符号学习简介

归纳逻辑程序设计（二）

(Press ? for help, n and p for next and previous slide)

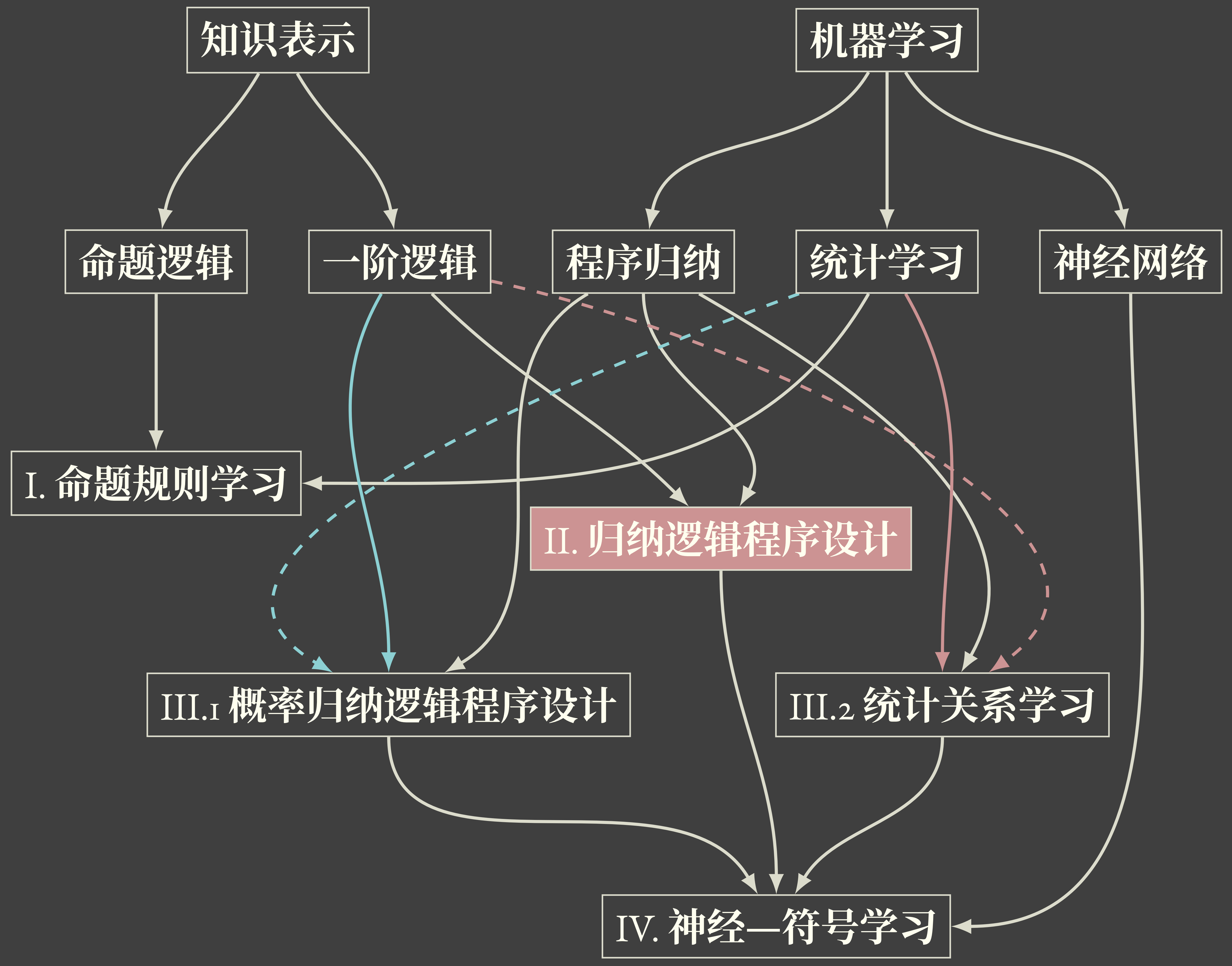
戴望州

南京大学智能科学与技术学院

2025年-秋季

https://daiwz.net

路径图


知识表示
机器学习
命题逻辑
一阶逻辑
程序归纳
统计学习
神经网络
I. 命题规则学习
II. 归纳逻辑程序设计
III.1 概率归纳逻辑程序设计
III.2 统计关系学习
IV. 神经—符号学习

符号学习简介 · 归纳逻辑程序设计（二）

1. **实质蕴涵**
2. 逆蕴涵
3. Progol

实质蕴涵

“Material Implication”

$$A \to B$$

真值表

A	B	$A \to B$
$True$	$True$	$True$
$True$	$False$	$False$
$False$	$True$	$True$
$False$	$False$	$True$

