人工智能与机器学习

叶琦
工业控制研究所

杭州 • 浙江大学 • 2022

一阶逻辑

7.1 表示方法的回顾

7.2 一阶逻辑的语法和语义

7.3 使用一阶逻辑

7.4 一阶逻辑中的知识工程

知识

■人的智能表示靠反射机制，而是对知识的内部表示进行推理过程。

■逻辑智能体是基于知识的智能体，它采用推理过程来得到关于新世界的表示，并且用这些新表示推到下一步做什么。

■基于知识的智能体，从通用的形式表达的知识中获益，通过对信息的组合和再组合，以适应各种用途。

知识

- Feigenbaum(爱德华·费根鲍姆)认为知识是经过削减、塑造、解释和转换的信息。简单地说，知识是经过加工的信息。
- Bernstein(伯恩施坦)说知识是特定领域的描述、关系和过程组成。
- Hayes-Roth(Frederick Hayes-Roth, 海斯-罗特)认为知识是事实、信念和启发式规则。
- 信息关联后所形成的信息结构：事实&规则。

- **人工智能系统所关心的知识：**一个智能程序高水平的运行需要有关的**事实知识、规则知识、控制知识和元知识。**
  - **事实：**是有关问题环境的一些事物的知识，常以“...是...”的形式出现。如事物的分类、属性、事物间关系、科学事实、客观事实等，在知识库中属于低层的知识。如雪是白色的、鸟有翅膀、张三李四是好朋友。
  - **规则：**是有关问题中与事物的行动、动作相联系的因果关系知识，是动态的，常以“如果...那么...”形式出现。特别是启发式规则是属于专家提供的专门经验知识，这种知识虽无严格解释但很有用处。
  - **控制：**是有关问题的求解步骤，技巧性知识，告诉怎么做一件事。也包括当有多个动作同时被激活时应选哪一个动作来执行的知识。
  - **元知识：**是有关知识的知识，是知识库中的高层知识。包括怎样使用规则、解释规则、校验规则、解释程序结构等知识。

知识表示（knowledge representation）

- 维基百科：**知识表示**是认知科学和人工智能两个领域共同存在的问题。
  - 在**认知科学**里，它关系到人类如何储存和处理资料。
  - 在**人工智能**里，其主要目标为储存知识，让程式能够处理，达到人类的智慧。
- **知识表示**是研究用机器表示知识的可行性、有效性的一般方法，是一种数据结构与控制结构的统一体，既考虑知识的存储又考虑知识的使用。
- **知识表示**可看成是一组描述事物的约定，以**把人类知识表示成机器能处理的数据结构**。

知识特性

- **相对正确性。**一定条件下/某种环境中
- **不确定性。**中间状态/为真程度/随机性/模糊性/经验性/不完全性
- **可表示性。**语言/文字/图像/视频/图形/音频/神经网络/概率图

知识的分类

- **范围。**常识性知识/领域性知识
- **作用。**事实性知识/过程性知识/控制知识
- **确定。**确定性知识/不确定性知识
- **表现。**逻辑性知识/形象性知识
- **抽象。**零级知识/一级知识/二级知识

7.1 表示语言

语言：人类所特有的用来表达意思、交流思想的工具，是一种特殊的社会现象，由语音、词汇和语法构成一定的系统。

自然语言（Natural Language）

就是人类讲的语言，比如汉语、英语和法语。这类语言不是人为设计（虽然有人试图强加一些规则）而是自然进化的。

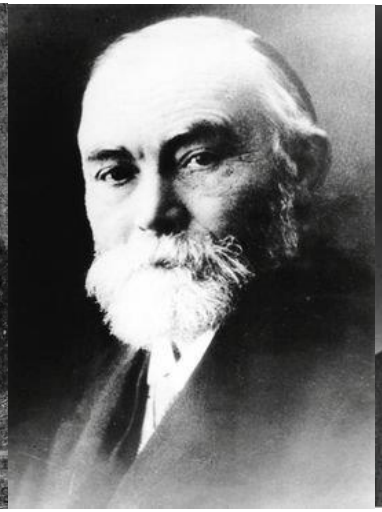
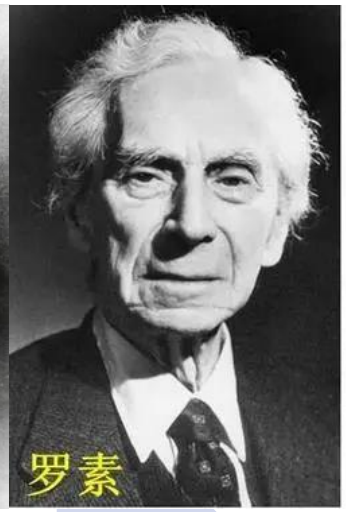
形式语言（Formal Language）

是为了特定应用而人为设计的语言。例如数学家用的数字和运算符号、化学家用的分子式等。是一种由标记和符号组成的形式化系统，包括该系统所容许的表达式的形成和转换的规则。

半形式化语言：自然语言+特定的符号

形式语言



亚里士多德

《前分析篇》三段论

莱布尼茨

设想要建立一种“普遍语言”（即：形式语言）

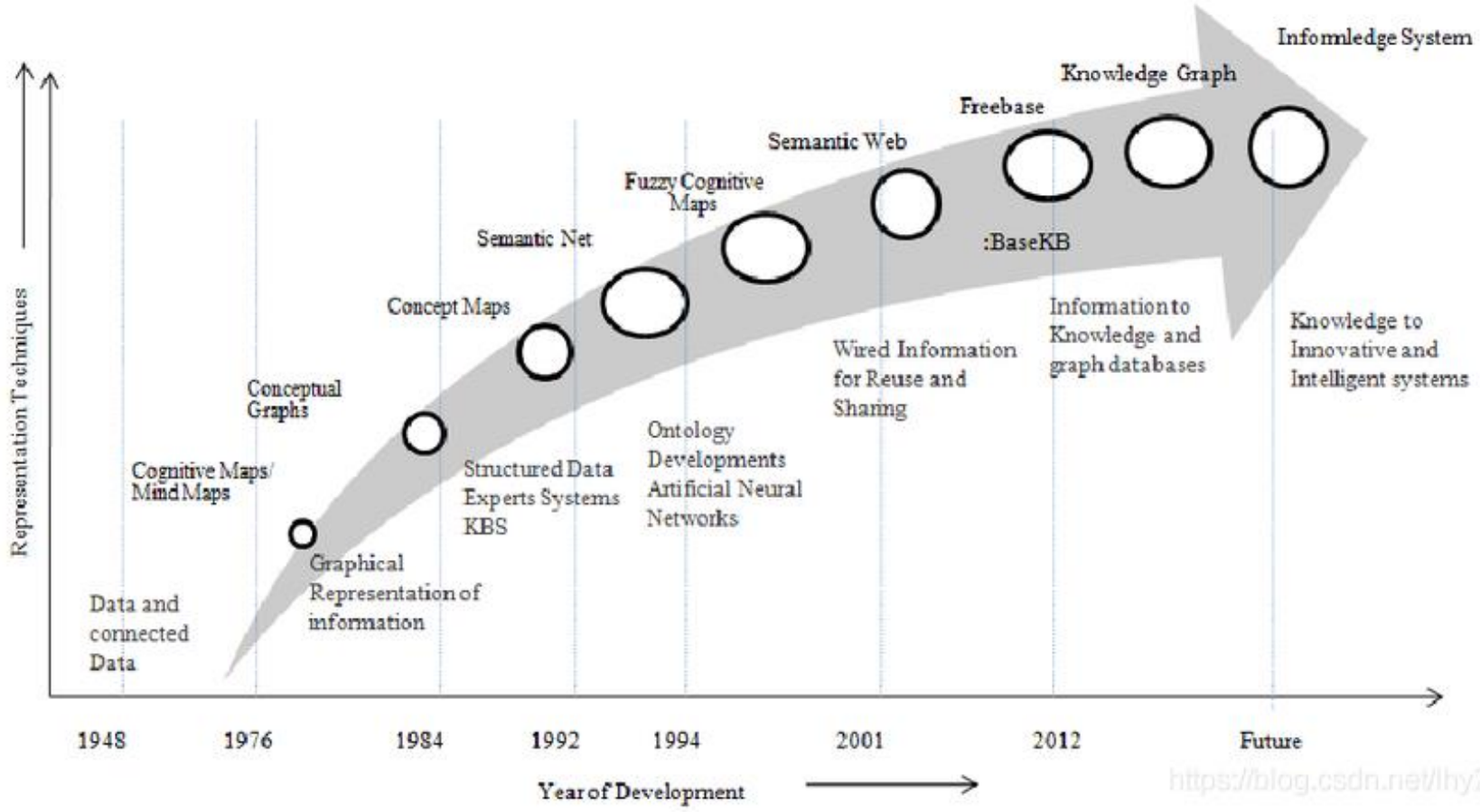
弗雷格

1879《概念文字——一种模仿算术语言构造的纯思维的形式语言》

罗素和怀特海

1910-1913年，《数学原理》发展和完善了弗雷格的形式语言和形式推理系统

所有人都会死
苏格拉底是人
苏格拉底也会死



细节上来说，知识表示主要经历了从一阶谓词逻辑表示、产生式规则、框架表示法、脚本表示法、语义网表示法、知识图谱表示法等几种表示。

形式语言 vs. 自然语言

形式语言			自然语言
程序设计语言	命题逻辑	一阶谓词逻辑	
过程性，本身只表示计算过程，无法从一个事实推导出新的事实；推理和表示分开，推理完全不依赖于领域	描述性/陈述性，？	在形式上可接近于人类自然语言，对于过程性和非确定性知识表达有限	
程序以及数据库中的数据结构，缺乏简便的表述方式，无法处理不确定性信息	有特定的语法，可以采用析取式和否定式来处理不完全信息；具有合成性，例如B1,1 P1,2 的含义从 B1,1和 P1,2得到。	表示能力较差，只能表达确定性知识	有语法特征？
	含义与上下文无关，语义基于语句和可能世界之间的真值关系、无歧义性	由于知识之间是相互独立的，知识与知识之间缺乏关联，使得知识管理实施相对困难	**语义取决于上下文、有歧义性**
	缺乏足够的表达能力，无法简洁描述有多个对象的环境	较为精确，表达自然	**非常强大的表达能力**

