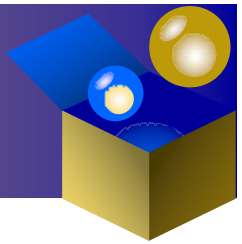
回答集求解中的"对称-破坏"方法

北京大学哲学系2009级逻辑学硕士生

王淑庆

2011年2月22日

Anil Nerode,Richard A.Shore

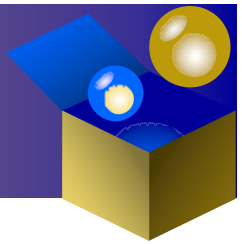


Since the 1970s,the winds of change have been blowing new seeds into the logic garden from computer science,AI and linguistics.

These winds have also uncovered a new topography with many prominences and depths,fertile soil for new logic subjects.

—— *Logic for Applications.*

本次报告的目的



- 介绍逻辑程序（**Logic Program**）的基本历史

  逻辑程序出现于上个世纪**70**年代初。

- 介绍逻辑编程（**Prolog，ASP**）的基本思想

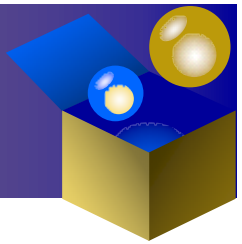
  Prolog=Programming in logic

  ASP=Answer Set Programming

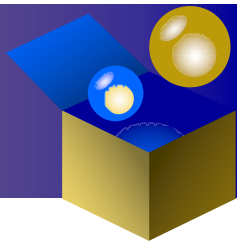
- 介绍**ASP**的一篇比较新的论文的基本思想

  Symmetry-breaking answer set solving（**Drescher,Tifrea,Walsh , August 10,2010**）。

目录



1，引言



1) 逻辑程序的历史发展

逻辑程序与归纳逻辑程序。

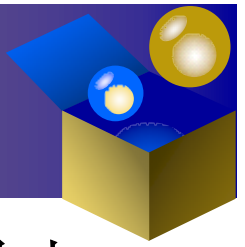
2) 非单调逻辑编程

非单调推理引入逻辑程序。

3) ASP的起源与现状

理论研究和应用研究。

1）逻辑程序的发展历史



逻辑程序早在1973左右就出现，当时即Prolog，它是由R.Kowalski,A.Colmerauer, R.Roussel等人开创的。逻辑程序近四十年的发展，主要体现在以下四个方面。

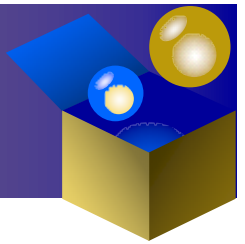
一是语形的发展变化。从最初的**Prolog**发展为带有两种否定的广义逻辑程序，近年来又发展为带有析取词和规则序的逻辑程序。

二是语义的细化与统一。逻辑程序的语义主要有模型论语义、不动点语义和执行语义等。由于语形的变化，语义产生了细化与分类。

三是逻辑程序语言和求解器的发展。求解原型系统主要有DLV，SMODELS，ASSAT。目前逻辑程序投身于商业开发的编程软件主要是Visual Prolog。

四是逻辑程序的应用研究不断发展。早期的逻辑程序主要用于专家系统，最近十来年它的应用越来越广泛。

1)逻辑程序的发展历史



逻辑程序

逻辑程序的发展主要体现在以下四个方面。

第一方面是逻辑程序语形的变化：**Horn**逻辑程序→广义逻辑程序→带有析取词和规则序的逻辑程序。

第二方面是逻辑程序语义的细化与统一。模型论语义、不动点语义和执行语义等。

第三方面是逻辑程序语言和求解系统的发展。

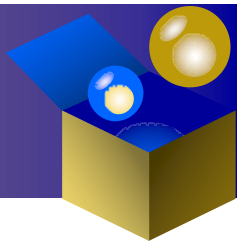
第四方面是逻辑程序的应用研究不断发展。

归纳逻辑程序

逻辑程序与机器学习的交叉研究领域，主要是研究机器学习中的归纳学习，使用的逻辑并不是归纳逻辑，仍然是逻辑程序中的子句逻辑。

1991年召开了第一届归纳逻辑程序设计国际会议，以后每年一次，促进了归纳逻辑程序的研究。

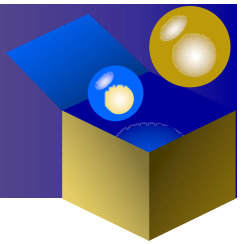
2）非单调逻辑编程



所谓非单调推理就是说随着前提集的增加而导致结论的修改，具体说来是：从$\Gamma \vdash \varphi$且$\Gamma \subseteq \Delta$，不能得到$\Delta \vdash \varphi$，甚至还可能得到$\Delta \vdash \neg\varphi$。非单调推理是常识推理的一个重要特征，而人工智能处理的推理很多是常识推理。非单调逻辑，主要有限定逻辑、缺省逻辑、自认知逻辑等。

最初Prolog不能处理非单调推理，直到1978年引入了"失败即否定"即缺省否定后才具有了处理非单调推理的功能。ASP引入了否定和析取结果推理能力，显然具有非单调推理的能力，从而使非单调逻辑从理论走向应用。

3) ASP的起源与研究现状

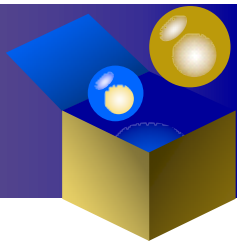


ASP的起源

ASP起源于1988年M.Gelfond和 V.Lifschitz为非单调编程创建的稳定模型语义，1991年在加入了缺省否定后，稳定模型语义改名为回答集语义。由此发展出一种重要的非单调逻辑编程技术，它是在融合了逻辑程序和非单调推理的基础上而发展起来的。

相对于过程式编程，回答集编程是一种陈述式编程。陈述式编程起源于一些用搜索算法来解答的问题，它的重心在于把逻辑描述告诉计算机，让计算机自动推理，而过程式编程则是告诉计算机每一步怎么做。搜索算法指这样一种算法，它把问题作为输入，对它的一系列可能的解答作评估之后选出某个回答作为输出。一个问题的所有可能的解答所形成的集合叫做它的搜索空间。

