Artificial Intelligence and Robotics Research 人工智能与机器人研究, 2019, 8(1), 7-16
Published Online February 2019 in Hans. http://www.hanspub.org/journal/airr
https://doi.org/10.12677/airr.2019.81002

Hans汉斯

Automatic Logic Reasoning in Artificial Intelligence

Dejun Qiu

School of Philosophy and Sociology, Lanzhou University, Lanzhou Gansu
Email: qiudjun@lzu.edu.cn

Received: Nov. 29th, 2018; accepted: Dec. 17th, 2018; published: Dec. 26th, 2018

Abstract

Human reasoning cannot be applied in artificial intelligence, thus making machines for automatic reasoning. This paper discusses the detailed steps of resolution principle, and makes it easier to master the reasoning methods.

Keywords

Resolution, Unification, Reasoning

人工智能中的自动逻辑推理

邱德钧

兰州大学哲学社会学院，甘肃 兰州
Email: qiudjun@lzu.edu.cn

收稿日期：2018年11月29日；录用日期：2018年12月17日；发布日期：2018年12月26日

摘　要

人类常用的推理方法不能直接在人工智能中应用从而实现机器的自动推理，本文论述了自动推理中的归结和合一算法的规则和详细步骤，达到使读者容易理解和掌握这一方法的目的，使更多人了解这一方法的作用和前景。

关键词

归结，合一，推理

文章引用: 邱德钧. 人工智能中的自动逻辑推理[J]. 人工智能与机器人研究, 2019, 8(1): 7-16.
DOI: 10.12677/airr.2019.81002

1. 引言

人类思维中严格的推理主要是两种，一种是使用真值表，一种是使用演绎规则。真值表方法，有两个特点，其一，是机械性；其二是，每一个要素，都需要用固定的对象表达出来，假设用 n 表示推理中一个命题中含有对象的数目，当 n 大到一定程度的时候，2^n 会是一个极其庞大的数目，这就限制了它在机器推理当中的应用。

思维中还可以使用演绎推理规则来进行推理，如常用的分离规则[1]，下面两个是利用分离规则的具体例子：

1、如果得了肺炎，那么就会发烧；某人得了肺炎，他一定会发烧。

2、如果得了肺炎，那么，就会出去旅游；某人得了肺炎，所以他会出去旅游。

1 是正确的推理，2 是错误的推理，这依赖于人类的经验。让机器从知识库中，学会人类的整体经验，显然存在很大的难度，即使一个库中包含着人类绝大多数经验，检索耗时也限制了其应用。因此这种方法也不适用于机器推理。

归结和合一算法的出现，为解决机器的自动推理提供了很好的方法并得到广泛应用，这一点早已为相关的研究者熟悉。但从事逻辑学研究的人员，往往限于依据罗素和怀特海在《数学原理》中构建的推理体系，不了解机器自动推理的进展。这在人工智能兴起的今天，显然是不满足时代要求的。更多的非专业人士其实也有一窥机器推理的需求，实际工作中，越来越需要构架起《数学原理》中的为人熟知的推理体系与人工智能中自动推理之间的桥梁，让更多的非人工智能领域的研究人员深入认识与了解该方法，这也是本文的主要目的。

1963 年，J. Alan Robinson 找到了在机器上实现逻辑推理的简单方法：著名的归结与合一(resolution and unification)算法[1]。归结原理实质上是一条简洁的推理规则。使用这一条规则，对一阶逻辑中的任一个恒真公式，都将是可证的。到了 1972 年，以 L. Wos 为代表建立了一个以归结方法为主的自动推理系统。这个系统对于解有限数学、线路设计、程序正确性验证及形式逻辑等领域中的疑难问题，提供了帮助。1997 年，Leitsch，Alexander 又在 The Resolution Calculus 中给出了总结。至此，该方法得以广泛应用。

2. 命题逻辑中的归结算法：

令 p，q，r，s…表达命题逻辑中的任意一个原子命题，通过命题逻辑的五个连接词 $\neg$，$\vee$，$\wedge$，$\rightarrow$，$\leftrightarrow$ 形成合式公式。令 A，B，C 为任一合式公式，利用以下规则：

(1) 等值式：

$$(A \leftrightarrow B) \leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$$