回顾：命题逻辑的基本概念

- **命题（proposition）**就是非真即假的陈述句。
  - 命题的真假，称为真值，“真”记为T（True）或1，“假”记为F（False）或0. 因为真值只有两种，这种逻辑也称为二值逻辑。
  - 数学上，为了符号化，用字母来表示任意命题，称为命题变项/命题变元。例如，可以用P,Q,R三个字母各表示一个命题。
- **命题联结词（connective）**可以把命题与命题联结起来，构成新的命题。命题联结词是命题的运算符，相当于1+2中的“加号”。
  - 常用的命题联结词：否定词¬（非），合取词∧（且），析取词∨（或），蕴含词⇒（推出，如果...那么……），等价词⇔（当且仅当）
- **命题公式的分类**
  - 重言式/永真式（tautology）【重（chóng）言：反复说，犹指“废话”。】
  - 矛盾式/永假式/不可满足式
  - 仅可满足式/可真可假式

结合形式语言和自然语言的优势：采用命题逻辑的基础—陈述式、上下文无关、无歧义和合成语义，并借用自然语言的思想，构造更具表达能力的逻辑。

在一阶谓词逻辑中，有三个东西去描述这个世界：

- **object对象：**一个一个的个体，一个人.....
- **relations关系：**是特殊的function函数，自变量是对象object，函数值要么是真要么是假（可以是一元，二元.....），比如我们定义father()是一个relation的话，结果就是true或者false。father(A,B)，相比之下，relation的范围更广。
- **functions函数：**是一个具体的函数，自变量是一个对象object，函数值是另外一个object，可以从一个object映射到另外一个object（可以　是一元，二元.....），比如定义一个father的function的话，father(B)的输出就是A，是一个one to one 的过程。

断言：1加2等于3
对象：1、2、3、1加2
关系：等于
函数：加

断言：与Wumpus相邻的方格是有臭味的
对象：Wumpus，方格
属性：有臭味的
关系：相邻

一阶逻辑语言

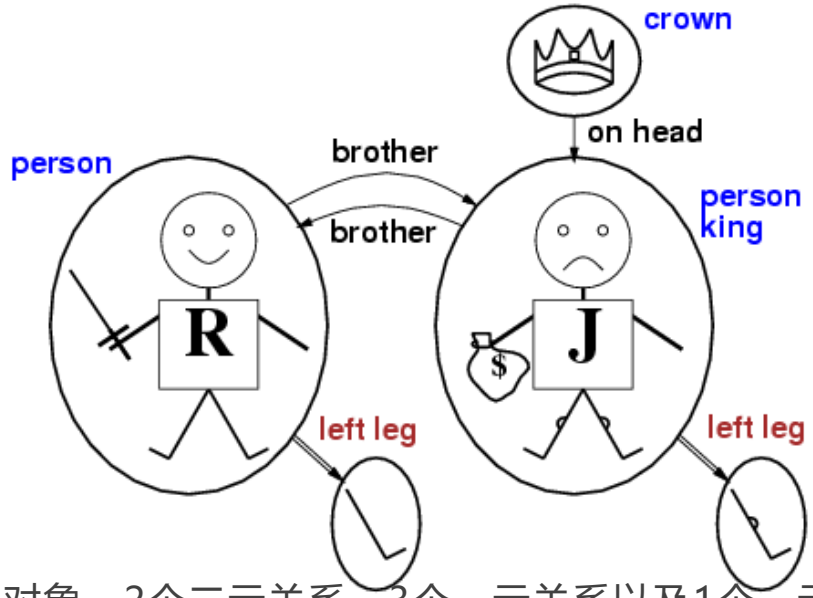
- 一阶逻辑的本质是一种语言，类似于我们的常用自然语言(例如，汉语，英语等，这里我们用英语来说明)。
- 一个语言包括的要素有字母表(26个英文字母)，单词，语句，语法，语义等
- 对于一阶逻辑语言来说，它包含的要素有
  - 字母表(对应于英文字母表)
  - 项(这里也可以将项理解为英文字母表)
  - 原子语句/原子公式(对应于英文单词)
  - 语句/公式(对应于英文语句)
  - 语法(对应于英文语法)
  - 语义(对应于英语中句子的含义)

理解“一阶逻辑的语义”

- 一个一阶逻辑**公式（语句）**就是一阶逻辑语言字母表上的**符号串** (这类似于一个英文句子是英文字母表上的字符串)。
- 一阶逻辑的**语法**的本质是规定了一阶逻辑语言字母表上的**什么样的符号串才是合法的**一阶逻辑公式。
- 一个公式本身是没有任何含义的，当我们**给公式中的字符都赋予了一定的含义**之后(即给公式中的字符做**解释**)，该公式就有了含义，这样就达到了将一些知识(或客观事实) 符号化的目的。
- 因此，如果我们想用一阶逻辑来描述一些知识，我们不仅要有一个合法的公式的集合(即合乎语法的公式)，还要对这些公式赋予一定的含义(即给出每个公式的语义)。
- 最后，将知识符号化 (即知识表示) 的终极目的是为了进行知识的推理，这也是一阶逻辑常用在知识表示与推理中的一个原因。

一阶逻辑的语义

- **论域：**讨论的范围（对象集）。例如，要研究一个学校里学生选课的情况，那么讨论的范围就限于该校的所有的学生以及该校开设的所有的课程。一般而言，一个论域是一个非空对象的集合，用 M来表示。
- **模型(结构)：**对常量符号（表示对象），谓词符号（表示关系）和函词（表示函数）的**解释**再加上**论域**就构成了一个模型(或结构)。
- 命题逻辑可以通过枚举所有可能模型以检验蕴涵，在一阶逻辑中不可行



包含5个对象、2个二元关系、3个一元关系以及1个一元函数左腿的模型

解释的作用：

- 将常量符号**映射**到对象。
- 将谓词符号**映射**到对象之间的关系。
- 将函词符号**映射**到对象上的函数

谓词的值域为$\{True, false\}$，函词的值域为对象的子集。

- 常量符号被解释为论域中的个体，即 $c^{\mathcal{M}} \in M$；
- 谓词符号被解释为论域上的关系，即若 P 是一个 n 元谓词符号，则 $P^{\mathcal{M}} \subseteq M^n$，即一个 n 元谓词符号被解释为论域 M 上的n元关系；
- 函数符号被解释为论域上的函数，即若 f 是一个 n 元函数符号，则 $f^{\mathcal{M}} : M^n \to M$，即一个 n 元函数符号被解释为论域 M 上的 n元函数。

注：特别的，变量符号被解释为论域上的变量，即$x^{\mathcal{M}} = x$。

字母表（句法，syntax）

Constants	$KingJohn,\ 2,\ UCB, \ldots$
Predicates	$Brother,\ >, \ldots$
Functions	$Sqrt,\ LeftLegOf, \ldots$
Variables	$x,\ y,\ a,\ b, \ldots$
Connectives	$\wedge\ \vee\ \neg\ \Rightarrow\ \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall\ \exists$

常量	*King* , *John*,
谓词	*Brother*, *Person*, …
函数	*Sum*, *FatherOf*, …
变量	*x*, *y*, *a*, *b*, …
连接词	¬、∧、∨、→/⇒、↔/⇔
等号	=
量词	∃，∀

项（term）

项是指代**对象**的逻辑表达式。

复合项： 由**函词**以及紧随其后的参数、被括号括起来的列表项组成。

如　　*Leftleg*(John)

一阶逻辑中的项通常用 t 来表示，项的递归定义如下：

- 一个常量符号 c 是一个项;
- 一个变量符号 x 是一个项;
- 如果 $t_1, \ldots, t_n$ 是项，f 是一个n元函数符号，则$f(t_1, \ldots, t_n)$也是一个项。

原子语句（atomic sentence）

项：指代对象
谓词：指代关系
（with value true or false）

→ 原子语句：陈述事实
represents a relation between terms

谓词符号(列表项)

Brother(Richard, John)

Married(*Father*(Richard), *Mother*(John))

Atomic sentence $= predicate(term_1, \ldots, term_n)$
or $term_1 = term_2$

Term $= function(term_1, \ldots, term_n)$
or *constant* or *variable*

E.g., $Brother(KingJohn, RichardTheLionheart)$
$> (Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))$

复合/复杂语句（complex sentence）

复合语句：由逻辑连接词（与、或、非、蕴含、等价）连接原子语句构成的语句

Atom(s) joined together using logical connectives *and/or quantifiers*

$S1 \wedge S2$　$S1 \vee S2$　$\neg S$　$S1 \Rightarrow S2$　$S1 \Leftrightarrow S2$

例如：　*Sibling* (John, Richard) ⇔ *Sibling* (Richard, John)

Older (John, 30) ∨ *Younger* (John, 30)

量词

全称量词 ∀

读作“任意”。对**每个**对象进行陈述。

∀ x: P (x)

表示 P (x) 对于所有 x 为真。

存在量词 ∃:

读作“存在”。对全域中**某些**对象进行陈述而无需对它命名。

∃x: P (x)

表示 P (x) 对于域中某些 x 为真。

全称量词通常接**蕴含符号 ⇒**

例如:

所有的国王都是人。

∀x King(x) ⇒ Person(x)

存在量词通常接**合取符号 ∧**

例如:

有王冠在King John头上。

∃x Crown(x) ∧ OnHead(x,John)

全称量词∀

- 对**每个对象**进行陈述
- 形式：∀ *variable sentence*，例如

$$\forall x,\ King(x) \Rightarrow Person(x)$$

- ⇒ 是使用∀时的**自然连接符**
- 经常犯的错误是用合取式代替蕴涵，导致过强的陈述.
  - 例如，∀x，At(x,ZJU) ∧ Smart(x)的含义是“每个人都在ZJU，而且每个人都聪明”。
  - 显然这没有抓住我们要表述的内涵。

$\forall \langle variables \rangle \quad \langle sentence \rangle$

*Zju*的每个人都聪明:

$\forall x \quad At(x, Zju) \Rightarrow Smart(x)$

在模型*m*中 $\forall x \quad P$为真，当且仅当*x*取模型m中所有可能的对象时*P*都为真

大致上等价于P的**实例化**的**合取**

$At(John, Zju) \Rightarrow Smart(John)$
$\wedge\ At(Tom, Zju) \Rightarrow Smart(Tom)$
$\wedge\ At(Zhu, Zju) \Rightarrow Smart(Zhu)$
$\wedge \ldots$

存在量词∃

- 对全域中**某些对象**进行陈述而无需对它命名
- 例如： ∃x，Crown(x)∧OnHead(x,John)
- ∧是使用∃时的自然连接符
- 在∃句中用⇒代替∧会导致过弱的陈述

