符号学习简介

一阶逻辑与逻辑程序（下）

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

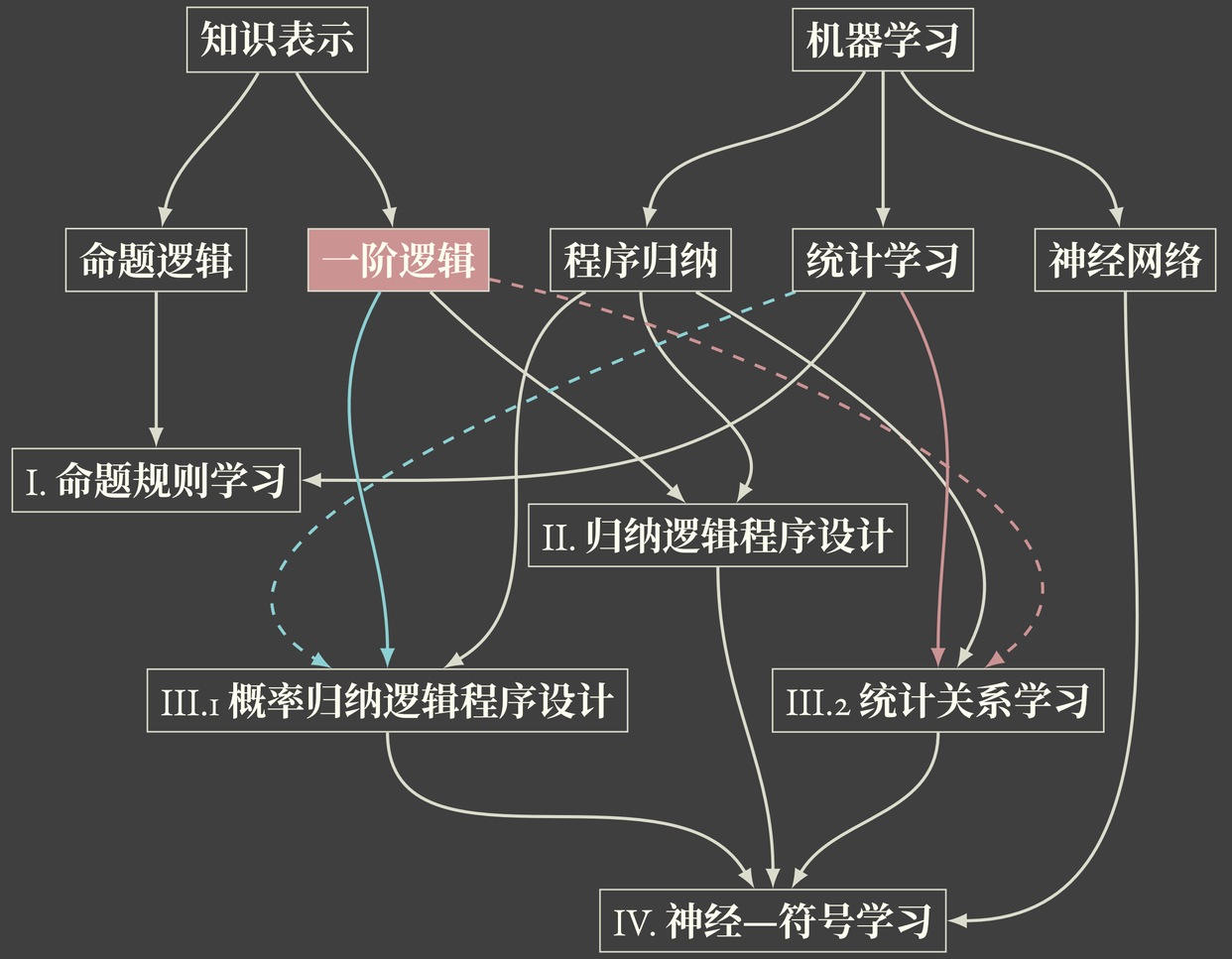
戴望州

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・一阶逻辑与逻辑程序（下）

1. **逻辑程序**
2. 逻辑程序的模型
3. 元解释器

Horn子句

至多只有一个肯定文字的**子句**

$$H \vee \neg B_1 \vee \neg B_2 \vee \cdots \vee \neg B_n = H \leftarrow B_1 \wedge B_2 \wedge \cdots \wedge B_n$$

