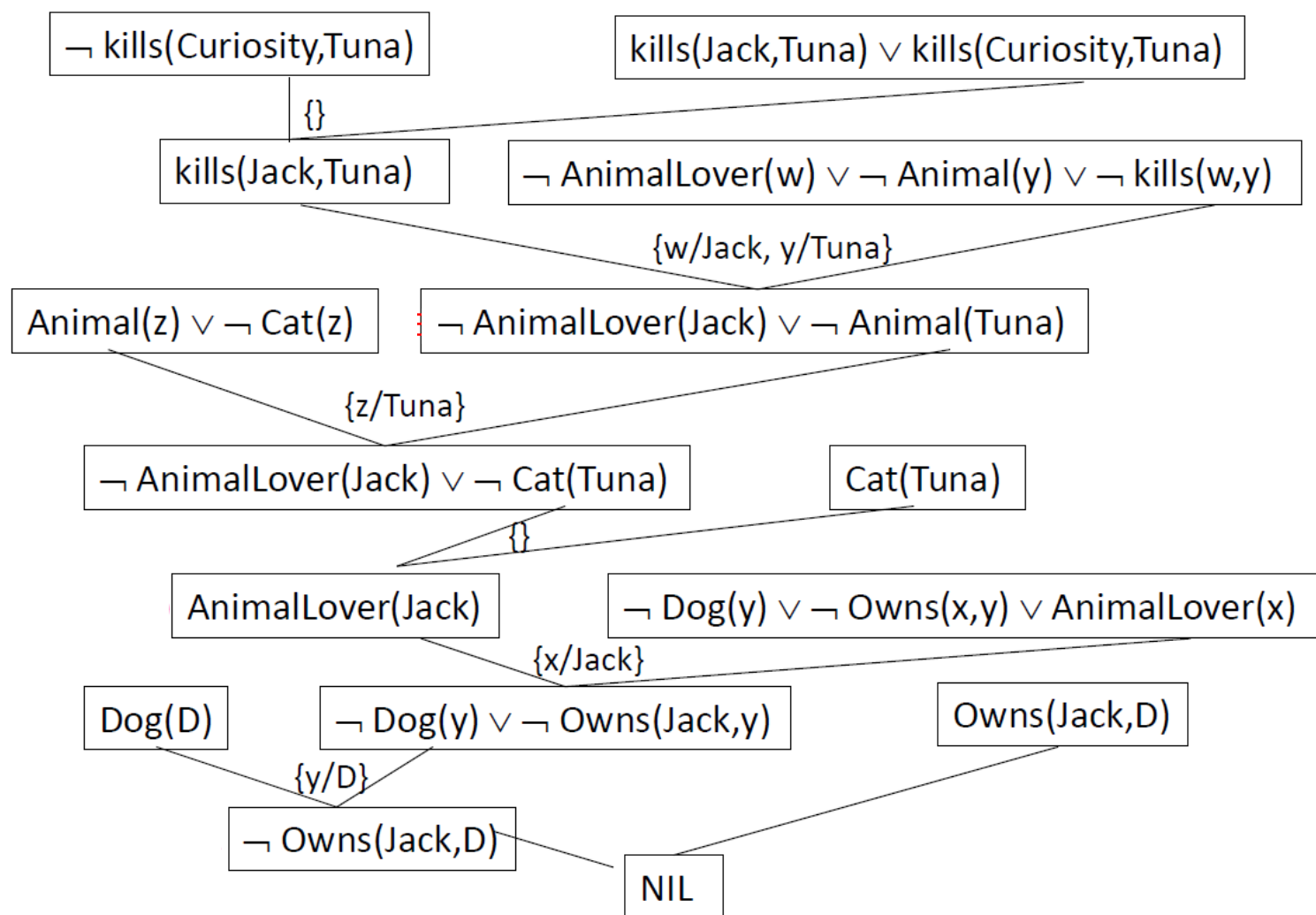
- 已知：
- 1) Jack owns a dog.
 - 2) Every dog owner is an animal lover.
 - 3) No animal lover kills an animal.
 - 4) Either Jack or Curiosity killed the cat, who is named Tuna.

证明： Curiosity killed the cat.