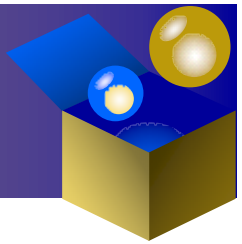
ASP的研究现状



1）理论方面：语法特性，计算复杂性，计算方法，与描述逻辑结合，偏好ASP语义，基于ASP的多语境系统，良基语义，权约束编程，分割理论，以及与各种非单调逻辑的关系等；

2）应用方面比较多：比较难的组合搜索、生物信息学、密码分析、自动化产品配置、数据库集成、诊断、决策支持、进化树推断、硬件设计、模型检测、AI规划、优先推理、语义网、航天飞机的高水平控制、自动音乐组合等。

2，Prolog基础



1）Prolog 的语法

Prolog所处理的推理其实就是一阶谓词逻辑的一个片断——Horn子句逻辑。

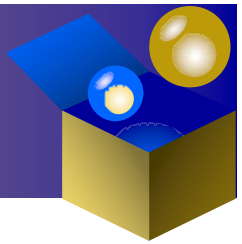
2）求解过程和语义

搜索、匹配(合一)与回溯。

3） 消解

消解， 各种加细消解， 线性消解。

1）Prolog 的语法



文字：指原子及其否定，分别叫做正文字和负文字。

子句：文字的有穷集合（析取）。如：$a \vee \neg b_1 \vee \cdots \vee \neg b_n$。

公式：子句的集合（合取，也即CNF）。

程序子句：恰含一个正文字的子句。表示为：a:-$b_1,\cdots,b_n$或a:-。

Horn子句：至多包含一个正文字的非空子句，故或者是程序子句，或者是目标子句。

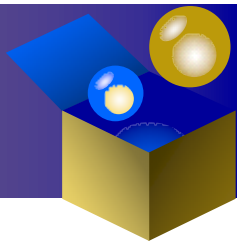
事实：不含负文字的程序子句。记为：p:-.

规则：含有负文字的程序子句。记为：a:- $b_1,\cdots,b_n$

询问：即目标子句，不含正文字的子句。记为：?- $b_1,\cdots,b_n$

Prolog程序：程序子句的集合，即只含规则和事实。

2）Prolog求解过程和语义



Prolog求解过程主要是通过搜索、匹配和回溯来完成。具体过程有些会比较复杂，下面仅通过一个简单的例子来说明这个过程。

例子：设一个Prolog程序由以下四条事实和两条规则组成，

f_1:on(b,a)；f_2:on(c,b)；f_3:on(f,c)；f_4:on(e,d)；

r_1:three_store(T,M,B):-on(T,M),on(M,B)；

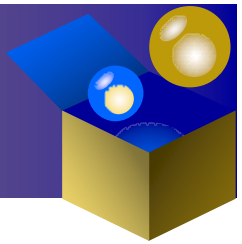
r_2:four_store(A,B,C,D):-three_store(A,B,C),three_store(B,C,D).

询问：?-three_store(Fst,Snd,Trd).

得到的答案是：solution1:Fst=c,Snd=b,Trd=a；

solution2:Fst=f,Snd=c,Trd=b.

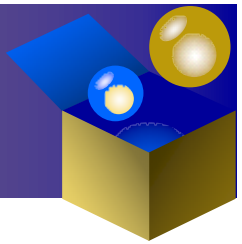
Prolog的基本思想



Prolog程序的基本思想是：给定一个由事实和规则组成的集合，我们希望知道一个公式φ是不是它的逻辑后承。具体来说就是，给定一个**Prolog**程序P，我们希望知道$q_1,q_2,\cdots,q_n$的合取是否是P的逻辑后承。也就是说，输入一个询问?-$q_1,q_2,\cdots,q_n$（即$\neg q_1 \vee \neg q_2 \vee \cdots \vee \neg q_n$），即添加子句G=$\{\neg q_1,\neg q_2,\cdots,\neg q_n\}$，然后问P∪{G}是否不可满足。如果不可满足，则是它的逻辑后承。

这个思想可由下面的引理保证行得通。

关于语义的定理

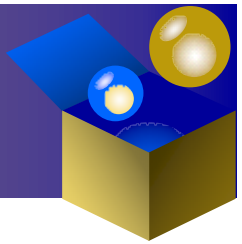


❖引理：若P是一个**Prolog**程序，G={$\neg q_1$，$\neg q_2$，…，$\neg q_n$}是目标子句，那么所有的q_i都是P的逻辑后承，当且仅当，P∪G不可满足。这其实就是说，$\Gamma \models \varphi$，当且仅当，$\Gamma \cup \{\neg\varphi\}$不可满足。

❖海布兰空间与库：若C是一个有限子句集，C的海布兰空间HU(C)是发生在其函数集F_C中的所有符号构成的项实例集。C的海布兰库HB(C)是根据HU(C)填充C直至饱和而得到的原子公式实例集。

❖斯柯伦范式：没有存在量词的-前束范式。∀-辖域里面一般化成合取范式。

海布兰模型



❖ 公式C的海布兰解释：C的一个海布兰解释I是HB(C)的一个子集。满足关系递归定义如下：

1)φ是原子公式实例p，I⊨φ当且仅当$p\in$I；

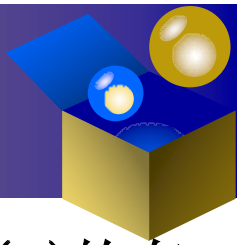
2)φ是否定文字实例$\neg p$，I⊨φ当且仅当$p\notin$I；

3)φ是子句实例$\boldsymbol{L}_1\vee\cdots\vee\boldsymbol{L}_n$，I⊨$\varphi$当且仅当至少有一个$\boldsymbol{L}_i$使I⊨$\boldsymbol{L}_i$；

4)φ是C中的一个子句$\boldsymbol{C}$，I⊨$\boldsymbol{C}$，当且仅当I⊨$\boldsymbol{C\sigma}$，其中任意实例$\boldsymbol{C\sigma}$是指通过用HU(C)中元素替代$\boldsymbol{C}$中的变元素获得的。

❖ 定理：如果M是公式C的一个一阶模型，则M'={$p\in$HB(C)|M⊨p}是C的一个海布兰模型。

3）消解



海布兰定理：C是不可满足的，当且仅当，至少存在一个HU(C)的有限子集，使得饱和填充实例Saturation(HU,C)是不可满足的。也即：任何不被满足的斯柯伦范式，总能找到有限个命题集是不可满足的。