存在量词

$\exists \langle variables \rangle \quad \langle sentence \rangle$

*Zju*的某人很聪明：　　$\exists x \quad At(x, Zju) \wedge Smart(x)$

在模型*m*中 $\exists x \quad P$ 为真，只要*x*为模型中某个对象时*P*为真

大致上等价于*P*的实例化的析取

$At(John, Zju) \wedge Smart(John)$
$\vee At(Tom, Zju) \wedge Smart(Tom)$
$\vee At(Zju, Zju) \wedge Smart(Zju)$
$\vee \ldots$

量词嵌套

Loves(x,y)：x爱y

∀x∃y Loves(x,y)：每个人都会爱上某人

∃y∀x Loves(x,y)：存在某个人被每个人爱

∀x∀y is the same as ∀y∀x

∃x∃y is the same as ∃y∃x

∃x∀y is not the same as ∀y∃x

量词等价

量词的**二元性**：每个都可以用另一个来表达，∃ 和 ∀ 可通过“非”操作而联系

“每个人都喜欢冰淇淋”意味着“没有一个人不喜欢冰淇淋”

“存在某个喜欢巧克力的人”意味着“并不是所有人都不喜欢巧克力”

$$\forall x \quad Likes(x, IceCream) \quad \text{等价于} \quad \neg\exists x \quad \neg Likes(x, IceCream)$$

$$\exists x \quad Likes(x, Chocolate) \quad \text{等价于} \quad \neg\forall x \quad \neg Likes(x, Chocolate)$$

等词

- 除了使用谓词和项产生原子语句之外，一阶逻辑还有另一种构造原子语句的方式。可以用**等词（等号=）**来表声明两个项指代同一个对象。
  - 可用于表述关于一个给定函数的事实
    - $Father\ (John) = Henry$
  - 加否定词时表示两个项不是同一个对象。
    - $\exists x,y\ \ Brother(x,Richard) \wedge Brother(y,Richard) \wedge \neg (x=y)$

    表示*Richard*至少有两个兄弟

$x \neq y$ 可以作为 $\neg (x=y)$的缩写

一阶逻辑的真值

- 语义必须把语句和模型结合起来以判定真值
- 为此，需要一个对分别被常量、谓词和函数符号指代的对象、关系和函数进行详细说明的解释
- 语句的真值由一个模型和对句子符号的解释来判定

Sentences are true with respect to a model and an interpretation

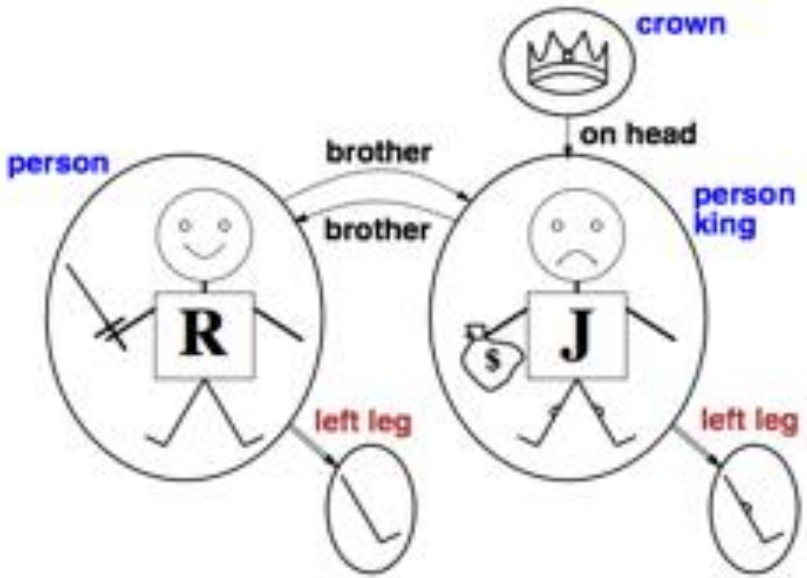
Model contains ≥ 1 objects (domain elements) and relations among them

Interpretation specifies referents for
- constant symbols → objects
- predicate symbols → relations
- function symbols → functional relations

An atomic sentence $predicate(term_1, \ldots, term_n)$ is true
iff the objects referred to by $term_1, \ldots, term_n$
are in the relation referred to by $predicate$

可满足性

- 如果在所有的模型中都为真，语句是永真的，
  - True
  - $A \vee \neg A$
  - $A \Rightarrow A$
  - $(A \wedge (A \Rightarrow B)) \Rightarrow B$
- 语句是可满足的，如果在某些模型中为真，
  - $A \vee B$
  - C
- 语句是不可满足的(永假)，如果在所有模型中都不为真，
  - $A \wedge \neg A$



Consider the interpretation in which
$Richard$ → Richard the Lionheart
$John$ → the evil King John
$Brother$ → the brotherhood relation

Under this interpretation, $Brother(Richard, John)$ is true just in case Richard the Lionheart and the evil King John are in the brotherhood relation in the model

命题逻辑中的蕴涵可以通过枚举模型来计算，在一阶谓词逻辑中，我们可以枚举出给定知识库词汇表KB的FOL模型（一阶谓词逻辑模型）

Entailment in propositional logic can be computed by enumerating models

We **can** enumerate the FOL models for a given KB vocabulary:

For each number of domain elements n from 1 to ∞
  For each k-ary predicate P_k in the vocabulary
    For each possible k-ary relation on n objects
      For each constant symbol C in the vocabulary
        For each choice of referent for C from n objects . . .

Computing entailment by enumerating FOL models is not easy!

通过列举FOL模型计算蕴涵是不容易的!

另一种语义：数据库语义

例如：*Richard*有两个兄弟*John*和*Geoffrey*。

$Brother(John,Richard) \wedge Borther(Geoffrey,Richard)$

错误：（没有排除John和Geoffery是同一个人）

$Brother(John,Richard) \wedge Borther(Geoffrey,Richard) \wedge \neg(John = Geoffrey)$

错误：（没有排除Richard还有其他更多兄弟的模型）

正确： $Brother(John,Richard) \wedge Borther(Geoffrey,Richard) \wedge \neg(John = Geoffrey) \wedge$

$\forall x\ Borther(x, Richard) \Rightarrow (x=John \vee x=Geoffrey)$

数据库语义:

关键字假设: 每个常量符号指代一个确定对象.

封闭世界假设: 我们不知道的所有原子语句事实上都为假.

论域闭包: 每个模型只包括常量符号指代的对象.

$Brother(John, Richard) \wedge Brother(Geoffrey, Richard)$

在数据库语义下, 也表示Richard只有两个兄弟John和Geoffrey.

高阶逻辑：把一阶逻辑中的关系和函数本身也视为对象。
比一阶逻辑的表达能力更强。

Higher-order logic also allows quantification over relations and functions.

- e.g., "two objects are equal iff all properties applied to them are equivalent":
- $(\forall x,y)\ ((x=y) \Leftrightarrow (\forall p,\ p(x) \Leftrightarrow p(y)))$

Higher-order logics are more expressive than first-order; we have little understanding on how to effectively reason with sentences in higher-order logic.

小结：一阶逻辑

一阶逻辑语言是围绕对象和关系建立起来的。

语言	本体论约定 （现实世界）	认识论约定 （agent所相信的事实）
命题逻辑	事实	真/假/未知
一阶逻辑	事实，对象，关系	真/假/未知
时态逻辑	事实，对象，关系，时间	真/假/未知
概率论	事实	可信度[0,1]
模糊逻辑	事实，真实度[0,1]	已知区间论

- 命题逻辑：作为一种表示语言，足以阐述逻辑和基于知识的agent的基本概念。但一种表达能力很弱，无法以简洁的方式表示复杂环境的知识。
- 一阶逻辑：具有丰富的表达能力，可以表示大量常识知识。

7.3 如何使用一阶逻辑语言

- 一阶逻辑的断言和查询
- 亲属关系域
- 数、集合和列表
- Wumpus世界

7.3.1 FOL的断言(Assertion)和查询(Query)

TELL将语句添加到知识库。这样的语句叫**断言**。

例如，　断言*John*是国王　TELL(*KB*, *King(John)*)

断言*Richard*是人　TELL(*KB*, *Person(Richard)*)

断言国王是人　TELL(*KB*, $\forall x\ King(x) \Rightarrow Person(x)$)

ASK向知识库询问问题。用ASK提出的问题被称为**查询**或目标。

例如，　ASK(*KB*, *King(John)*)

返回*True*

一般而言，被知识库逻辑蕴涵的任何查询都肯定可以得到回答。

例如，　已知上述断言，查询　ASK(*KB*, *Person(John)*)

也应该返回*True*

可以提出量化查询，如　ASK(*KB*, $\exists x\ Person(x)$)

返回*True*

ASKVARS询问什么样的*x*使得语句为真

例如，　ASKVARS(*KB*, *Person(x)*)

返回**答案流**，如$\{x/John\}$和$\{x/Richard\}$

这样的答案被称为**置换**或**绑定表**。

7.3.2 亲属关系论域（the kinship domain）

函数(function)：	返回“值”，	如 father_of(Mary) = Bill
谓词(predicate)：	返回“真”或“假”，	如 father_of(Mary, Bill)

一元谓词: *Male, Female*

二元谓词: *Parent, Sibling, Brother, Sister, Child, Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, Uncle*

函数: *Mother*、*Father*

考察每个函数和谓词，可以按照符号定义写出我们知道的知识。如：

- 兄弟是同胞兄弟姐妹关系

$$\forall x,y \quad Brother(x,y) \Leftrightarrow Sibling(x,y)$$

- 某人的母亲是此人的女性双亲

$$\forall m,c \quad Mother(c) = m \Leftrightarrow (Female(m) \wedge Parent(m,c))$$

不是所有关于论域的逻辑语句都是公理,有些是定理——它们通过公理推导而来。

同胞关系的对称性	$\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$

- **公理：**提供基本的事实信息，根据这些信息可以推导出有用的结论，是推理的基础，也是构成知识库的基础
- **定义：**公理+谓词
- **定理：**通过公理发展出来的，即可由公理通过推理得到的
- 从**纯逻辑观点**看：知识库只要有公理，无需定理
- 从**实用观点**看：定理对降低计算成本极为重要