(2) 蕴析律：

$$(A \rightarrow B) \leftrightarrow \neg A \vee B$$

(3) 德・摩根律：

$$\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$$

[1] 分离规则是在给定前提 A→B,A 都为真时，得到结论 B 为真的规则。

$$\neg(A \wedge B) \leftrightarrow \neg A \vee \neg B$$

(4) 双重否定：

$$\neg\neg A \leftrightarrow A$$

(5) 分配律：

$$A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C)$$

$$A \wedge (B \vee C) \leftrightarrow (A \wedge B) \vee (A \wedge C)$$

可将任意一个合式公式改写为合取范式(CNF)或者析取范式(DNF)，其中的合取范式就是以下进行归结运算的基础。设已有 CNF： $p \wedge (\neg p \vee \neg q) \wedge (p \vee q)$[2]，归结的步骤如下：

1、将合取范式中的每个简单析取用集合表达，整个合取范式为集合的集合：$\{\{p\},\{\neg p,\neg q\},\{p,q\}\}$。

2、在不同的集合中寻找同一个命题字母及其否定，即 π 与 $\neg\pi$ 这样的互补对。如上面第一个集合里的 p 和第二个集合里的 $\neg p$ 就是互补对。

3、由两个集合减去互补对得到的并集，组成新的集合就是进行归结，集合 $\{p\}$,和集合 $\{\neg p,\neg q\}$ 归结为 $\{\neg q\}$。

4、归结持续到不能归结或出现空集为止[2]，继续将 $\{\neg q\}$ 和 $\{p,q\}$ 归结得到 $\{p\}$。

由此可见，归结算法处理命题逻辑推理，简单明了，尤其适合于计算机编程执行，这一方法使得人类第一次在机器上实现了复杂的逻辑推理。付出的代价也是巨大的，一个简单的推理，需要经过繁琐的步骤，因为得到范式本身就需要许多步骤；稍复杂的推理其步骤是非常庞杂的。

3. 一阶谓词逻辑中的归结算法

归结算法同样适用于谓词演算。但有两个难点，第一是谓词逻辑中多出来量词、谓词和个体词，增加了问题的复杂性；第二是 Skolem 函数引入，进一步提高了处理推理的难度。

我们先看一阶谓词演算公式要得到范式新增加的规则：

(6) ¬符号移至量词的辖域之内：

$$\neg\exists x Fx \leftrightarrow \forall x \neg Fx$$

$$\neg\forall x Fx \leftrightarrow \exists x \neg Fx$$

(7) 如果两个量词的约束变量名相同，则对其中一个变量改名，使得每个量词约束唯一变量。如可将 $\neg\forall x Fx \wedge \exists x \neg Fx$ 改为 $\neg\forall x Fx \wedge \exists y \neg Fy$。

(8) 量词前置形成前束范式，依据以下规则：若 x 在公式 A 中不出现，则

$$\forall x Fx \wedge A \leftrightarrow \forall x(Fx \wedge A)$$

$$\forall x Fx \vee A \leftrightarrow \forall x(Fx \vee A)$$

$$\exists x Fx \wedge A \leftrightarrow \exists x(Fx \wedge A)$$

$$\exists x Fx \vee A \leftrightarrow \exists x(Fx \vee A)$$

$$\forall x Fx \wedge \forall x Gx \leftrightarrow \forall x(Fx \wedge Gx)$$

$$\exists x Fx \vee \exists x Gx \leftrightarrow \exists x(Fx \vee Gx)$$

$$\forall x Fx \wedge \forall y Gy \leftrightarrow \forall x \forall y(Fx \wedge Gy)$$

[2] 原来的公式为 $\neg p \wedge (p \leftrightarrow q)$ [3]，此处省去具体求范式的步骤。

$$\exists xFx \wedge \exists yGy \leftrightarrow \exists x\exists y\left(Fx \wedge Gy\right)$$

$$\forall xFx \wedge \exists yGy \leftrightarrow \forall x\exists y\left(Fx \wedge Gy\right)$$

$$\exists xFx \wedge \forall yGy \leftrightarrow \exists x\forall y\left(Fx \wedge Gy\right)$$

末尾四个公式对析取也成立，这里不再列出。应用以上的(1)~(8)规则，可以求出任一一阶谓词逻辑公式的合取范式。例如，“小王所有的朋友都不喜欢快餐”用公式表达为：

$$\forall x\left(friend\left(x,Wang\right) \to \neg like\left(x,snack\right)\right)$$

注意，为了理解方便和与 prolog 语言统一，采用英语单词表达谓词。其范式为：

$$\forall x\left(\neg friend\left(x,Wang\right) \vee \neg like\left(x,snack\right)\right)$$