Prolog和大多数自动定理证明器的证明方法都是消解(resolution)。

定义：如果分别从形如$\{l\}\cup C_1$，$\{l'\}\cup C_2$，推导$C=C_1\cup C_2$，那么C叫做C_1和C_2的消解式。其中，l和l'分别表示一个文字及其补文字，∪在这里特指不相交集合的并。

可以证明在Prolog里，消解反驳既是可靠的又是完全的。

前面所讲的消解要找出一整颗树，因此出现了各种加细消解，在各种加细消解中，线性消解实行起来效率最高。

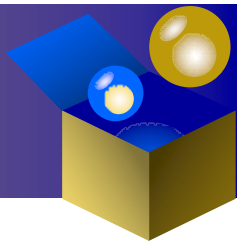
定义：（I）C的从S出发的线性消解证明是指满足$C=C_{n+1}$以及下列条件的有序对序列：$\langle C_0,B_0\rangle$，…，$\langle C_n,B_n\rangle$：

（1）C_0以及每个B_i，要么是S中的元素，要么是某个$j<i$的C_j；

（2）每个C_{i+1}($i\leqslant n$)是C_i和B_i的消解式。

（II）如果C是从S出发的线性可证的，记为$S\vdash_{\mathcal{L}}C$。若$S\vdash_{\mathcal{L}}\square$，则称存在S的一个线性反驳。

3，ASP基础



1）语法

在**Prolog**的基础上增加了两种否定和析取。

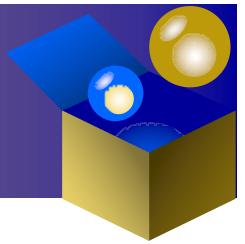
2）语义

即它的回答集语义，基于极小的海布兰模型。

3）求解过程

“生成-测试”法。

1）ASP的语法



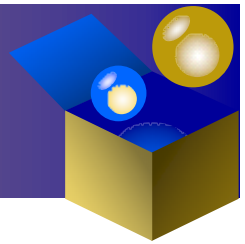
定义：原子集$\mathcal{A}$上的一个**ASP**程序是指一个有限的规则集，其中的规则r形如：

$$a_1 \vee \cdots \vee a_l \leftarrow b_1,\cdots,b_m,\ \sim c_1,\cdots,\sim c_n$$

其中a_i，b_j，c_k是文字(正文字或负文字，正文字即原子)，~是缺省否定，逗号表示合取。

←的左边称为规则的头部(head)，右边称为规则的躯干(body)，所以，r也可以表示为：head(r)←body$^+$(r), ~body$^-$(r)。r的直观意思是，如果有$b_1,\cdots,b_m,\ \sim c_1,\cdots,\sim c_n$，那么就有$a_1 \vee \cdots \vee a_l$。如果$l$=0，则称这条规则为一个约束，通常简写为:$\leftarrow b_1,\cdots,b_m,\sim c_1,\cdots,\sim c_n$ 。这样的一条约束规则排除了$b_1,\cdots,b_m$，或者使得$c_1,\cdots,c_n$ 中的某一个在模型中成立。如果P的规则r中的文字都是原子，则称P为正规逻辑程序；若再l>1，则称P是正规析取逻辑程序，否则是正规非析取逻辑程序。

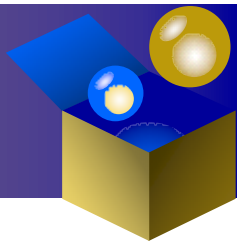
2）ASP的语义



ASP的语义是以回答集为模型。**AS**程序里的规则都是一阶逻辑中的公式，但是回答集语义的方法则是一种退化的方式，对任意一个程序P，首先构造它有ground(P)，也就是把规则里的变元用所有不含变元的项来代入，由此得到新的程序。

构造了ground(P)之后，再寻找满足这个新程序的文字——一个回答集中的元素都是文字，并且这个回答集满足新程序中的所有规则。这和命题逻辑中一个公式集的模型有点类似，但它就是海布兰模型，即论域中的元素是命题而不是个体。一般地说，程序P的一个模型S是集合{P的所有有基文字}的子集，它满足P的所有规则，它的回答集就被定义为是极小的那个海布兰模型。极小，指的是它的每个真子集都不是该程序的海布兰模型。

2）ASP的语义



为求一个ASP程序的回答集，需要G-L转换，它是AS语义的核心。

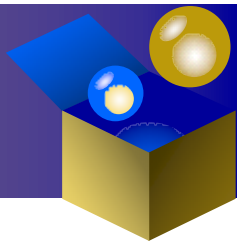
我们先只考虑正规非析取程序的G-L转换：最初的AS语义是针对正规非析取程序的，所以我们下面假定P是正规非析取程序。Ground(P)为P的填充实例，M是P的一个海布兰解释，对于Ground(P)的任意规则 r：$a_1 \leftarrow b_1,\cdots,b_m, \sim c_1,\cdots,\sim c_n$，G-L转换过程如下：

若$body^-(r) \cap M \neq \emptyset$（即至少包含了$r$否定体中的一个原子公式实例），则删除$r$；

否则，即$body^-(r) \cap M = \emptyset$，则将$r$改为$a_1 \leftarrow b_1,\cdots,b_m$（即删除$r$的否定体）。

经G-L转换逻辑程序是P相对于M的，故记作P^M，所以P^M其实就可以表示为 $P^M = \{a_1 \leftarrow body^+(r) \mid r \in P, body^-(r) \cap M = \emptyset\}$。显然，$P^M$是Horn程序。我们定义，如果M与$P^M$的最小H-模型相等，我们就说M是P的一个回答集。

2）求ASP的回答集的例子



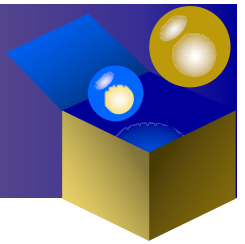
例1：求$\mathbf{P_1}=\{a\leftarrow\sim b;\ b\leftarrow\sim a\}$的回答集。

解：显然$A=\{a,\ b\}$，假设$M_1=\{a\}$，则$\mathbf{P^M}=\{a\leftarrow\}$，它的极小海布兰模型为$\{a\}$，故$M_1$是$P_1$的一个回答集。同理假设$M_2=\{b\}$，可以得到$M_2$也是$P_1$的一个回答集。