7.3.3 数、集合和表

以**自然数**为例：谓词 *NatNum* 表示是否自然数; 常数符号0; 函词S（后继）

自然数的递归定义

$$NatNum(n)$$

$$\forall n, NatNum(n) \Rightarrow NatNum(S(n))$$

即：0是自然数，而且对于每个对象n，如果n是自然数，那么S(n)是自然数。

对后继函数的约束

$$\forall n, 0 \neq S(n)$$

$$\forall m, n\ m \neq n \Rightarrow S(m) \neq S(n)$$

用后继函数来定义加法

$$\forall m, Natnum(m) \Rightarrow +(0, m) = m$$

$$\forall m, n\ Natnum(m) \wedge Natnum(n) \Rightarrow +(S(m), n) = S(+(m, n))$$

第一条：0加上任何自然数m得到m本身

第二条：+(m,0)可以写成m+0；S(n)可以写成n+1

$$\forall m, n\ Natnum(m) \wedge Natnum(n) \Rightarrow (m + 1) + n = (m + n) + 1$$

集合

一元谓词	**Set**		
二元谓词	**$x \in s$**	**$s1 \subseteq s2$**	
二元函词	**$\{x|s\}$**	**$s_1 \cup s_2$**	**$s_1 \cap s_2$**

(1) 集合是空集或通过将一些元素添加到集合中而构成

$$\forall s\ Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_2\ Set(s_2) \wedge s = \{x|s_2\})$$

(2) 空集没有任何元素 $\neg\exists x, s\ \{x|s\} = \{\}$

(3) 将已经存在于集合的元素添加到集合中，该集合无任何变化

$$\forall x, s \quad x \in s \Leftrightarrow s = \{x|s\}$$

(4) 集合的元素是那些被添加到集合中的元素。（递归的表示方式）

$$\forall x, s\ x \in s \Leftrightarrow [\exists y, s_2\ (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))]$$

(5) $\forall s_1, s_2\ s_1 \subseteq s_2 \Leftrightarrow (\forall x\ x \in s_1 \Rightarrow x \in s_2)$ 子集

(6) $\forall s_1, s_2\ (s_1 = s_2) \Leftrightarrow (s1 \subseteq s_2 \wedge s_2 \subseteq s_1)$ 集合相等

(7) $\forall x, s_1, s_2\ x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$ 交集

(8) $\forall x, s_1, s_2\ x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$ 并集

7.3.4 Wumpus世界

任务环境的精确定义（PEAS）

- **性能度量：** gold+1000, death-1000, action-1, using the arrow -10.
- **环境**
  - 4×4网格
  - 智能体初始在[1,1]，面向右方
  - 金子和wumpus在[1,1]之外随机均匀分布
  - [1,1]之外的任意方格是陷阱的概率是0.2
- **执行器**
  - 智能体可向前、左转或右转
  - 智能体如果进入一个有陷阱或者活着的wumpus的方格，将死去。
  - 如果智能体前方有一堵墙，那么向前移动无效
  - Grab：捡起智能体所在方格中的一个物体
  - Shoot：向智能体所正对方向射箭（只有一枝箭）
- **传感器**
  - 在wumpus所在之处以及与之直接相邻的方格内，智能体可以感知到臭气。
  - 在与陷阱直接相邻的方格内，智能体可以感知到微风。
  - 在金子所处的方格内，智能体可以感知到闪闪金光。
  - 当智能体撞到墙时，它感受到撞击。
  - 当wumpus被杀死时，它发出洞穴内任何地方都可感知到的悲惨嚎叫。
  - 以5个符号的列表形式将感知信息提供给智能体，例如(stench, breeze, none, none, none)。

例子：和Wumpus世界互动

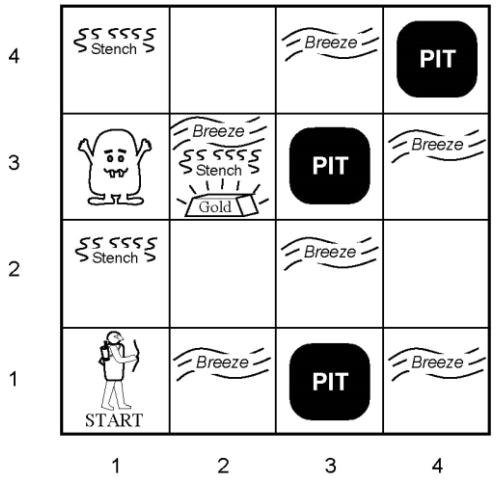
智能体使用FOL KB，并在t=5时感知到臭气和微风（没有看到闪光），**断言**:

TELL(KB, Percept ([Stench, Breeze, *None,* None, None], 5))

智能体向FOL KB**查询**，在t=5时有没有最好的行动

ASK(KB, a BestAction(a,5))

回答: Yes, {a/Shoot}



智能体采用FOL KB，并在t = 5时感知到臭气、微风和闪光，告知KB当前的感知：

TELL(KB, Percept ([Stench, Breeze, *Glitter,* None, None], 5))

行动可用逻辑项表示：

Turn(Right), Turn(Left), Forward, Shoot, Grab

为了选择一个行为，可构造一个查询

ASK (KB, $\exists a \ Action(a, 5)$)

返回一个绑定列表 {a/Grab}

如何用FOL表示 妖兽世界的感知和行为

原始感知数据暗示关于当前状态的某些事实

$$\forall t,s,g,m,c \quad Percept([s,Breeze,g,m,c],t) \Rightarrow Breeze(t)$$
$$\forall t,s,b,m,c \quad Percept([s,b,Glitter,m,c],t) \Rightarrow Glitter(t)$$

简单“反射”行为可以由**量化蕴涵**语句实现

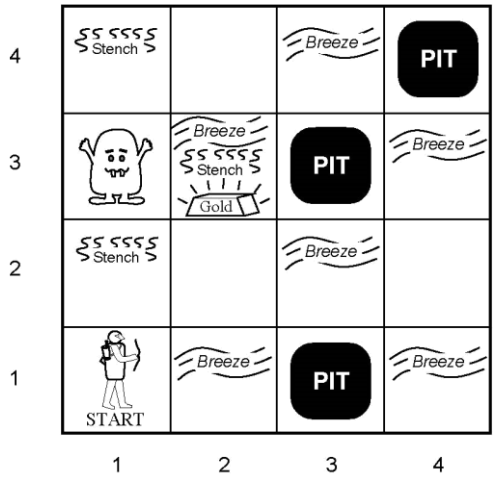
$$\forall t \quad Glitter(t) \Rightarrow Action(Grab,t)$$

如何用FOL表示/描述 环境

- **方格**的描述：用列表项[1, 2]表示$Square_{1,2}$，
  任意两个方格相邻可表示为：

$$\forall x, y, a, b \quad Adjacent([x, y], [a, b])$$
$$(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$$

- **陷阱**的描述：用**一元谓词**$Pit([x, y])$来描述，
  如果方格$[x, y]$中包含陷阱则为真
- **妖兽**的描述：用**函数**$Home(Wumpus)$来描述，
  指代Wumpus所在的那个方格
- 智能体的**位置**随时间变化，用$At(Agent, s, t)$
  来表示智能体在时间t位于方格s



演绎隐藏特性

- 方格的属性：

$$\forall s,t \quad At(Agent,s,t) \wedge Smell(t) \Rightarrow Smelly(s)$$

$$\forall s,t \quad At(Agent,s,t) \wedge Breeze(t) \Rightarrow Breezy(s)$$

- 给定当前位置，可以推测出方格的一些性质。**注意，方格的属性不会随时间而变化**
- 与陷阱相邻的方格中有微风：
  - **诊断规则** — 从结果推出原因

$$\forall y \quad Breezy(y) \Rightarrow \quad \exists x \quad Pit(x) \wedge Adjacent(x,y)$$

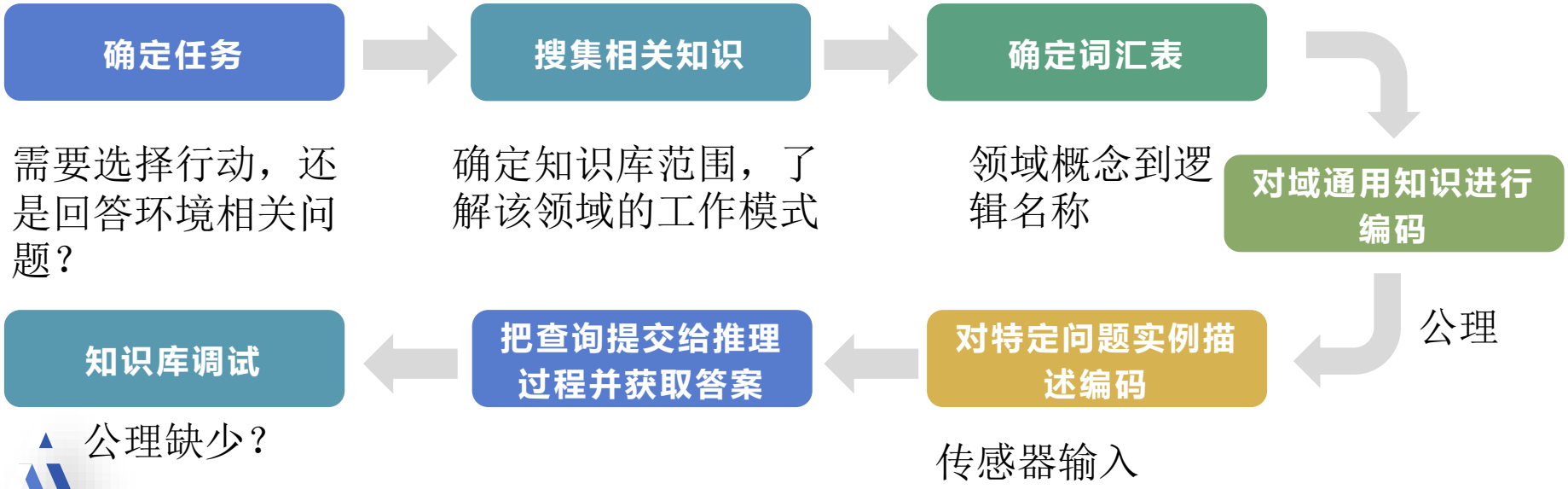
  - **因果规则** — 从原因推导结果

$$\forall x,y \quad Pit(x) \wedge Adjacent(x,y) \Rightarrow Breezy(y)$$

- 这两种方法都不完备 —— 例如，因果规则不能演绎出远离陷阱的方格是否也会有微风；诊断规则不能推出Pit到底在哪个方格中。

7.4 一阶逻辑中的知识工程

知识工程的概念是1977年美国斯坦福大学计算机科学家费根鲍姆教授(E．A．Feigenbaum)在第五届国际人工智能会议上提出的。可以看成是人工智能在知识信息处理方面的发展，研究**如何由计算机表示知识，进行问题的自动求解**。