给定的句子加常识：

1. $\exists x : Dog(x) \wedge Owns(Jack, x)$
2. $\forall x : (\exists y Dog(y) \wedge Owns(x, y)) \Rightarrow AnimalLover(x)$
3. $\forall x : AnimalLover(x) \Rightarrow (\forall y : Animal(y) \Rightarrow \neg Kills(x, y))$
4. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
5. $Cat(Tuna)$
6. $\forall x : Cat(x) \Rightarrow Animal(x)$

例



等价式

$$E_{25} \quad (\exists x)(A(x) \vee B(x)) \Leftrightarrow (\exists x)A(x) \vee (\exists x)B(x)$$

$$E_{26} \quad (\forall x)(A(x) \wedge B(x)) \Leftrightarrow (\forall x)A(x) \wedge (\forall x)B(x)$$

$$E_{27} \quad \neg(\exists x)(A(x)) \Leftrightarrow (\forall x)(\neg A(x))$$

$$E_{28} \quad \neg(\forall x)(A(x)) \Leftrightarrow (\exists x)(\neg A(x))$$

$$E_{29} \quad (\forall x)(A(x) \vee P) \Leftrightarrow (\forall x)A(x) \vee P$$

$$E_{30} \quad (\forall x)(A(x) \wedge P) \Leftrightarrow (\forall x)A(x) \wedge P$$

$$E_{31} \quad (\exists x)(A(x) \vee P) \Leftrightarrow (\exists x)A(x) \vee P$$

$$E_{32} \quad (\exists x)(A(x) \wedge P) \Leftrightarrow (\exists x)A(x) \wedge P$$

$$E_{33} \quad (\forall x)(A(x) \rightarrow P) \Leftrightarrow (\exists x)A(x) \rightarrow P$$

$$E_{34} \quad (\exists x)(A(x) \rightarrow P) \Leftrightarrow (\forall x)A(x) \rightarrow P$$

$$E_{35} \quad (\forall x)(P \rightarrow A(x)) \Leftrightarrow P \rightarrow (\forall x)A(x)$$

$$E_{36} \quad (\exists x)(P \rightarrow A(x)) \Leftrightarrow P \rightarrow (\exists x)A(x)$$

$$E_{37} \quad (\exists x)(A(x) \rightarrow B(x)) \Leftrightarrow (\forall x)A(x) \rightarrow (\exists x)B(x)$$

$$I_{16} \quad (\forall x)A(x) \vee (\forall x)B(x) \Rightarrow (\forall x)(A(x) \vee B(x))$$

$$I_{17} \quad (\exists x)(A(x) \wedge B(x)) \Rightarrow (\exists x)A(x) \wedge (\exists x)B(x)$$

$$I_{18} \quad (\exists x)A(x) \rightarrow (\forall x)B(x) \Rightarrow (\forall x)(A(x) \rightarrow B(x))$$

$$I_{19} \quad (\forall x)A(x) \Rightarrow (\exists x)A(x)$$

等价式

$$B_1 \quad (\forall x)(\forall y)L(x, y) \Leftrightarrow (\forall y)(\forall x)L(x, y)$$

$$B_2 \quad (\forall x)(\forall y)L(x, y) \Rightarrow (\exists y)(\forall x)L(x, y)$$

$$B_3 \quad (\forall y)(\forall x)L(x, y) \Rightarrow (\exists x)(\forall y)L(x, y)$$

$$B_4 \quad (\exists y)(\forall x)L(x, y) \Rightarrow (\forall x)(\exists y)L(x, y)$$

$$B_5 \quad (\exists x)(\forall y)L(x, y) \Rightarrow (\forall y)(\exists x)L(x, y)$$

$$B_6 \quad (\forall x)(\exists y)L(x, y) \Rightarrow (\exists y)(\exists x)L(x, y)$$

$$B_7 \quad (\forall y)(\exists x)L(x, y) \Rightarrow (\exists x)(\exists y)L(x, y)$$

$$B_8 \quad (\exists x)(\exists y)L(x, y) \Leftrightarrow (\exists y)(\exists x)L(x, y)$$

回顾

- 命题逻辑的不足
- 谓词逻辑
 - 个体常元、个体变元、论域、谓词、量词、函词、项
 - 约束变元、自由变元
 - 解释
- 谓词逻辑推理问题
 - 变元替换、合一
 - 广义归结、广义肯定前件

End