第一种解释

A	B	$A \to B$
$True$	$True$	$True$
$True$	$False$	$False$
$False$	$True$	$X =?$
$False$	$False$	$Y =?$

第二种解释

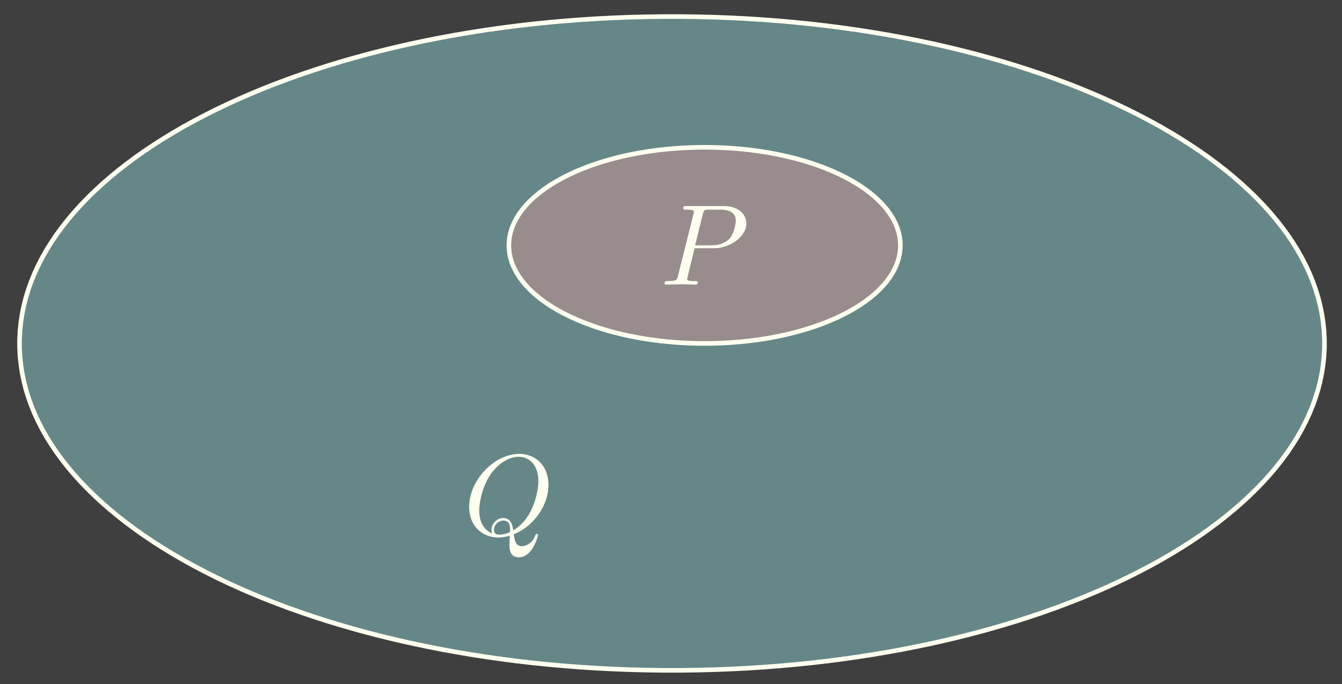
$$(A \land B) \to B$$

第三种解释

“如果中国男足出线，我就把名字倒着写”

Vacuous Truth

$$\forall a P(a) \to Q(a)$$



Entailment：重言蕴涵

$$\Gamma, A \models B$$

Entailment：重言蕴涵

$$\Gamma \models A \to B$$

摄涵与实质蕴涵

摄涵不等于蕴涵，例如

$$
\begin{aligned}
C &= \mathrm{nat}(s(X)) \leftarrow \mathrm{nat}(X) \\
D &= \mathrm{nat}(s(s(Y))) \leftarrow \mathrm{nat}(Y)
\end{aligned}
$$

有$C \rightarrow D$却没有$C \preceq D$。

摄涵与实质蕴涵

没有函数符的例子

$$
\begin{aligned}
C &= p(X,Z) \leftarrow p(X,Y) \wedge p(Y,Z) \\
D &= p(A,D) \leftarrow p(A,B) \wedge p(B,C) \wedge p(C,D)
\end{aligned}
$$

有$C \to D$却没有$C \preceq D$。

从SLD-归结的角度理解

$$
\begin{tikzcd}
C \arrow[d] & B_0 \arrow[dl] \\
R_1 \arrow[d] & B_1 \arrow[dl] \\
R_2 \arrow[d, dashed] & B_{s-2} \arrow[dl] \\
R_{s-1} \arrow[d] & B_{s-1} \arrow[dl] \\
R_s = \Box &
\end{tikzcd}
$$

摄涵与实质蕴涵

定理：若C不是自归结的，且D不是重言式，那么$C \to D$等价于$C \preceq D$。

实质蕴涵意义上的泛化

若$C \rightarrow D$，那么

1. D是重言式;
2. $C \preceq D$;
3. $E \preceq D$且E是由C通过自归结得到。

然而，正是自递归导致$C \rightarrow D$是不可判定的！所以子句集的Least General Generalisation under Implication (LGGI)是**不可计算**的。

如何解决?