由于全称量词约束整个公式的所有变量，在一定的个体域内，依据一阶谓词逻辑的全称销去规则，可进一步得到：

$$\neg friend\left(x,Wang\right) \vee \neg like\left(x,snack\right)$$

这使我们得到了规则：

(9) 位于公式左端的全称量词全部可以销去。

如果公式中包含存在量词，在进行归结算法时需要进行 Skolem 化，以达到销去存在量词的目的。分为两种情况：第一，当个体域中有 k 个个体时 $\exists xFx$ 定义为：

$$F\left(x_1\right) \vee F\left(x_2\right) \vee \cdots \vee F\left(x_k\right)$$

当 $\exists xFx$ 为真时，$x_1, x_2, \cdots, x_k$ 中至少有一个 x_i 使得 $\exists xFx_i$ 为真，此时用一个公式中未出现的新个体常量 c，得到 $F(c)$可方便的替代了 $\exists xFx$，达到销去存在量词的目的。第二，考虑如下谓词公式：

$$\forall x\exists yF\left(x,y\right)$$

对于所有 x 都存在一个 y，使得 x 和 y 具有 F 关系，如“每个人都读一些书籍”：

$$\forall x\left(people\left(x\right) \to \exists y\left(books\left(y\right) \wedge read\left(x,y\right)\right)\right)$$

这时，存在量词不可为个体常量 c 替代，不论 c 指的是那些书，一旦用 c 替代，就意味着每个人都读同样的书籍。解决的办法是定义 Skolem 函数 f，来代替被存在量词约束的变量 y，函数 f 将每个 x 映射到他读的书籍：

$$f = book\_read\left(x\right)$$

x 是变量，能将每个 x 映射到它所对应的书籍。这样就用函数 f 将存在量词销去得到：

$$\forall x\left(people\left(x\right) \to \left(book\left(f\left(x\right)\right) \wedge read\left(x,f\left(x\right)\right)\right)\right)$$

Skolem 化的应用，形成了归结算法的新规则：

(10) 若谓词公式中有存在量词，对他进行 Skolem 化。

把(1)~(10)条规则，综合应用于句子“每个人都读一些书”：

$$\forall x\left(people\left(x\right) \to \exists y\left(books\left(y\right) \wedge read\left(x,y\right)\right)\right)$$

$$\forall x\left(\neg people\left(x\right) \vee \exists y\left(book\left(y\right) \wedge read\left(x,y\right)\right)\right)$$

$$\forall x\exists y\left(\neg people\left(x\right) \vee \left(book\left(x\right) \wedge read\left(x,y\right)\right)\right)$$

$$\forall x\left(\neg people\left(x\right)\vee\left(book\left(f\left(x\right)\right)\wedge read\left(x,f\left(x\right)\right)\right)\right)$$

$$\neg people\left(x\right)\vee\left(book\left(f\left(x\right)\wedge read\left(x,f\left(x\right)\right)\right)\right)$$

$$\left(\neg people\left(x\right)\vee book\left(f\left(x\right)\right)\right)\wedge\left(\neg people\left(x\right)\vee read\left(x,f\left(x\right)\right)\right)$$

这是规范的，机器能识别的谓词逻辑公式。到此，仍然没有实现机器对谓词逻辑推理的处理，还要引进新的算法－合一。

4. 合一

合一是一个替换个体词的集合。具体的推理，涉及不止一个语句，往往是多个句子形成一个推理。这些句子中可能有大量的个体常项和许多个体变量，如果不进行一定的替换，或者随机替换，就会使得计算数量庞大到不能付之于实际应用。

首先是匹配文字。在范式中的两个不同的简单析取或简单合取中的两个文字相匹配，而且其中一个是另外一个的否定(又称互补)，则该文字可以归结。具体的匹配条件是：

1、n 元谓词的名称和元：谓词名称完全一样，“book”和“books”是两个不同的谓词。谓词的元也必须一样，book(x, y)和 book(x, y, z)一个是二元谓词，一个是三元谓词。

2、两个常量的匹配：两个常量的每个字符一一匹配时二者匹配。如 mary 只能与 mary 匹配，而与 Mary 不匹配。

3、常量 c 与变量 x 的匹配：如果 x 没有赋值，则二者匹配；如果 x 已经赋值，则当且仅当 x 的赋值与 c 匹配时 c 与 x 才匹配。

4、未赋值的两个变量 x，y 总是匹配的，赋值后要看结果，可能匹配，也可能不匹配。

举例如下：mother(x, y)与 mothers(z, u)不匹配，谓词不同；up(x, y)与 up(x, y, z)不匹配，前者是二元谓词，后者是三元谓词；brother(x, y)与 brother(john, w)匹配，实际上是使用赋值 x/john，y/w；brother(x/robert, y)与 brother(john, z)不匹配，x 已经赋值给 robert，它与 john 是两个不同的个体常项。