例2：求$P_2=\{p(1,2);\ q(x)\leftarrow p(x,y),\sim q(y)\}$的回答集。

解：第二条规则的代入实例为$\{q(1)\leftarrow p(1,1),\sim q(1);\ q(1)\leftarrow p(1,2),\sim q(2);\ q(2)\leftarrow p(2,1),\sim q(1);\ q(2)\leftarrow p(2,2),\sim q(2)\}$。那么，$A$={q(1),q(2),p(1,1),p(1,2),p(2,1),p(2,2)}。设M_1={q(2)}，则P^M={p(1,2)；q(1)←p(1,1)；q(2)←p(2,1)}，它的极小海布兰模型为{p(1,2)}≠M_1，故M_1不是P_2的一个回答集。再假设，M_2={p(1,2),q(1)}，则P^M={p(1,2)；q(1)←p(1,2)；q(2)←p(2,2)}，它的极小海布兰模型为{p(1,2),q(1)}=M_2，故M_2是P_2的一回答集。然而，我们知道，A的子集有2^6个，这样试下去效率是非常低的，好在计算机的速率比人脑快许多倍，并且事实上只需要验证A的部分子集。

3）ASP的求解过程

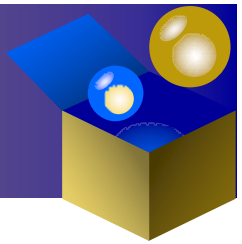


从上面两个例子可以看出，求AS程序的回答集，就是用“假设-测试”法去求的。我们再具体分析一下上面的例子2：首先，$p(1,2)$肯定要在回答集里面；其次，不能有$p(1,1), p(2,1), p(2,2)$；再次，剩下还有三种情况，即$\{p(1,2), q(2)\}$，$\{p(1,2), q(1), q(2)\}$，$\{p(1,2)\}$，经测试，它们都不是回答集。

但是，只要基公式稍微多一点，回答集的求解过程就会比较麻烦。因此，如何降低它的求解的复杂度，就是一项非常重要的工作。

与Prolog里的消解不同的是，ASP中的求解算法总是可以停止的。

4，对称-破坏



1）基本思想

通过研究逻辑程序对称发现和对称破坏来抛弃搜索空间的对称部分，从而简化求解过程。

2）对称发现

把对称发现归约为求图自同构问题。

3）对称破坏

通过对称发现后，再增加对称破坏约束，这样就可以应用到任何存在的回答集求解器中。

4）实验结果举例

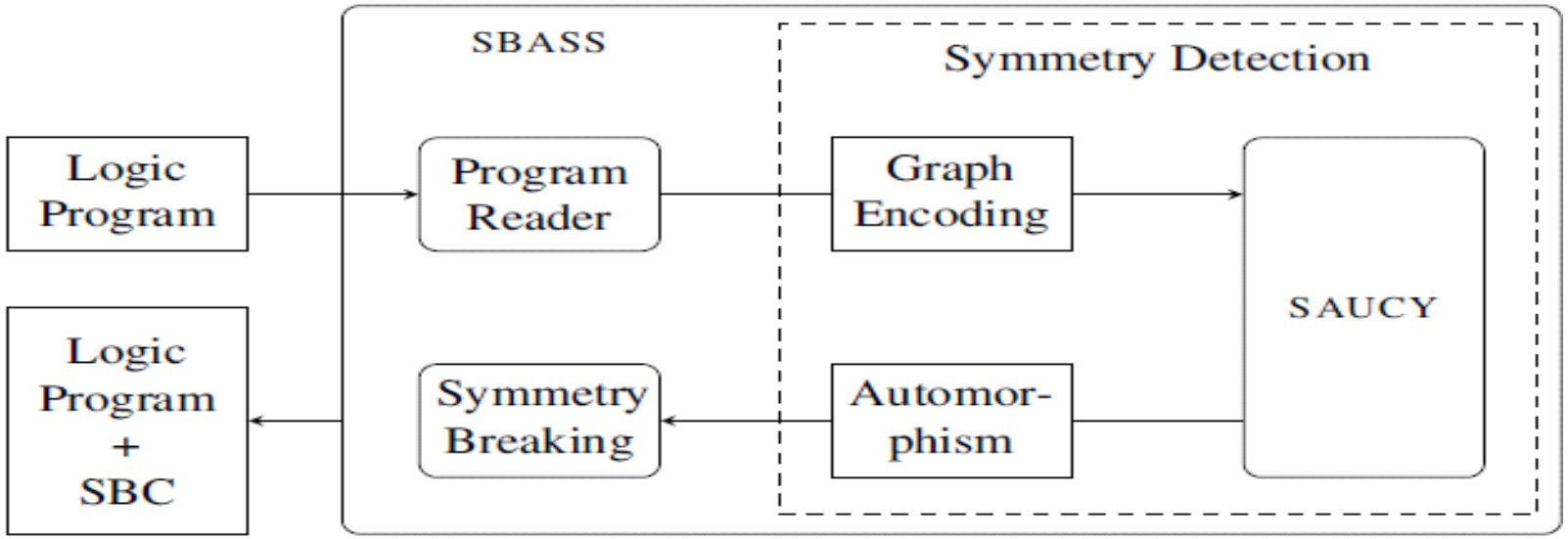
1）基本思想

通过对逻辑程序的原子进行置换，发现有些置换不会影响程序本身（即这个置换就是对称），这正如图自同构问题。因此把逻辑程序的对称发现化归为图自同构问题就是一件很自然的想法。

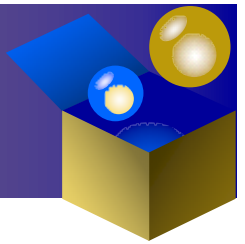
虽然一般的图自构问题是个尚未解决的问题，但对于具体的不是太复杂的图，它的自同构问题可以用**SAUCY**系统比较方便地求出。这就是要研究逻辑程序的对称发现的一个理由。

找到一个逻辑程序的对称并不是为了对称本身，而是为了由对称而建立的对称破坏，从而简化求逻辑程序的回答集搜索空间，以达到简化求它的回答集的目的。

从对称发现到对称破坏，具体实施是在**SBASS**系统中实现的，过程图如下：



2）对称发现



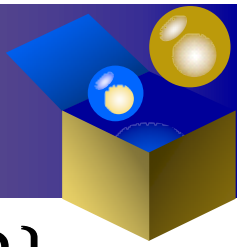
定义1：集合S的一个置换是一个双射：S→S。

定义2：有向图G=(V,E)，V是顶点集，E⊆V×V是边的集合，且至少有一条边是有向的。G的自同构或对称就是指它的顶点的置换，使得有向边对有向边，非有向边对非有向边，且保持边的方向性。