1. "Roots of self-resolution" [S. Muggleton, 1992]
2. T-Implication [P. Idestam-Almquist, 1995]
3. Sub-saturants [S. Muggleton, 1995]

但可惜的是由于递归导致的$C \rightarrow D$不可判定，这些基于实质蕴涵的泛化无法做到完备，即无法保证在有限时间内计算出正确的C。

符号学习简介・归纳逻辑程序设计（二）

1. 实质蕴涵
2. **逆蕴涵**
3. Progol

语构蕴涵

$$\cancel{B, H \vdash_{\text{SLD-Resolution}} E}$$

- 由于**自递归**的缘故，直接用θ-subsumption进行归纳是不可判定的

语义蕴涵

$$B, H \models E$$

逆（语义）蕴涵

$$B, \neg E \models \neg H$$

逆（语义）蕴涵

$$B, \neg E \models \neg\bot \models \neg H$$

- $\neg\bot$：在每个$\mathrm{M}_{B\wedge\neg E}$中均为真的**原子命题**

逆（语义）蕴涵

$$H \models \perp$$

- H 中的每一个子句均是 $\perp$ 的子集
- 即 $H \preceq \perp$（**为什么这里没有摄涵和自递归导致的问题？**）

底子句（Bottom clauses）

- B:
  - `anim(X):-pet(X).`
  - `pet(X):-dog(X).`
- E: `nice(X):-dog(X).`
- $\bot$: `nice(X):-dog(X),pet(X),anim(X).`

底子句（Bottom clauses）

- B:
  - `hasbeak(X):-bird(X).`
  - `bird(X):-vulture(X).`
- E: `hasbeak(tweaty).`
- $\bot$:
  - `hasbeak(tweaty).`
  - `bird(tweaty).`
  - `vulture(tweaty).`

底子句（Bottom clauses）

- B: `sentence([],[]).`
- E: `sentence([a,a,a],[]).`
- $\bot$: `sentence([a,a,a],[]):-sentence([],[]).`

逆蕴涵完备吗?

- 看起来，只要$\models$完备，它就完备了
- 若出现自递归，可以想办法控制“$\models$”递归深度，这可基于下面两个假设
  - 越一般的程序应该越短，所以除非有一个极端样本，如$nat(s(s(s(s(s(\ldots(0)))))))$。考虑

```
% Background knowledge
nat(0).
nat(s(s(X))) :- nat(X).
% Example
nat(s(s(s(s(...s(x))))))). % odd numbers of "s/1"
```

  - 在定子句程序中“$\models$”可以通过SLD-归结快速计算（由有效性保证）

逆蕴涵完备吗?

- 互递归呢? [A Yamamoto, 1997]

```
% Background knowledge
even(0).
even(s(X)) :- odd(X).
% Example
odd(s(s(s(0)))).
% Bottom clause
not(Bot) = ( even(0), not( odd( s(s(s(0))) ) ) ).
```

让逆蕴涵变完备

利用闭世界假设，将Herbrand Base里所有非Herbrand Model的原子全部纳入进$\perp$。

[Muggleton ,1998]

定义（增广底子句集）令B为一个Horn子句集合，E为一个子句使得$F=(B\cup\neg E)$可满足（即$B\not\models E$）。E在B下的增广底子句集$BOT(B,E)$定义如下：

- $BOT^{+}(B,E)=\{a|a\in \mathrm{B}(F)\backslash \mathrm{M}(F)\}$
- $BOT^{-}(B,E)=\{\neg a|a\in \mathrm{M}(F)\}$
- $BOT(B,E)=BOT^{+}(B,E)\cup BOT^{-}(B,E)$

其中$\mathrm{B}(F)$为F的Herbrand Base，$\mathrm{M}(F)$为F的Least Herbrand Model。

让逆蕴涵变完备

Yamamoto 的例子：

$$B = \left\{ \begin{array}{l} even(0) \leftarrow \\ even(Y) \leftarrow s(X,Y), odd(X) \end{array} \right\}$$

$$E = odd(Z) \leftarrow s(Y,Z), s(X,Y), s(0,X)$$

$$\neg E = \left\{ \begin{array}{c} \leftarrow odd(c_z) \\ s(c_y, c_z) \leftarrow \\ s(c_x, c_y) \leftarrow \\ s(0, c_x) \leftarrow \end{array} \right\}$$