其中H和B_i均为**原子公式**

Horn子句可以分为三种类型：

1. **定子句（definite clauses）**：有且仅有一个肯定文字

$$H \leftarrow B_1, B_2, \cdots, B_n$$

2. **单位子句（unit clauses）**：无否定文字的定义子句

$$H \leftarrow$$

3. **目标子句（goal clauses）**：无肯定文字

$$\leftarrow B_1, B_2, \cdots, B_n$$

Prolog规则

在逻辑程序（**Pro**gramming with **Log**ic, Prolog）中，我们用：

- `":-"`（“colon dash”）代替“$\leftarrow$”
- `","`代替“$\wedge$”
- `"."`代表句子完结

对应地，有三类Prolog语句：

1. **规则（rules）**：`H :- B_1, B_2, ..., B_n.`
2. **事实（facts）**：`H.`
3. **问句（queries）**：`?- B_1, B_2, ..., B_n.`

- **逻辑程序 = 规则 + 事实**
- “Programming as theory building”
- Prolog (Horn子句)是一阶逻辑语言的一个**图灵完备子集**

Prolog中的推理

1. 归结（resolution）
2. 线性归结（linear resolution）
3. 选择线性归结（selective linear resolution, SL-resolution）
4. **定子句上的选择线性归结（SLD-resolution）**

归结原理

$$\frac{L_1 \vee \cdots \vee L_i \vee \cdots \vee L_k, \quad M_1 \vee \cdots \vee M_j \vee \cdots \vee M_n}{(L_1 \vee \cdots \vee L_{i-1} \vee L_{i+1} \vee \cdots \vee L_k \vee M_1 \vee \cdots \vee M_{j-1} \vee M_{j+1} \vee \cdots \vee M_n)\theta}$$

其中：

- L_i与M_j谓词相同、符号相反
- θ是L_i与$\neg M_j$的**最大一般合一化子**（Most General Unifier, MGU）
- 经过替换后的$L_i\theta$与$\neg M_j\theta$为**互补**（complement）文字
- 横线下面的部分被称为**归结商**（resolvent）

对于一个一阶逻辑合式公式集合$S = \{\alpha, \beta, \ldots\}$

- 它的**合一化子**（unifier）是一个替换θ，使得$\alpha\theta = \beta\theta = \ldots$
- S的**最大一般合一化子**是一个替换θ^*，使得S的任意一个合一化子$\theta = \theta^*\sigma$，其中σ是一个非空替换

归结原理

归结定理（Resolution Theorem）： 若S是任意有穷子句集，那么S是不可满足的当且仅当$\mathfrak{R}^n(S)$中包含空子句，其中$\mathfrak{R}^n(S)$是对S中的子句连续做n次归结后的结果。

归结反驳（Resolution refutation）：“反证法”

$$\Gamma \vdash A \Leftrightarrow \Gamma, \neg A \vdash \square$$

归结原理

例子：一个有结合律的乘法系统中，若左乘和右乘均有解，那么这个系统中至少存在一个右单位元。

$$
\begin{aligned}
C_1 &: mul(U,Z,W) \vee \neg mul(X,Y,U) \vee \neg mul(Y,Z,V) \vee \neg mul(X,V,W) \\
C_2 &: mul(X,V,W) \vee \neg mul(X,Y,U) \vee \neg mul(Y,Z,V) \vee \neg mul(U,Z,W) \\
C_3 &: mul(g(X,Y),X,Y) \\
C_4 &: mul(X,h(X,Y),Y) \\
C_5 &: mul(X,Y,f(X,Y)) \\
C_6 &: \neg mul(X,k(X),X)
\end{aligned}
$$