定义3：图G=(V,E)的一个自同构或对称是一个置换$\pi \in \prod(V)$，使得$(u,v)\in E$，当且仅当，$(u,v)^{\pi}\in E$。

定义4：图的自同构问题，就是说，给定一个图，如何找出它的所有的自同构图，也就是如何找到它的所有对称。

对称发现



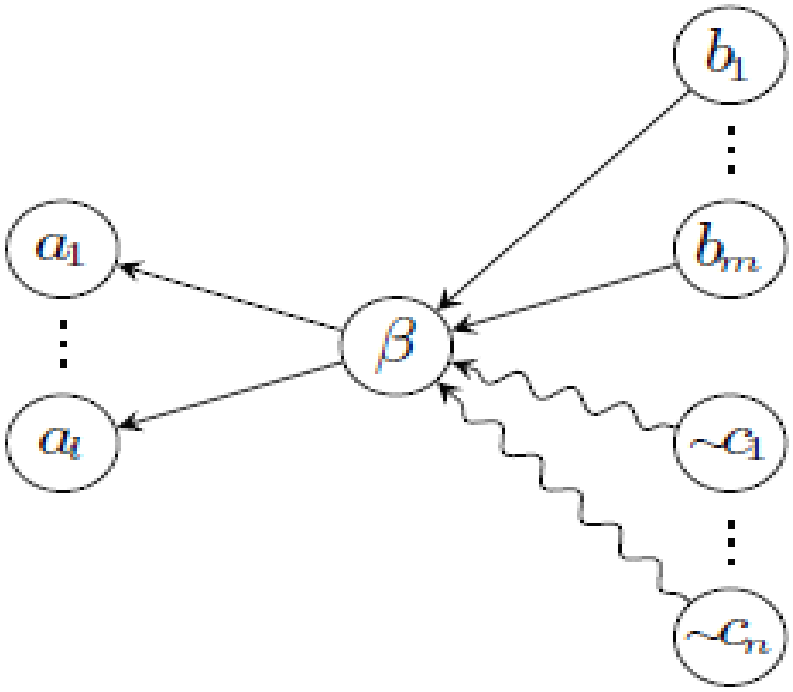
定义5：逻辑程序P的一个置换是指$P^{\pi}=\{r^{\pi}|r\in P\}$，$\pi\in\prod(atom(P))$，$r^{\pi}=a_1^{\pi}\vee\cdots\vee a_l^{\pi}\leftarrow b_1^{\pi},\ldots,b_m^{\pi},\sim c_1^{\pi},\ldots,\sim c_n^{\pi}$。

定义6：逻辑程序P的一个对称是指它的一个置换$\pi\in\prod(atom(P))$，使$\pi(P)=P$。因此，P的对称保持它的回答集不变。例子：如果一个逻辑程序$\mathbf{P}=\{a\leftarrow\sim b;\ b\leftarrow\sim a\}$，显然它的一个对称为$\pi=(a,b)$，因为$\pi(P)=P$。

注意：可以证明，给定一个逻辑程序P的一个回答集，所有通过P的对称所得到的集合也一定是P的回答集；类似地，对称也使得非回答集对应非回答集。这个规律是后面我们所研究的对称发现与对称破坏的意义的基础。另外，表现出对称性的逻辑程序其实是某类逻辑程序，比如组合搜索问题类的逻辑程序。

躯干-原子图

定义：一个逻辑程序的躯干-原子图$G=(V,E_0 \cup E_1,E_2)$，其中，$V=body(P) \cup atom(P)$,$E_0=\{(\beta,\alpha)|\alpha \in head(P),\beta \in body(\alpha)\}$，$E_1=\{(\alpha,\beta)|\beta \in body(P),\alpha \in \beta^+\}$，$E_2=\{(\alpha,\beta)|\beta \in body(P),\alpha \in \beta^-\}$。如下图：