一阶逻辑小结

- 知识表示语言应该是陈述性的、可合成的、有表达力的、上下文无关的以及无歧义的。
- 逻辑学在本体论约定和知识论约定上存在着不同。命题逻辑只是对事实的存在进行限定,而一阶逻辑对于对象和关系的存在进行限定,因此有更强的表达力。
- 一阶逻辑的语法建立在命题逻辑的基础上。它增加项来表示对象,并且使用全称量词和存在量词对变元进行量化来构建断言。
- 一阶逻辑的可能世界或模型包括通过对象集和解释,解释把常量符号映射到对象,谓词符号映射成对象之间的关系,函词映射成对象上的函数。
- 原子语句仅在谓词所表示的关系在项所指代的对象上成立时为真。扩展解释将量化的变元映射到对象上,定义了量化语句的真值表。
- 用一阶逻辑开发知识库是一个细致的过程,包括对领域进行分析、选定词汇表、对推理结论必不可少的公理进行编码。

回顾：命题逻辑 PL vs. 一阶逻辑 FOL

命题逻辑（Propositional Logic），简称PL

- 形如 ¬P，P∧Q，P∨Q，P → Q ，P↔Q的语句，值为True或者False
- 推理规则较简单，往往通过（1.真值表 2.为数不多的推理规则，例如Modus ponesn等几个）
- 缺点，不能或者很难表示复杂的语句，不能记录推理过程中的变化

一阶逻辑（First Order Logic），简称FOL

包含的东西有

常量（Constant symbol），谓词符号（Predicate symbol），函数符号（Function symbol），变量（Variable），连词（ ∧∨→↔），量词（Quantifiers, ∃∀），例如：Father(Mary) = Bob father_of(Mary, Bob)

回顾：命题逻辑的推理规则

假言推理规则（Modus Ponens，拉丁文）

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$

只要给定任何形式为$\alpha \Rightarrow \beta$和α的语句，就可以推导出语句β

消去合取词

$$\frac{\alpha \wedge \beta}{\alpha}$$

可以从合取式推导出任何合取子式。

单元归结（unit resolution）推理规则　l_i和m是互补文字

$$\frac{l_1 \vee \cdots \vee l_k,\quad m}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k}$$

全归结（full resolution）推理规则　l_i和m_j是互补文字

$$\frac{l_1 \vee \cdots \vee l_k,\quad m_1 \vee \cdots \vee m_n}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

7.5 一阶逻辑的推理

- 简化为命题推理 Reduction
- 直接在一阶逻辑中推理

首先需要将量化语句用非量化语句替换（实例化语句）

量词的推理规则

- **全称量词实例化**

- **存在量词实例化**

代换(Substitution)

- FOL is Similar to PL
  - Important differences
    - Quantifiers
    - Variables
- Important concept: *substitution*(代换)
  - Subst(θ, α), θ is like {x/Michael, y/Bob}
  - replace variables with terms

$$(\forall x)(Man(x) \Rightarrow Mortal(x))$$

$$\text{Subst}(\{x/Michael\}, (\forall x)(Man(x) \Rightarrow Mortal(x)))$$

$$Man(Michael) \Rightarrow Mortal(Michael)$$

注意：

常量不可以代换，变量才可以代换
就算是用常量代替常量也不可以

- Subst({Michael/x}, Man(Michael) ⇒ Mortal(Michael))
  = Man(x) ⇒ Mortal(x)　　no

- Subst({Michael/Bob}, Man(Michael)
  ⇒Mortal(Michael))
  = Man(Bob) ⇒ Mortal(Bob)　　no

- Subst({x/Bob}, Man(x) ⇒ Mortal(x))
  = Man(Bob) ⇒ Mortal(Bob)　　yes

全称量词实例化推理规则（UI规则）

对于任意变量v和基项（没有变量的项）g有：

$$\frac{\forall v \quad \alpha}{\mathrm{SUBST}(\{v/g\},\alpha)}$$

例如：$\forall x \quad King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

经过**代换**后可得

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$
$$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$$
$$\ldots$$

全称量词可多次实例化得到多个不同的结果

存在量词实例化推理规则（EI规则）

对于任意语句的α，变量v，以及从未在知识库中出现过的常量符号k 有：

$$\frac{\exists v \quad \alpha}{\mathrm{SUBST}(\{v/k\},\alpha)}$$

例如：

$$\exists x \quad Crown(x) \wedge OnHead(x, John)$$

经过置换｛x/C1｝后可得

$$Crown(C_1) \wedge OnHead(C_1, John)$$

在全称量词辖域内的存在量词

- Existential instantiation
  - consider: Everyone has a mother.
  - $\forall y \exists x$ Mother(x,y)
  - how to instantiate x?
  - $\forall y$ Mother(m(y),y)
  - m(y) is a Skolem function
- Instantiate

$(\forall x_1)\ldots(\forall x_{k-1})(\exists x_k)(Qx_{k+1})\ldots(Qx_n)M(x_1, \ldots, x_k, \ldots, x_n)$

replace x_k with a Skolem function $f(x_1, \ldots, x_{k-1})$

$(\forall x_1)\ldots(\forall x_{k-1})(Qx_{k+1})\ldots(Qx_n)M(x_1,\ldots,f(x_1,\ldots,x_{k-1}),\ldots,x_n)$

Skolem函数： $(\forall y)[(\exists x)P(x, y)]$中，存在量词是在全称量词的辖域内，允许所存在的x可能依赖于y值。令这种依赖关系明显地由函数g(y)所定义，它把每个y值映射到存在的那个x。这种函数叫做Skolem函数。

如果用Skolem函数代替存在的x，就可以消去全部存在量词，并写成：

$$(\forall y)P(g(y), y)]$$

存在量词只能代换一次

- 对同一条语句的同一个量词，全称量词实例化推理规则可运用多次，存在量词实例化推理规则只能运用一次
  - ∃x Kill (x, Victim)
  - There exists a *x* who killed *Victim*.
    - Someone killed the victim
    - Maybe more than one person killed the victim
    - Existential quantifier says at least one person was killer
  - Replacement is
    - Kill (Murderer, Victim)

简化到命题推理Reduction

一阶语句命题化

- 存在量化语句可以被实例化语句代替
- 全称量化语句可以被所有可能的实例化集代替

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

实例化后删除全称量化语句

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

King(John)等可看作命题符号，
该知识库本质上就是命题逻辑

知识库中包含函词时

- 当知识库中包含函词时，**基项置换集**可能是无限的！例如，如果知识库包括符号*Father*，那么可以构造无限多个嵌套项，如*Father*(*Father*(*Father*(*John*)))。我们的命题算法处理无限的语句集合时有困难。

$$\forall x \ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$
$$King(John)$$
$$Greedy(John)$$
$$Greedy(Father(John))$$

- Jacques Herbrand（1930）针对这种情况提出了一个著名的定理，这就是如果某个语句被原始的一阶知识库蕴涵，则存在一个只涉及命题化知识库的*有限子集*的证明。

半可判定性

- 如果语句不被蕴涵，我们能否做出判断？对于一阶逻辑，答案是否定的。证明程序可以不断进行下去，生成越来越深的**嵌套项**，我们不知道它是陷在无望的循环中，还是快要证明出结果了。这很像图灵机的**停机问题**。Alan Turing（1936）和Alonzo Church（1936）用不同的方法证明了这种事件状态的必然性。*一阶逻辑的蕴涵问题是**半可判定的**——存在算法能够证明被蕴涵的语句，不存在算法否定不被蕴涵的语句。*

当蕴涵成立时，证明会停机给出结果。
当蕴涵不成立时，证明会陷入无望的循环中

合一与提升：直接在FOL中推理

假言推理规则

- 证明结论Evil(John)

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

两步，运用两条推理规则：

1. 全称量词实例化推理规则
2. (命题逻辑的)假言推理规则

假言推理规则（Modus Ponens，拉丁文）

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$

只要给定任何形式为$\alpha \Rightarrow \beta$和α的语句，就可以推导出语句β

一般化的假言推理规则

广义假言推理

- 一般化的假言推理规则将假言推理规则从命题逻辑提升到一阶逻辑

$$\frac{p_1', p_2', \cdots, p_n', \quad (p_1 \wedge p_2 \wedge \cdots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

- 代换θ使得

SUBST(θ, $p_1' \wedge \ldots \wedge p_n'$) = SUBST($\theta$, $p_1 \wedge \ldots \wedge p_n$)

找到使不同的逻辑表示变得相同的代换

$\forall x$ *King*(*x*) $\wedge$ *Greedy*(*x*) $\Rightarrow$ *Evil*(*x*)
King(*John*)
Greedy(*John*)
Brother(*Richard*, *John*)

$$\frac{p_1', p_2', \cdots, p_n', (p_1 \wedge p_2 \wedge \cdots \wedge p_n \Rightarrow q)}{SUBST(\theta, q)}$$

Generalized Modus Ponens

- 例子：

  p_1' is *King*(*John*)　　p_1 is *King*(*x*)

  p_2' is *Greedy*(*John*)　　p_2 is *Greedy*(*x*)

  θ is {x/John,y/John}　　q is *Evil*(*x*)

  subst(q, θ) is *Evil*(*John*)

合一 Unification

合一：

- GMP推理规则中找到相关置换的过程。
- Unify(α,β) = θ if subst(α, θ)= subst(β,θ)

- 找到使不同的逻辑表示变得相同的代换的过程称为**合一**。**合一算法**Unify以两条语句为输入，如果存在合一代换则返回该**合一代换（合一元）**，否则返回失败。

UNIFY(p, q) = θ ，where SUBST(θ, p) = SUBST(θ, q)

UNIFY(*Knows*(*John*, *x*), *Knows*(*John*, *Jane*)) = {*x* / *Jane*}
UNIFY(*Knows*(*John*, *x*), *Knows*(*y*, *Bill*)) = {*x* / *Bill*, *y*/*John*}
UNIFY(*Knows*(*John*, *x*), *Knows*(*y*, *Mother*(*y*))) = { *y*/*John* , *x* / *Mother*(*John*)}

标准化分离

- Consider the sentence
  - UNIFY(Knows(John,x),Knows(x,Elizabeth)) = Fail
    - This fails because *x* cannot take on two values
    - But “Everyone knows Elizabeth” and it should not fail
  - Must **standardize** apart one of the two sentences to eliminate reuse of variable ()
    - UNIFY (Knows(John,x), Knows(x1,Elizabeth))

p	q	θ
Knows (*John*, *y*)	*Knows* (*John*, *Jane*)	{*x*/*Jane*}
Knows (*John*, *x*)	*Knows* (*y*, *OJ*)	{*x*/*OJ*, *y*/*John*}
Knows (*John*, x)	*Knows* (*y*, *Mother* (*y*))	{*y*/*John*, *x*/*Mother* (*John*)}
Knows (*John*, *x*)	*Knows* (*x*, *OJ*)	*fail*
Knows (*John*, *x*)	*Knows* (*y*, *z*)	{*y*/*John*, *x*/*z*}

标准化分离避免了变量重复使用，例如重新命名 *q*: *Knows* (z_1, *OJ*)，则可得到合一

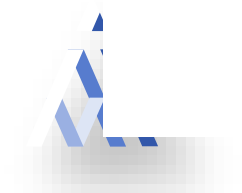
{*x*/*OJ*, z_1/*John*}

最一般合一代换MGU

- 合一代换可能不唯一
  - UNIFY (Knows(John,x), Knows(y,z))
  - {y/John, x/z} or {x/John, y/John, z/John}
  - Knows (John, z) or Knows (John, John)

  - First unifier is more general than second because it places fewer restrictions on the values of the variables

- There is a *most general unifier (MGU)* for every unifiable pair of expressions



合一算法

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

  if θ = failure **then return** failure
  else if $x = y$ **then return** θ
  else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP, θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(x.REST, y.REST, UNIFY(x.FIRST, y.FIRST, θ))
  else return failure

---

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

  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

知识库举例

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.