其次，匹配和给变量赋值的过程，就是在进行合一运算。如上例子中 brother(x, y)与 brother(john, w)匹配，得到的集合{x/john, y/w}就称作合一。

通过一个具体例子，能很好说明机器在一阶谓词逻辑中的自动推理。

小王所有的朋友都不喜欢快餐。

每个不喜欢快餐的人都不吃方便面。

小星是小王的朋友。

小星喜欢方便面吗？

符号化为谓词公式：

$$\forall x\left(friend\left(x,wang\right)\rightarrow\exists y\left(snack\left(y\right)\wedge\neg like\left(x,y\right)\right)\right)$$

$$\forall x\exists y\left(snack\left(y\right)\wedge\neg like\left(x,y\right)\rightarrow\neg eat\left(x,noodles\right)\right)$$

$$friend\left(xing,wang\right)$$

将以上三个句子先求范式再化为机器识别的标准公式：

$$\forall x\left(\neg friend\left(x,wang\right)\vee\exists y\left(snack\left(y\right)\wedge\neg like\left(x,y\right)\right)\right)$$

$$\forall x\exists y\left(\neg friend\left(x,wang\right)\vee\left(snack\left(y\right)\wedge\neg like\left(x,y\right)\right)\right)$$

$$\forall x\left(\neg friend\left(x,wang\right)\vee\left(snack\left(f\left(x\right)\right)\wedge\neg like\left(x,f\left(x\right)\right)\right)\right)$$

$$\left(\neg friend\left(x,wang\right)\vee\left(snack\left(f\left(x\right)\right)\wedge\neg like\left(x,f\left(x\right)\right)\right)\right)$$

$$\left(\neg friend\left(x,wang\right)\vee snack\left(f\left(x\right)\right)\right)\wedge\left(\neg friend\left(x,wang\right)\vee\neg like\left(x,f\left(x\right)\right)\right)$$

$$\forall x\exists y\left(\neg\left(snack\left(y\right)\wedge\neg like\left(x,y\right)\right)\vee\neg eat\left(x,noodles\right)\right)$$

$$\forall x\exists y\left(\neg snack\left(y\right)\vee like\left(x,y\right)\vee\neg eat\left(x,noodles\right)\right)$$

$$\forall x\left(\neg snack\left(f\left(x\right)\right)\vee like\left(x,f\left(x\right)\right)\vee\neg eat\left(x,noodles\right)\right)$$

$$\left(\neg snack\left(f\left(x\right)\right)\vee like\left(x,f\left(x\right)\right)\vee\neg eat\left(x,noodles\right)\right)$$

用集合形式表达得到的公式：

1、$\left\{\neg friend\left(x,wang\right),snack\left(f\left(x\right)\right)\right\}$

2、$\left\{\neg friend\left(x,wang\right),\neg like\left(x,f\left(x\right)\right)\right\}$

3、$\left\{\neg snack\left(f\left(x\right)\right),like\left(x,f\left(x\right)\right),\neg eat\left(x,noodles\right)\right\}$

4、$\left\{friend\left(xing,wang\right)\right\}$

根据匹配规则，寻找互补的文字对，得到：

5、$\left\{snack\ f\left(xing\right)\right\}$ (1,4 归结，x/xing)

6、$\left\{\neg like\left(xing,f\left(xing\right)\right)\right\}$ (2,4 归结)

7、$\left\{like\left(xing,f\left(xing\right)\right),\neg eat\left(xing,noodles\right)\right\}$ (3,5 归结)

8、$\left\{\neg eat\left(xing,noodles\right)\right\}$ (6,7 归结)

这显然是正确的结论，“小星不吃方便面”。至此，通过繁琐的步骤，Robinson 实现了准确的自动推理，使命题逻辑公式和一阶谓词逻辑公式的推理自动的在机器上运行，Robinson 的贡献在人工智能发展中其地位如何高估都不为过。