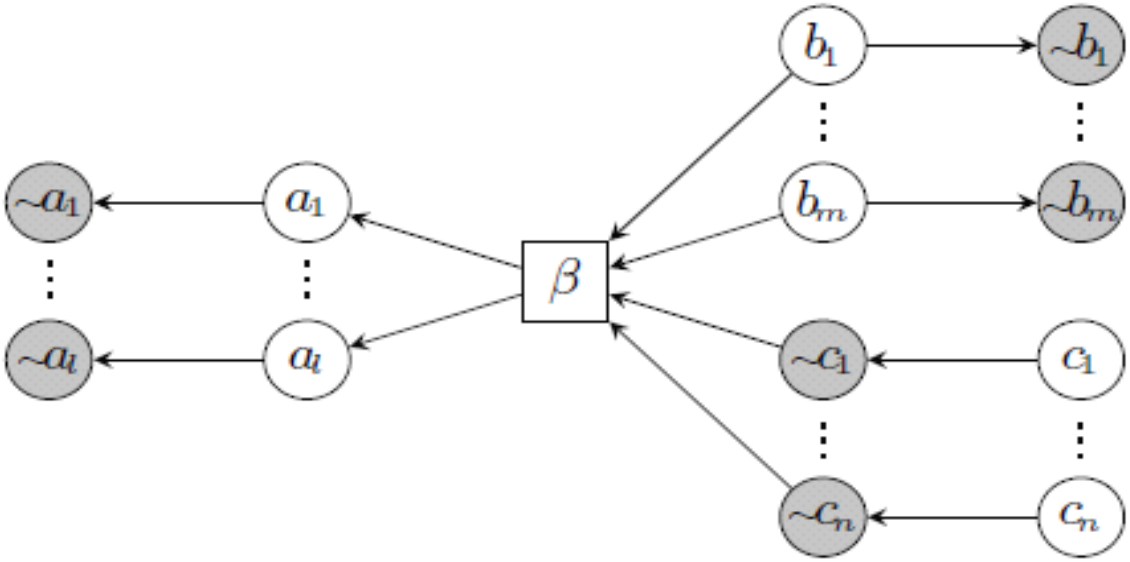
躯干-原子图转换为三色有向图

把一个躯干-原子图转换为三色有向图，有以下三个步骤：

1，分别用颜色1和颜色2表示正文字和缺省文字；

2，用颜色3表示躯干顶点，所以一条规则就是躯干文字连接躯干顶点，而且躯干顶点连接头的原子；

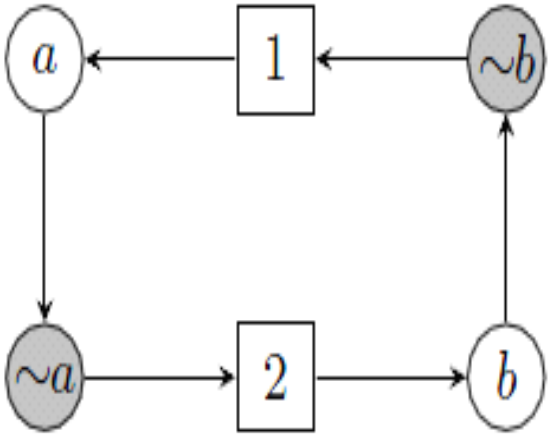
3，为保持一致性，a映到b当且仅当$\sim a$映到$\sim b$，相反的文字通过正到负的有向边连接。



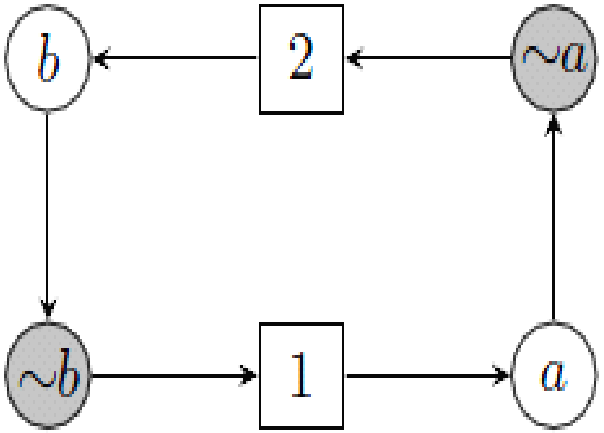
对称发现的例子

例子1：$P_1=\{a \leftarrow \sim b;\ b \leftarrow \sim a\}$

首先画出它的三色有向图如下：　　然后，找出它的一个对称后的三色有向图：



Original 3-coloured graph of P_1

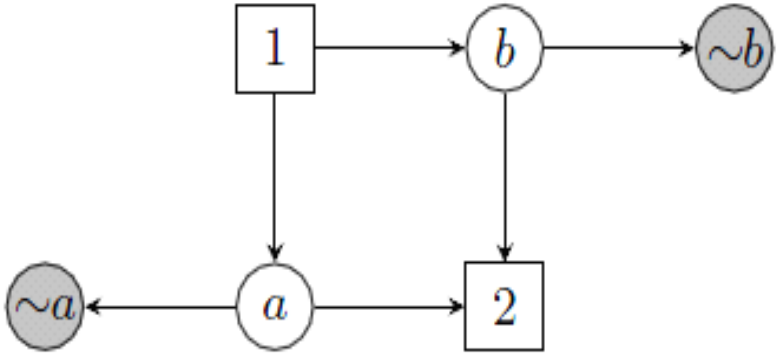


$\pi_1 = (a\ b)\ (\sim a \sim b)\ (1\ 2)$

对称发现的例子

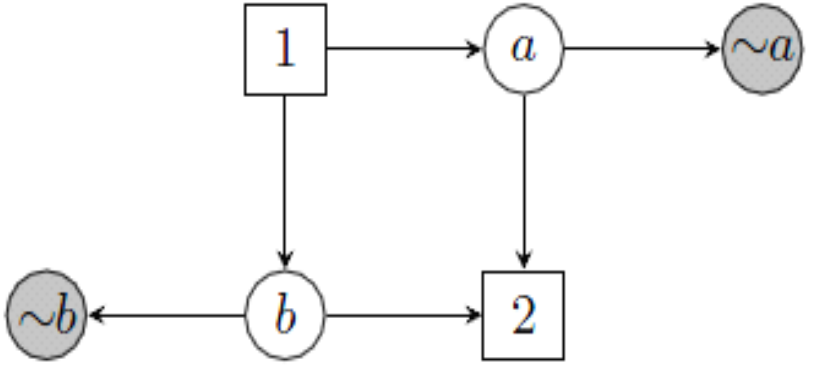
例子2：$P_2=\{a \vee b \leftarrow \ ; \ \leftarrow a,b\}$

首先画出它的三色有向图如下：



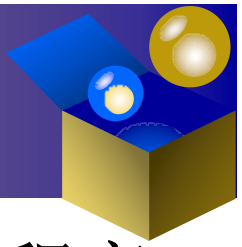
Original 3-coloured graph of P_2

然后画出它的一个对称后的三色有向图：



$\pi_2 = (a\ b)\ (\sim a \sim b)$

对称发现的定理



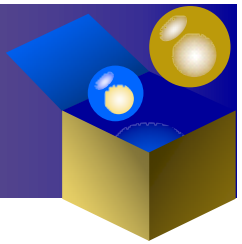
定理：一个回答集逻辑程序的所有对称与这个程序的三色有向图的所有对称是一一对应的。