证明： Col. West is a criminal

… it is a crime for an American to sell weapons to hostile nations:

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

Nono … has some missiles

即，$\exists x\ Owns(Nono, x) \wedge Missile(x)$:

(2) $Owns(Nono, M_1)$

(3) $Missile(M_1)$

… all of its missiles were sold to it by Colonel West

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

Missiles are weapons：

(5) *Missile* (*x*) ⇒ *Weapon* (*x*)

An enemy of America counts as “hostile”：

(6) *Enemy* (*x*, *America*) ⇒ *Hostile* (*x*)

West, who is American …

(7) *American* (*West*)

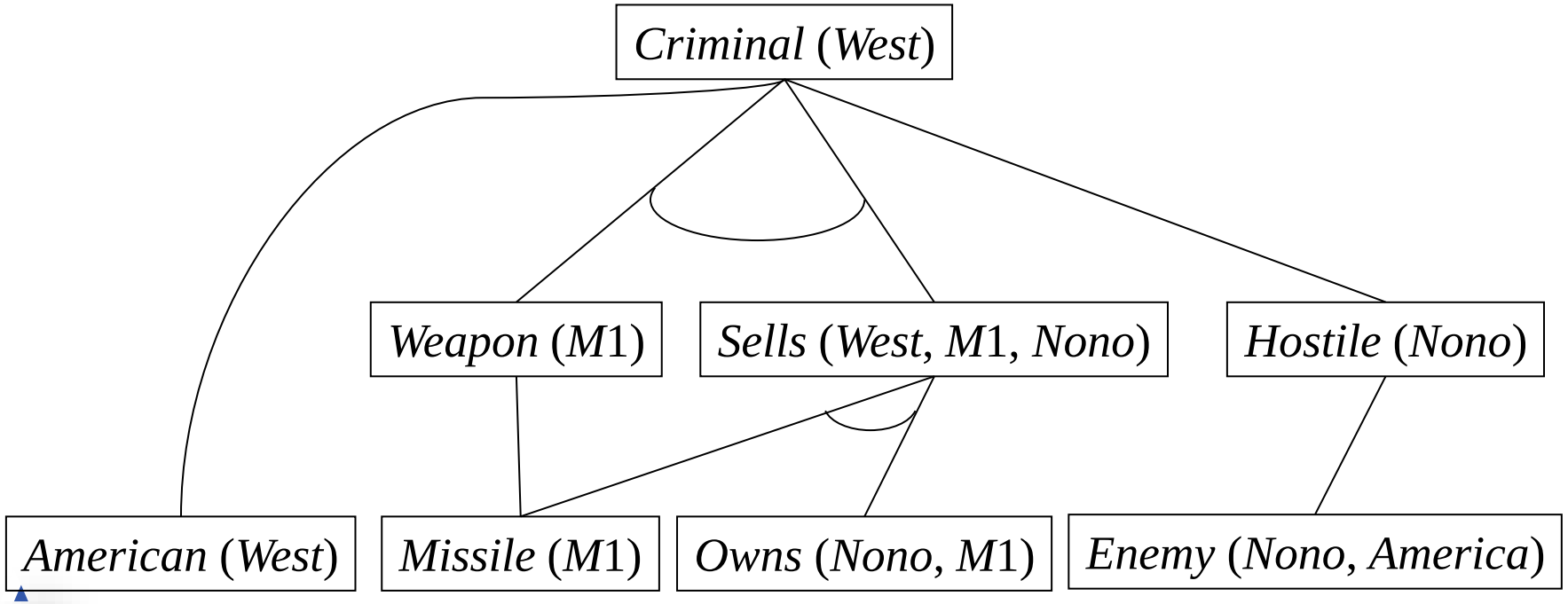
The county Nono, an enemy of America …

(8) *Enemy* (*Nono*, *America*)

American(x)∧Weapon(y)∧Sells(x,y,z)∧Hostile(z) ⇒ Criminal(x);

Missile(x)∧Owns(Nono,x) ⇒ Sells(West,x,Nono);

Missile(x) => Weapon(x); Enemy(x,America) ⇒ Hostile(x)



前向链的性质

对于定子句FOL KB来说前向链推理是合理且完备的

对于只包含确定子句不包含函数关系的FOL KB，假设

k：表示谓词的最大参数个数；

n：表示常量符号的个数；

p：表示谓词的数量，

则到达推理不动点的迭代次数最多为$p \times n^k$，即只需多项式级的迭代次数

对于包含函数关系的FOL KB，可能不会到达推理终点

命题逻辑的限定子句与前向链接

- **限定子句**：正好只有一个正文字的子句。
  - 它可以是原子语句；
  - 或是蕴含语句，蕴含语句的前提是正文字的合取，结论是单个正文字。

- 命题逻辑中，建立在限定子句上的**前向链接算法**：从知识库中的原子语句出发，在前向推理中应用**假言推理规则**，增加新的原子语句，直到不能进行任何推理。
- 如何应用于一阶限定子句，如何高效地实现。

一阶限定子句

- **一阶限定子句**中，假设变量是全称量化的，书写时省略全称量词
  - King(x) $\wedge$ Greedy(x)$\Rightarrow$ Evil(x)
  - King(John)
  - Greedy(y)
- 有些知识库可写成限定子句的集合，但并非每个知识库都可写为限定子句的集合(单一正文字的限制过于严格)。

前向链的效率

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

根据事实对合取前项进行合一存在合取排序问题，这是NP-hard问题

采用增量前向链。如果一条规则在$k-1$轮没有增加前项，则第k轮不进行匹配，只匹配刚刚获得新前项的那些规则

前向链会产生与目标无关的事实

前向链接产生**所有**基于已知事实的容许的推理，即使它们与要达到的目标无关

前向链广泛应用于数据库演绎

前向链接算法中可能的复杂性来源

- 算法的“内循环”涉及寻找所有可能的合一者，把规则的前提与知识库中一个适合的事实集进行合一（模式匹配），成本可能很高
- 算法每次遍历都要对每条规则进行重新检查，即使每次遍历添加到知识库的规则很少
- 算法可能生成许多与目标无关的事实

对于已知事实的规则匹配

- 例如，Missle(x) ⇒Weapon(x)，需要找出所有能和Missle(x) 合一的事实
  - 在已经建立索引的知识库中，上述过程的时间复杂度为O(1)
- 寻找一个**循序**来解决规则前提的合取，以使得总成本最少。
  - 例如： Owns(Nono,x) ∧ Missile(x) ⇒ Sells(West,x,Nono)，如果知识库包含很多Nono拥有的对象，但包含较少的Missile，较优的办法是先找出所有能和Missle(x) 合一的事实
- 寻找最优排序是个NP难题，但有一些可用的优秀启发式。

增量前向链接

Missile(x) ⇒ Weapon(x) 与 *Missile(M1)*

- 每个第t次迭代推理出来的新事实应该由至少一个第t-1次迭代推理出来的新事实导出，那么多余的规则匹配就可以避免。
- No need to match a rule on iteration *k* if a premise wasn't added on iteration *k-1*

⇒网（Rete）算法

无关的事实：解决办法

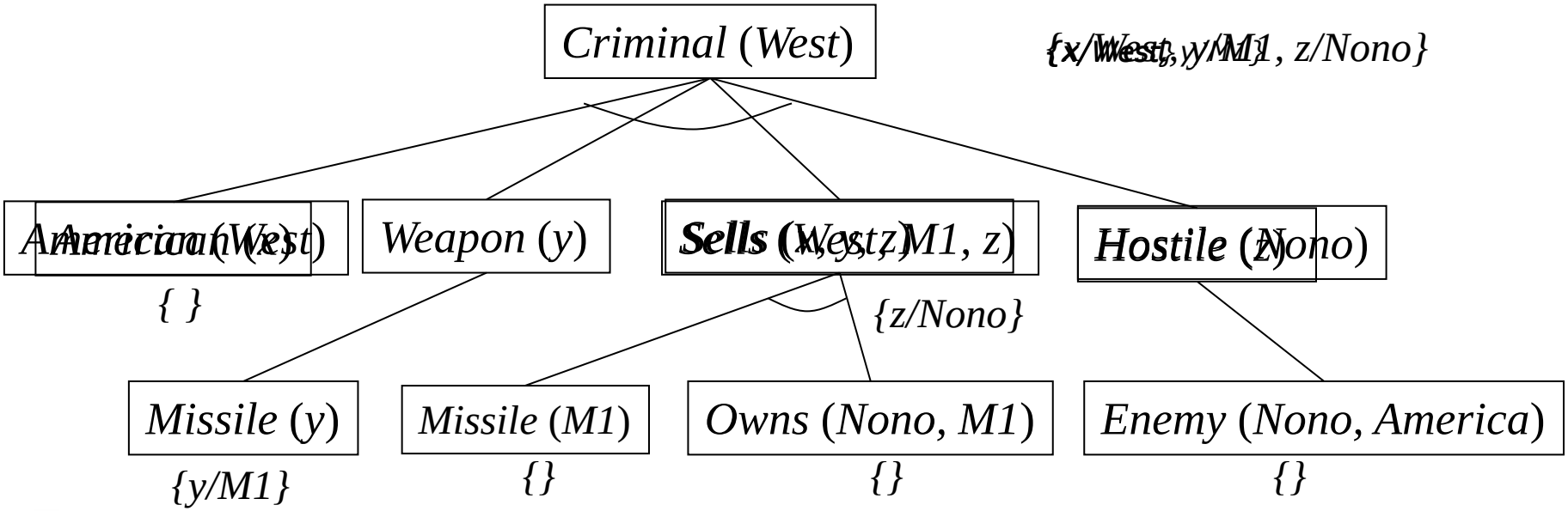
- 采用反向链接
- 把前向链接限制在一个选定的规则子集内
- 在演绎数据库内，利用从目标得到的信息重写规则集，从而在向前推理过程中只考虑相关的变量绑定。（魔法集方法，介于前向推理和反向预处理之间）