5. 代码实现

最终目的是用机器代替人工，实现全自动化证明。而这是容易做到的：

```
#ifndef SUBSENTENCE_H
#define SUBSENTENCE_H

#include <string>
#include <set>
#include <algorithm>
#include <random>
#include <stdexcept>
#include <cctype>

namespace FormulaNamepace {

// 公式符号定义
```

```
const char EQ = '#';      // 存在量词符号
const char UQ = '@';      // 全称量词符号
const char IMPLICATION = '>';  // 蕴含符号
const char NEGATION = '~';     // 否定符号
const char CONJUNCTION = '&';// 合取符号
const char DISJUNCTION = '|';  // 析取符号

const char CONSTANT_ALPHA[] = { 'a', 'b', 'c', 'd', 'e',
                                'i', 'j', 'k'};

typedef std::string Formula;
typedef std::string Subsentence;
typedef std::set<Subsentence> SubsentenceSet;

bool IsConstantAlpha(char ch);

// 移除最外层的括号对
Formula& RemoveOuterBracket(Formula& f);

// 前向扫描，查找匹配的一对组件，并且返回后者所在位置
template<typename FwdIter, typename Compo>
FwdIter FindPairChar(FwdIter first, FwdIter last,
                                    Compo former, Compo latter)
{
   std::size_t pairCnt = 0;
   while(1) {
      if(first == last) return last;

      if(*first == former)
            ++pairCnt;
      else if(*first == latter)
            if(--pairCnt == 0)
                break;
      ++first;
   }
   return first;
}

// 查找谓词符号
template<typename FwdIter>
```

```
inline FwdIter FindPredicate(FwdIter first, FwdIter last)
{
    return std::find_if(first, last,
                        std::ptr_fun<int, int>(islower));
}

// 查找公式符号
template<typename FwdIter>
inline FwdIter FindFormula(FwdIter first, FwdIter last)
{
return std::find_if(first, last,
                    std::ptr_fun<int, int>(isupper));
}

// 查找量词符号
template<typename FwdIter>
inline FwdIter FindQuantifier(FwdIter first, FwdIter last)
{
    return std::find_if(first, last,
                        [] (const typename FwdIter::value_type& lhs)
                        { return lhs == EQ || lhs == UQ; }
                       );
}

template<typename Set>
char FindNewLowerAlpha(const Set& s)
{
    const size_t alphaCnt = 26;
    std::default_random_engine e;
    for(size_t i = 0; i < alphaCnt; ++i) {
        char ch = 'a' + e() % alphaCnt;
        if(s.find(ch) == s.end())
            return ch;
  }
  throw std::logic_error("no more new alpha can use");
}

inline bool IsConnector(char ch)
{
    return ch == CONJUNCTION || ch == DISJUNCTION;
```

```
}

Formula& ReplaceAlphaWithString(Formula& target,
                    char alpha, const Formula& str);

// 获取谓词串
Subsentence GetPredicate(
                    typename Subsentence::const_iterator first,
                    typename Subsentence::const_iterator last);

// 获取公式串
Subsentence GetFormula(
                    typename Subsentence::const_iterator first,
                    typename Subsentence::const_iterator last);

// 消去蕴含连接词
Formula& RemoveImplication(Formula& f);

// 将否定符号移到紧靠谓词的位置
Formula& MoveNegation(Formula& f);

// 对变元标准化
Formula& StandardizeValues(Formula& f);

// 化为前束范式
Formula& TransformToPNF(Formula& f);

// 化为 Skolem 标准型
Formula& TransformToSkolem(Formula& f);

// 消去存在量词
Formula& RemoveEQ(Formula& f);

// 消去全称量词
Formula& RemoveUQ(Formula& f);

// 消去合取符号，获得子句集
void ExtractSubsentence(SubsentenceSet& subset,
                                const Formula& f);
}
```

```
#endif // SUBSENTENCE_H
```

对这一方法也有不同的质疑，根源于这一算法的繁琐，有人批评这种自动逻辑推理，认为人类在生活中从不这样机械和繁琐地思考。明显这种批评很无力，人类也从不高速旋转运动，但机器的高速旋转却极大提高了生产力。机器不一定跟人类思维方法一样，它用自己的方式达到了和人们一样的推理结果，这正是归结算法在人工智能发展中的真正意义所在。

参考文献

[1] 敖友云. 基于谓词逻辑的归结原理研究[J]. 计算机科学与应用, 2011(1): 51-56.

[2] 陶景侃, 大学逻辑教程[M]. 兰州: 兰州大学出版社, 2001: 274.

[3] (美) Rob. Callan. 人工智能[M]. 北京: 电子工业出版社, 2002: 62.