证明略。

从逻辑程序的三色有向图的构造来看，这个定理应该是没有什么问题的。

这个定理说明了把逻辑程序的对称转化为有向三色图的自同构问题是合理和可行的。因为一般的图自同构问题可以用SAUCY系统求出来。

3）对称破坏



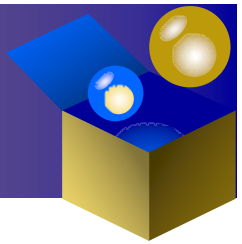
对称发现后，就需要在逻辑程序中加入非对称约束来实现它的对称破坏。

比如说，上面提到的P_1和P_2，它们的对称是一样的，那么我们就可以构造出下面的对称破坏约束SBC来：

$$\leftarrow b, \sim a$$

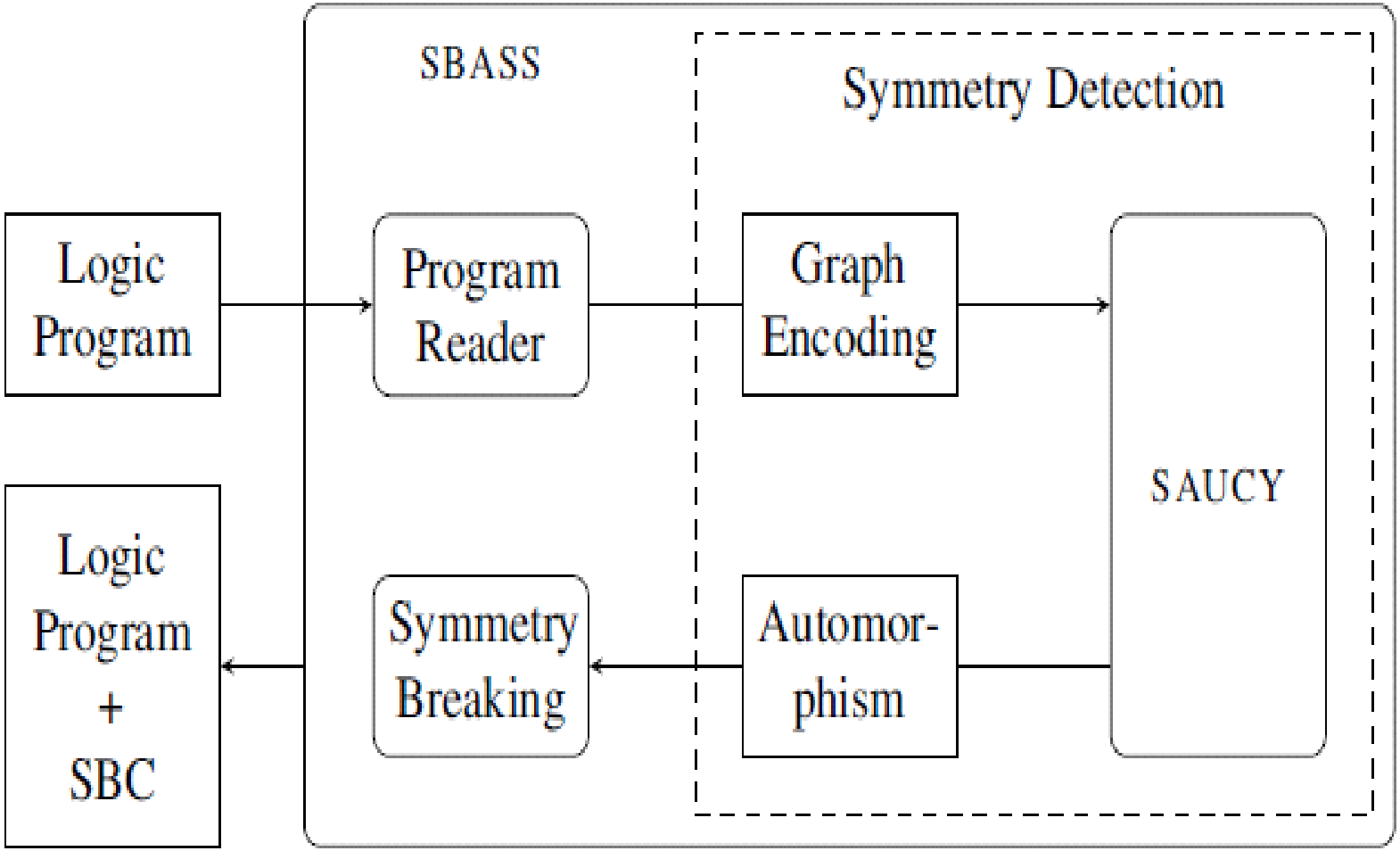
在这里有了SBC，a对b就有一种优先关系，当搜索到$\{a\}$并测试它为回答集，则不会再去搜索$\{b\}$是否为回答集了。

部分对称破坏

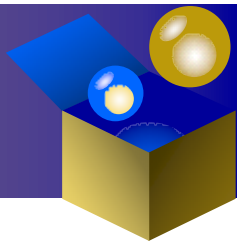


破坏一个逻辑程序的所有对称并不一定会加速搜索过程，因为一个逻辑程序的对称部分有可能是指数级多的。因此，我们的办法就是根据每个具体的逻辑程序自身的特点，把它的足够多的对称给破坏掉。

总过程图


SBASS
Symmetry Detection
Logic Program
Program Reader
Graph Encoding
SAUCY
Logic Program + SBC
Symmetry Breaking
Automor-phism

4）实验结果举例



鸽-洞问题：把n只鸽子放到$n-1$个洞中，使得每一只鸽子都放到不同的洞中。很显然，这是不可能的。用消解方法去解决这个问题，需要指数级的时间，基于**ASP**且用对称破坏方法，可以把计算时间减少为多项式时间。
首先，把它转换为逻辑程序。

$$p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n-1} \leftarrow \qquad i \in 1 \cdots n$$

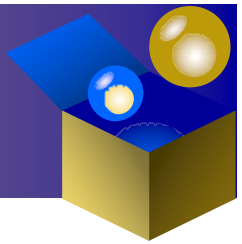
$$\leftarrow p_{i,j}\ ,\ p_{k,j} \qquad i \neq k$$

其次，画出它的躯干-原子图。省略。

再次，把躯干-原子图转换为它的有向三色图。

最后就是求出它的图自同构对称，同时构造它的对称破坏约束。

4）实验结果举例

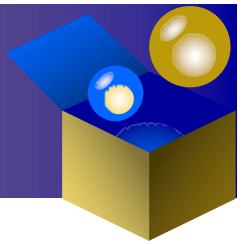


实验图如下：

Table 1. Runtime results in seconds for Pigeon Hole problems.