反向链证明

American(x)∧Weapon(y)∧Sells(x,y,z)∧Hostile(z) => Criminal(x);

Missile(x)∧Owns(Nono,x) => Sells(West,x,Nono);

Missile(x) => Weapon(x); Enemy(x,America) => Hostile(x)



Backward chaining algorithm

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

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

算法采用了置换的合成，

置换COMPOSE(θ1, θ2)与依次分别应用每个置换的效果相同，

即

SUBST(COMPOSE(θ1, θ2), p) = SUBST(θ2, SUBST(θ1, p))

反向链的性质

- 深度优先搜索
- 由于无限循环而导致不完备
  - 通过检测当前目标是否在堆栈中来弥补
- 由于反复重复子目标导致效率低下（无论成功或失败）
  - 通过保存先前的结果来弥补（额外空间）
- 广泛应用于**逻辑推理**

假设知识库为：

$\forall x \quad P(x) \Rightarrow Q(x)$

$\forall x \quad \neg P(x) \Rightarrow R(x)$

$\forall x \quad Q(x) \Rightarrow S(x)$

$\forall x \quad R(x) \Rightarrow S(x)$

要证明：

$S(A)$

对于FOL KB来说，广义假言推理是不完备的

归结推理规则

$$\frac{l_1 \vee \cdots \vee l_k, \quad m_1 \vee \cdots \vee m_n}{(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

其中 $UNIFY(l_i, \neg m_j) = \theta$ 。

例如：$$\frac{\neg Rich(x) \vee Unhappy(x), \quad Rich(Ken)}{Unhappy(Ken)}$$

采用 $\theta = \{x / Ken\}$

将归结应用于 $CNF(KB \wedge \neg \alpha)$ ，对于FOL是完备的

转换为合取标准型(CNF)

“Everyone who loves all animals is loved by someone”:

$$\forall x \quad [\forall y \quad Animal(y) \quad \Rightarrow \quad Loves(x,y)] \quad \Rightarrow \quad [\exists y \quad Loves(y,x)]$$

1. 消除等价和蕴含符号

$$\forall x \quad [\neg \forall y \quad \neg Animal(y) \vee Loves(x,y)] \vee [\exists y \quad Loves(y,x)]$$

2. 缩小¬的辖域:

$$\neg \forall x, p \equiv \exists x \quad \neg p; \quad \neg \exists x, p \equiv \forall x \quad \neg p$$

$$\forall x \quad [\exists y \quad \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y \quad Loves(y,x)]$$

$$\forall x \quad [\exists y \quad \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y \quad Loves(y,x)]$$

$$\forall x \quad [\exists y \quad Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y \quad Loves(y,x)]$$

转换为CNF（续）

3. 变量标准化： 对每个量词应该使用不同的变量

$$\forall x \quad [\exists y \quad Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z \quad Loves(z,x)]$$