逆蕴涵的完备性

定义（增广底子句集下的逆蕴涵）令B为一个Horn子句集，E为一个子句使得$B \nvDash E$。一个子句H被称为从E通过B下的逆蕴涵获得，当且仅当存在一个$H' \in BOT(B, E)$有$H \preceq H'$。

定理（逆蕴涵的完备性）令B为一个Horn子句集，E为一个子句使得$F = (B \cup \neg E)$可满足（即$B \nvDash E$）。$G = (B \cup H \cup \neg E)$不可满足（即$B \cup H \models E$）且$\mathrm{B}(G) = \mathrm{B}(F)$，那么$H$可以从$E$通过$B$下的逆蕴涵获得。

逆蕴涵的问题

显然，

这个包含程序Herbrand Base的空间**太大了**！

符号学习简介・归纳逻辑程序设计（二）

1. 实质蕴涵
2. 逆蕴涵
3. **Progol**

Mode 声明语言

Progol 为了控制⊥形式的种类，使用两种mode声明谓词：`modeh(N, Atom)` 和 `modeb(N, Atom)`

- 其中`N`叫做recall，表示一条⊥子句中`Atom`能被实例化的种类个数， `N=*`表示无限制次数。
- `Atom=p(T,T,...)`是原子公式的模板。
- `T` 可以是`+Type, -Type, #Type`； `Type`可以是`int, list, any`等等。
  - `#`表示具体项（ground term）
  - `+`表示输入变量（来自它前面原子的`-`变量或规则头）
  - `-`表示输出变量（用来给后面的原子做输入）

Mode 声明语言

Progol 为了控制 $\perp$ 形式的长度，使用参数 i 控制规则深度（即i,j-determinism 里的 i），用 h 控制归结证明长度。

例子:

```
modeh(*, f(+int, -int)).
modeb(*, d(+int, -int)).
modeb(*, f(+int, -int)).
modeb(*, m(+int, +int, -int)).
```

在 $i = 2$ 时允许下列形式的底子句:

```
f(A,B) :- d(A,C),f(C,D),m(A,D,B).
f(A,B) :- f(B,C),d(A,C),d(C,D),m(C,D,A).
...
```

Progol 算法

1. 从样本集合里挑一个正样例 e；
2. 根据`mode`声明和参数 i 构造一个摄涵 e 的底子句 $\perp_i$；
3. 对 $\perp_i$ 进行利用精化算子进行泛化（Best-first search）；
4. 重复步骤1，直至覆盖全部样例。

Progol 算法

构造$\bot_i$

1. 初始化：将e中的常元替换为变元（相同常元换位同一变元）
2. 令V为所有与+关联的变元（初始化时均为e中变元），并记录它们代换了哪些项
3. 令深度$k=0$
4. 循环：
   - 若$k=i$则返回$\bot_i$，否则$k=k+1$
   - 对背景知识中所有逻辑文字：
     - 找到所有**符合约束**的变元代换形式（如果和e中有一样的项，用V里对应的变元代换）
     - 尝试在h步内对它们进行证明
     - 将证明成功的原子公式加入$\bot_i$并将输出（被重新绑定）的新变元加入V

例子

```
:- mode(*,mem(+any,+list)).
:- mode(1,((+list) = ([-any|-list]))).

:- aleph_set(i,3).
:- aleph_set(noise,0).

:- determination(mem/2,mem/2).
:- determination(mem/2,'='/2).

:-begin_bg.

:-end_bg.
:-begin_in_pos.
mem(0,[0]).
mem(1,[1]).
mem(2,[2]).
mem(3,[3]).
mem(4,[4]).
mem(0,[0,0]).
mem(0,[0,1]).
mem(0,[0,2]).
```

例子

```
?- induce(Program).
[select example] [1]
[sat] [1]
[mem(0,[0])]

[bottom clause]
mem(A,B) :-
   B=[A|C].
[literals] [2]
[saturation time] [0.00024224999999999994]
[reduce]
[best label so far] [[1,0,2,1]/0]
mem(A,B).
[19/6]
mem(A,B) :-
   B=[A|C].
[12/0]
[-------------------------------------]
[found clause]
mem(A,B) :-
   B=[A|C]
```

小结

小结

- Inverse implication 与 θ-subsumption
- Inverse entailment
- Progol 是直到 2012 年 Metagol 提出前最成功的 ILP 系统
  - 背景知识可以是 Horn 子句
  - 训练样例可以是 Horn 子句
    - 甚至可以泛化负样例（`:-Negative_Example.`）
  - 第一次较好地解决了递归问题
  - **很难进行谓词发明**