$\#n$	#gen.	*sbass*	$clasp_1^\pi$	$clasp_5^\pi$	$clasp_\infty^\pi$	$clasp_\infty^\theta$	*clasp*
11	18	0.05	0.38	0.15	0.06	0.02	0.62
12	20	0.08	4.09	0.07	0.22	0.03	5.99
13	22	0.11	30.57	0.43	0.32	0.03	53.39
14	24	0.16	272.72	4.95	1.73	0.04	448.98
15	26	0.23	—	62.61	3.02	0.04	—
16	28	0.32	—	—	23.01	0.07	—
17	30	0.44	—	—	130.87	0.10	—

5，结论与展望



1）结论

对称破坏可以达到简化某类**AS**程序的求解过程的目的，它还有一些有意义的应用。

2）展望

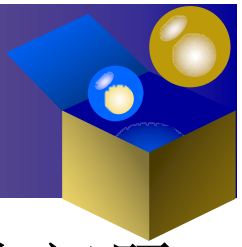
ASP的应用前景。

莱布尼茨的梦想。

3）进一步的学习和研究计划

细化学习**ASP**的理论并作出初步的研究设想。

1）结论

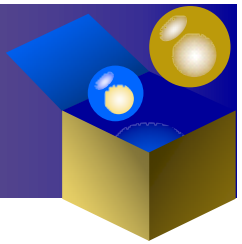


除了鸽-洞问题外，对称破坏方法至少还在以下几个问题上有比较好的应用：

1，拉姆西定理(有限版本)：任给三个正整数s,r,n，且n≥r，则一定存在一个正整数m大于等于n，使得对于集合X={0，…，m-1}，不管怎样把所有的X的r元子集划分成s个类，总是存在X的n元子集Y使得Y的所有r元子集都属于同一个类。

2，全间隔系列问题：
3，优雅图：
4，回答集枚举：
5，约束回答集编程：
6，分布式非单调多语境系统：

2）展望



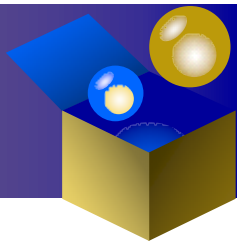
1、ASP的应用前景：自**ASP**出现以来，逻辑程序研究的热点就转向了**ASP**，近十年来关于**ASP**的应用研究更是增长迅速，这在很大原因上是它在知识表达和处理非单调推理上的方便性和实用性。

2、莱布尼茨的梦想：我们知道，他的一个梦想是希望把推理化归为计算。

计算机问世后，人们自然会想着把推理让计算机去实现，这就是机器推理。尽管逻辑学家已经建立了那么多逻辑演算系统，但让机器实现完全的自动推理，困难却是非常大的。

随着计算机科学的进展和计算硬件软件性能的提高，以及人工智能和逻辑学本身的发展，我们希望，机器自动化推理可以像牛顿-莱布尼茨所发明的微积分那样，有着许多广泛而不同的应用。这可以看作是莱布尼茨梦想的一个现代推广。

3）进一步的学习和研究计划



1、学习方面

1）深入学习逻辑程序特别是ASP的语法特性和各种语义，这是从数学和逻辑的角度；

2）从算法的角度深入了解Prolog和ASP求解机制的不同，以及它们是如何在计算机上实现的，这是从计算机理论科学的角度；

3）编程实现：利用visual prolog编一些有意思的逻辑程序，以熟悉prolog编程的语法。

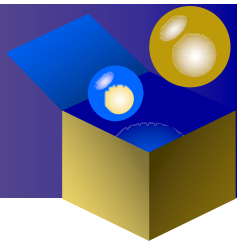
2、研究方面

1）把对称-破坏方法扩展地用于含选择规则和权约束的ASP中去；

2）尝试把多主体道义逻辑、分布式非单调多语境系统、博弈逻辑和ASP结合起来研究；

3）研究有序ASP的良基语义和算法实现，以及有序ASP的各类语义之间的关系。

致谢



首先，感谢大家耐心地听我把这个报告作完；

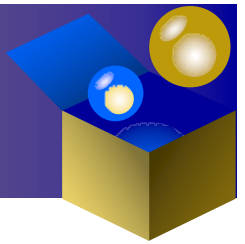
其次，特别感谢叶峰老师和傅庆芳师姐给我指出了原**PPT**中的某些错误；

再次，感谢叶峰老师、王彦晶老师、傅庆芳师姐、陈星群师兄、沙春燕师姐，他们五人给我提的修改意见加起来有一百多条！很抱歉我没有来得及认真考虑每一条意见，也没能在最大程度上进行一定的修改；

又次，感谢大洋彼岸的徐召清师兄以及清华的刘奋荣老师，他们认真地看完了我的**PPT**并表达了看法；

最后，感谢李延军师弟和许春梅师姐两位评论人，感谢从未谋面的安春蕊女士和廉德宁同学，她们没怎么学过逻辑，却愿意在网上甘当我的听众。

参考文献



1，Christian Drescher.Symmetry Breaking for Answer Set Programming,2010.

2，Minh Dao-Tran,Thomas Eiter,Michael Fink,Thomas Krennwallne. Distributed Nonmonotonic Multi-Context Systems,2010.

3，Chitta Baral.*Knowledge representation reasoning and declarative problem solving with Answer sets*,2001.

4，Michael Spivey.*An introduction to logic programming through Prolog*,2008.

5，Georg Boenn, Martin Brain,Marina de vos,John ffitch Martin.Automatic Music Composition using Answer Set Programming ,2010.

6，Michael Gelfond,Vladimir Lifschitz.The stable model semantics for logic programming ,1988.

7，Vladimir Lifschitz.What is answer set programming,2008.

8，Christian Drescher,Oana Tifrea,Toby Walsh.Symmetry-breaking answer set solving,2010.

9，Thomas Linke and Vladimir Sarsakov.Suitable Graphs for Answer Set Programming,Springer-Verlag Berlin Heidelber, 2005.

10，Anil Nerode,Richard A.Shore.*Logic for Applications*,China Machine Press, 2006.

参考文献

11，谢红梅.有序逻辑程序的回答集语义研究，南京航空航天大学硕士学位论文，2005.

12，李鑫.Answer Set编程及其应用研究，电子科技大学博士学位论文，2009.

13，谢颖.归纳逻辑程序设计初探，北京师范大学硕士学位论文，2008.

14，周立柱.Prolog逻辑程序设计及应用[M]，清华大学出版社，1991.

15，Anil Nerode,Richard A.Shore著，丁德成、徐亚涛、吴永成、金陈园译.应用逻辑[M]，机械工业出版社，2007.

16，张镇华.幸福结局问题——鸽笼原理与拉姆西定理，2003.

17，雷英杰，刑清华，孙金萍.Visual Prolog语言教程[M]，陕西科学技术技术出版社，2001.

18，戴琼，邹潇湘，谭建龙.关于图同构复杂性的分析[J]，计算机科学，2006.


Thank You!