4. 存在量词实例化：用Skolem函数取代处于某个全称量词辖域中的存在量词：

$$\forall x \quad [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$$

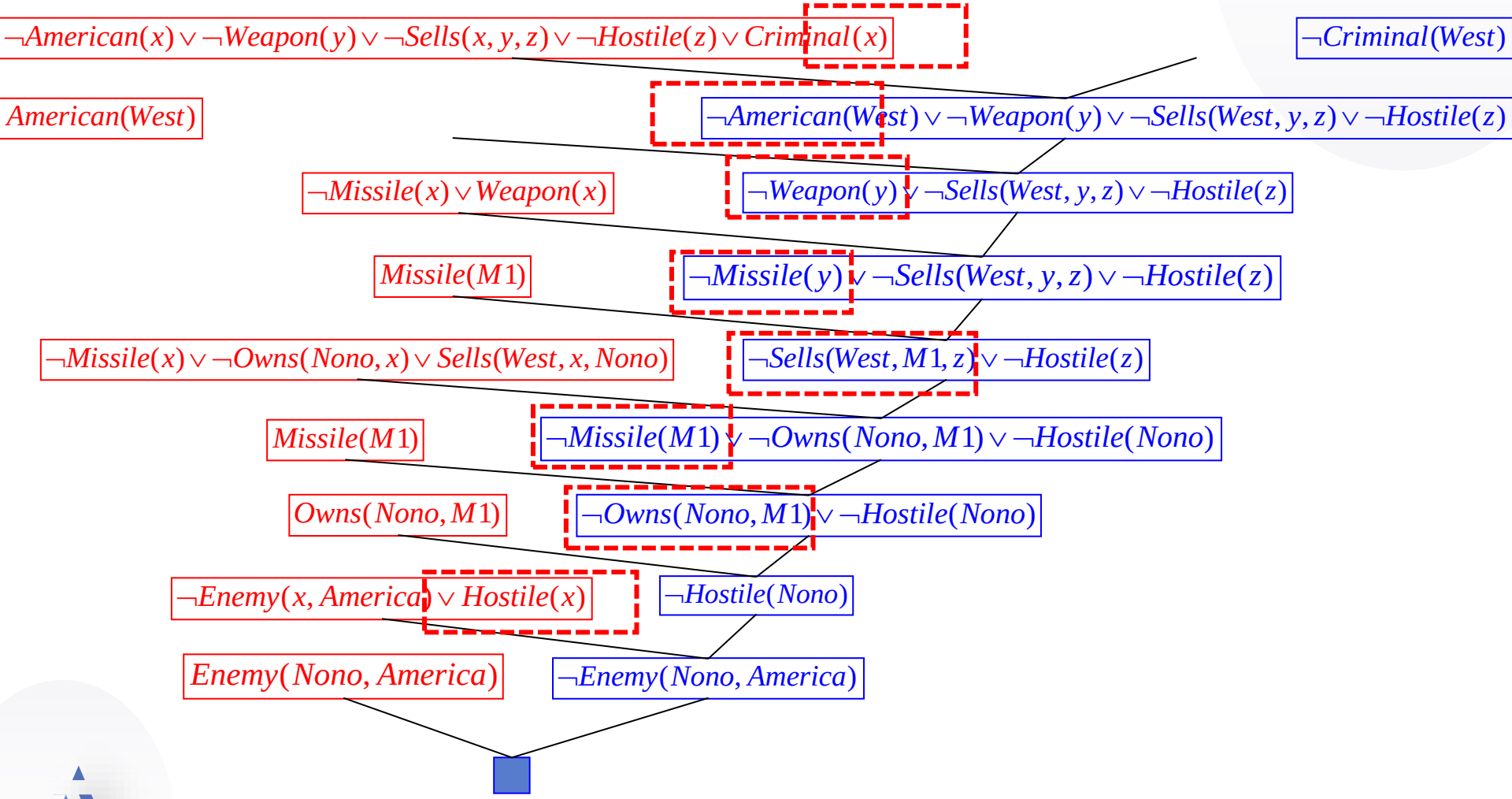
5. 去除全称量词：

$$[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$$

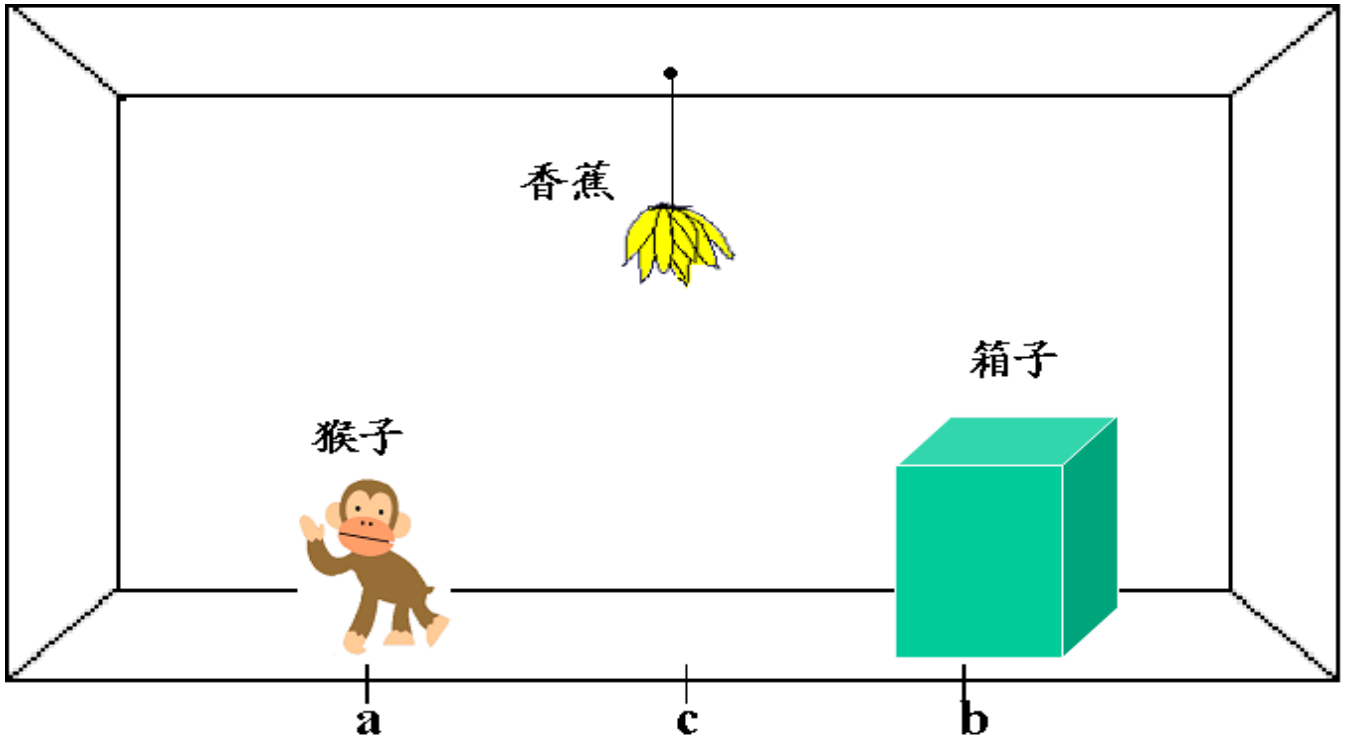
6. 转换为合取范式：

$$[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$$

归结推理证明


¬American(x) ∨ ¬Weapon(y) ∨ ¬Sells(x, y, z) ∨ ¬Hostile(z) ∨ Criminal(x)
¬Criminal(West)
American(West)
¬American(West) ∨ ¬Weapon(y) ∨ ¬Sells(West, y, z) ∨ ¬Hostile(z)
¬Missile(x) ∨ Weapon(x)
¬Weapon(y) ∨ ¬Sells(West, y, z) ∨ ¬Hostile(z)
Missile(M1)
¬Missile(y) ∨ ¬Sells(West, y, z) ∨ ¬Hostile(z)
¬Missile(x) ∨ ¬Owns(Nono, x) ∨ Sells(West, x, Nono)
¬Sells(West, M1, z) ∨ ¬Hostile(z)
Missile(M1)
¬Missile(M1) ∨ ¬Owns(Nono, M1) ∨ ¬Hostile(Nono)
Owns(Nono, M1)
¬Owns(Nono, M1) ∨ ¬Hostile(Nono)
¬Enemy(x, America) ∨ Hostile(x)
¬Hostile(Nono)
Enemy(Nono, America)
¬Enemy(Nono, America)

例：猴子摘香蕉问题



push(x,S)：在状态S下，猴子把箱子推到水平位置x

climb(S)：在状态S下，猴子爬上箱顶

grasp(S)：在状态S下，猴子摘到香蕉

问题的表示

已知：

1, $\neg ON(s_0)$

2, $(\forall x)(\forall s)(\neg ON(s) \Rightarrow AT(box, x, push(x, s)))$

3, $(\forall s)(ON(climb(s)))$

4, $(\forall s)((ON(s) \wedge AT(box, c, s)) \Rightarrow HB(grasp(s)))$

5, $(\forall x)(\forall s)(AT(box, x, s) \Rightarrow AT(box, x, climb(s)))$

求解：$(\exists s)HB(s)$　要证的结论

问题的子句集

- ✓ 去掉量词，取目标的非
- ✓ 子句变量标准化

1, $\neg ON(s_0)$

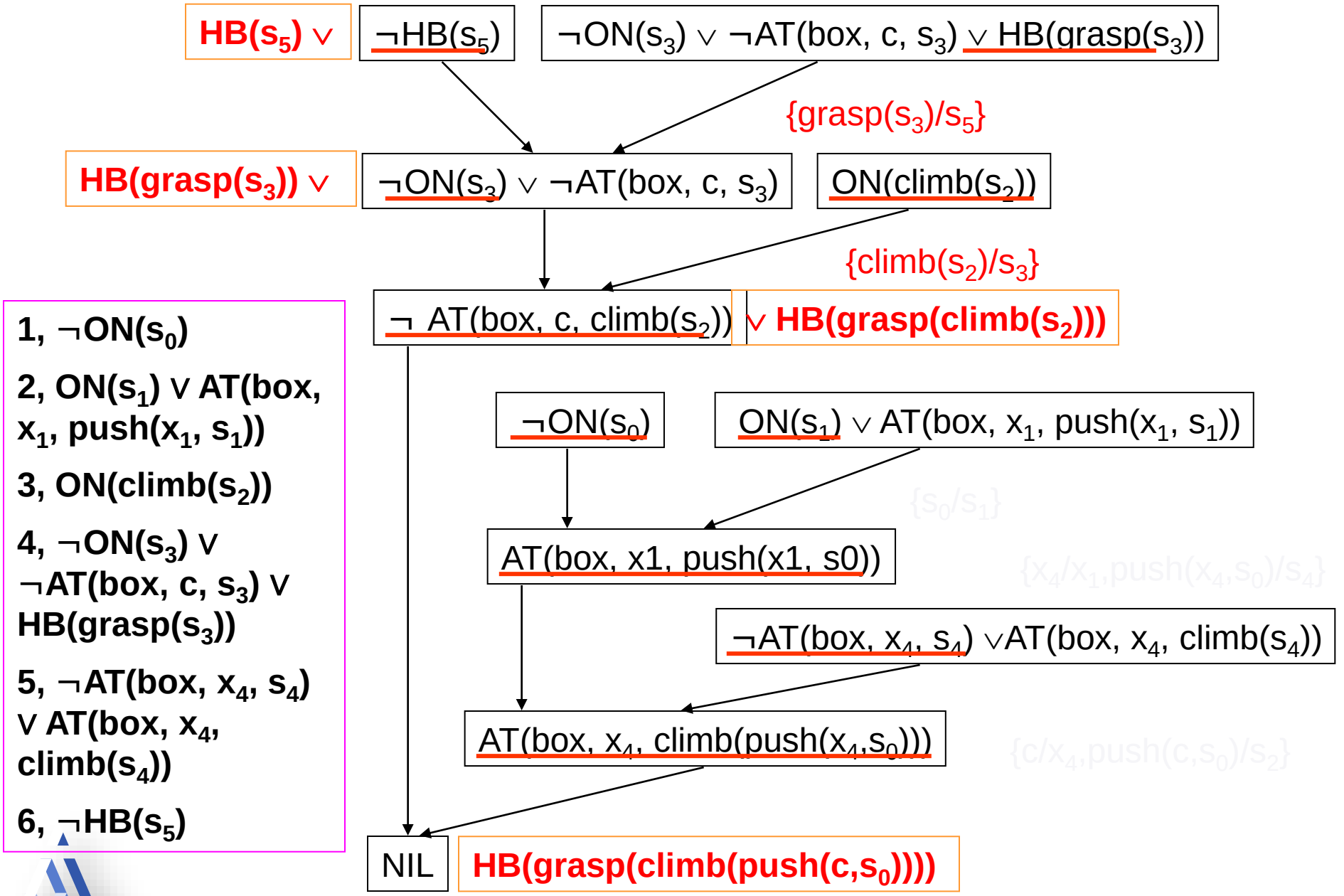
2, $ON(s_1) \vee AT(box, x_1, push(x_1, s_1))$

3, $ON(climb(s_2))$

4, $\neg ON(s_3) \vee \neg AT(box, c, s_3) \vee HB(grasp(s_3))$

5, $\neg AT(box, x_4, s_4) \vee AT(box, x_4, climb(s_4))$

6, $\neg HB(s_5)$


HB(s5) ∨
¬HB(s5)
¬ON(s3) ∨ ¬AT(box, c, s3) ∨ HB(grasp(s3))
{grasp(s3)/s5}
HB(grasp(s3)) ∨
¬ON(s3) ∨ ¬AT(box, c, s3)
ON(climb(s2))
{climb(s2)/s3}
¬ AT(box, c, climb(s2)) ∨ HB(grasp(climb(s2)))
1, ¬ON(s0)
2, ON(s1) ∨ AT(box, x1, push(x1, s1))
3, ON(climb(s2))
4, ¬ON(s3) ∨ ¬AT(box, c, s3) ∨ HB(grasp(s3))
5, ¬AT(box, x4, s4) ∨ AT(box, x4, climb(s4))
6, ¬HB(s5)
¬ON(s0)
ON(s1) ∨ AT(box, x1, push(x1, s1))
{s0/s1}
AT(box, x1, push(x1, s0))
{x4/x1,push(x4,s0)/s4}
¬AT(box, x4, s4) ∨AT(box, x4, climb(s4))
AT(box, x4, climb(push(x4,s0)))
{c/x4,push(c,s0)/s2}
NIL
HB(grasp(climb(push(c,s0))))

小结

- 知识表示语言应该是陈述性的、可合成的、有表达能力的、与上下文无关的，以及无歧义的。
- 一阶逻辑对于对象和关系的存在进行限定，因而获得比命题逻辑更强的表达能力
- 一阶逻辑的基本概念：对象、关系和函数
- 在一阶逻辑中开发知识库是一个细致的过程，包括对域进行分析、选择词汇表、对支持所需推理必不可少的公理进行编码。
- 一阶逻辑的命题化，运行速度很慢，产生大量和目标无关的句子。
- 合一消除了一阶逻辑证明的实例化步骤，从而提高了推理过程的效率。
- GMP利用合一提供了一条自然而又强大的推理规则。
  - GMP对于确定字句是完备的，但蕴涵问题是半可判定的。
- 前向链接用于演绎数据库和产生式系统中
- 逻辑程序设计系统，如Prolog中采用了反向链接

习题8.6,8.7

8.6 用一个没有矛盾的词汇表（需要你自己定义）在一阶逻辑中表示下列语句：

a. 某些学生在 2001 年春季学期上法语课。

b. 上法语课的每个学生都通过了考试。

c. 只有一个学生在 2001 年春季学期上希腊语课。

d. 希腊语课的最好成绩总是比法语课的最好成绩高。

e. 每个买保险的人都是聪明的。

f. 没有人会买昂贵的保险。

g. 有一个代理，他只卖保险给那些没有投保的人。

h. 镇上有一个理发师，他给所有不自己刮胡子的人刮胡子。

i. 在英国出生的人，如果其双亲都是英国公民或永久居住者，那么此人生来就是一个英国公民。

j. 在英国以外的地方出生的人，如果其双亲生来就是英国公民，那么此人血统上是一个英国公民。

k. 政治家可以一直愚弄某些人，也可以在某个时候愚弄所有人，但是他们无法一直愚弄所有的人。

8.7 用谓词演算表示语句“所有的德国人说相同的语言”。用 *Speaks*(*x*, *l*)表示人 *x* 说语言 *l*。

习题9.4,9.9

9.4 对于下列每对原子语句，请给出最一般合一者，如果存在的话：

a. $P(A, B, B), P(x, y, z)$

b. $Q(y, G(A, B)), Q(G(x, x), y)$

c. $Older(Father(y), y), Older(Father(x), John)$

d. $Knows(Father(y), y), Knows(x, x)$

9.9 写出下列语句的逻辑表示，使得它们适合应用一般化分离规则：

a. 马、奶牛和猪都是哺乳动物。

b. 一匹马的后代是马。

c. Bluebeard 是一匹马。

d. Bluebeard 是 Charlie 的父亲。

e. 后代和双亲是逆关系。

f. 每个哺乳动物都有一个双亲。