线性归结

线性归结（Linear Resolution）：子句集合S的一个线性归结D是一个序列$(C_1, \ldots, C_n)$，使得$C_1 \in S$且C_{i+1}是C_i与B的归结商，其中要么：

- $B \in S$，或者
- $B = C_j$是C_i的祖先，且$j < i$

例子：$S = \{\neg p(X) \vee \neg p(Z), p(X) \vee r(Z), \neg r(X) \vee q(Y), \neg q(Y) \vee \neg r(Y)\}$

- 有一个线性归结序列：$D = (\neg p(X) \vee \neg p(Z), r(Z), q(Y), \neg r(Y), \square)$

线性归结

线性归结的搜索空间非常大：$S = \{P \vee Q, \neg P, \neg Q \vee R, \neg R \vee S, \neg R \vee \neg S \vee T, P \vee \neg T\}$

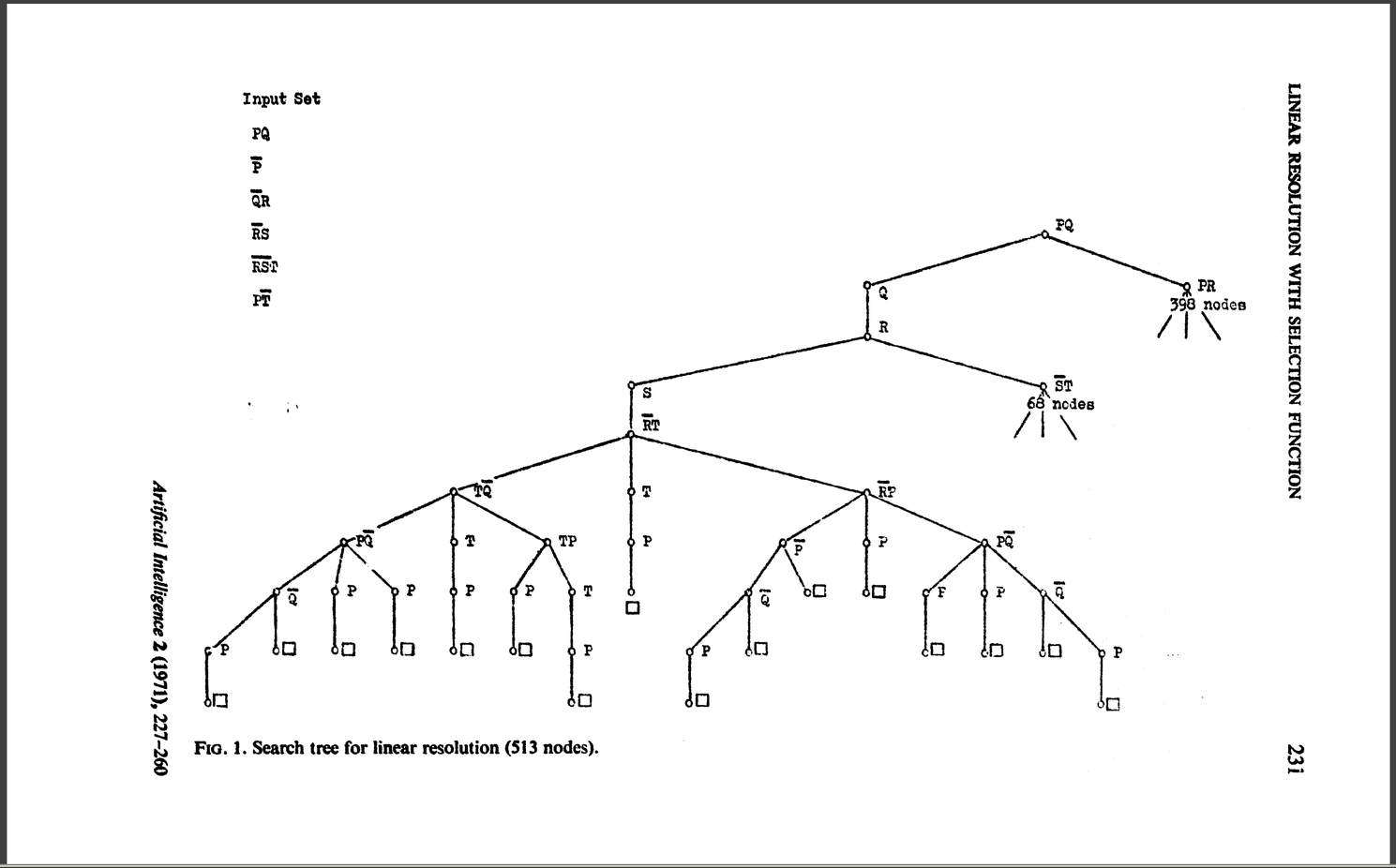
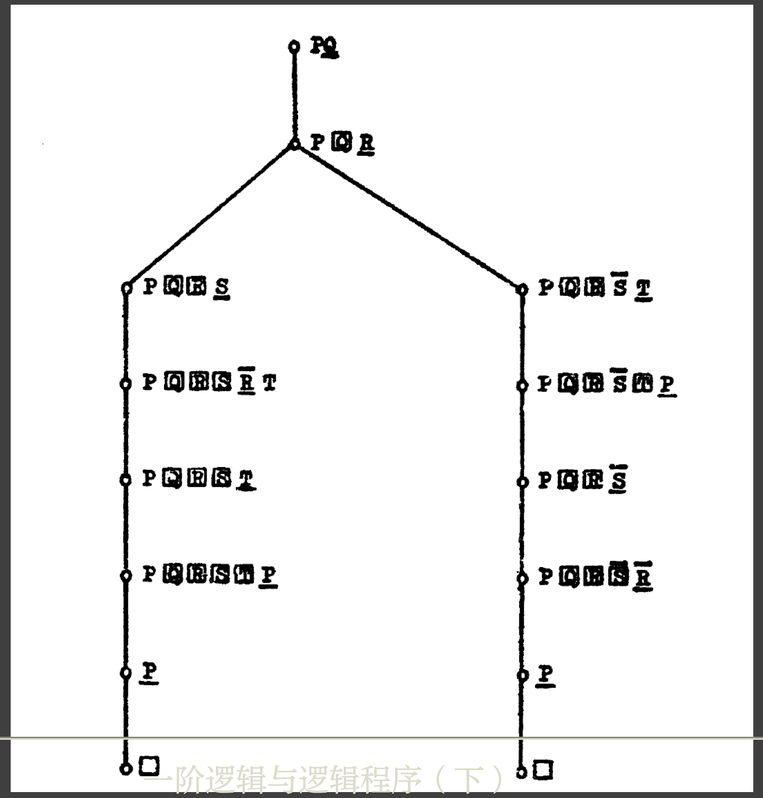


FIG. 1. Search tree for linear resolution (513 nodes).

选择线性归结

选择线性归结（Selective Linear Resolution）在线性归结的基础上添加了选择函数（selection function），每次只选择一个文字进行归结

例如，依然对于$S = \{PQ, \bar{P}, \bar{Q}R, \bar{R}S, \bar{R}\bar{S}T, P\bar{T}\}$



定子句的选择线性归结

Demo:

```
% Facts
mother(ann,amy).
mother(ann,andy).
mother(amy,amelia).
mother(linda,gavin).
father(steve,amy).
father(steve,andy).
father(gavin,amelia).
father(andy,spongebob).
% Rules
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
% Query
?- grandparent(ann, spongebob).
```

SLD-归结

SLD-归结在做什么？

1. 将目标子句通过规则进行归结，分解为更多的子目标
2. 对子目标继续归结
3. 递归地进行深度优先搜索
4. 失败时回溯

令$G = \leftarrow A_1, \ldots, A_m, \ldots, A_k$且$C = A \leftarrow B_1, \ldots, B_q$，那么我们称$G'$是通过从$G$与$C$用它们的MGU θ**推导**而来的（ derived ），当：

1. A_m是一个原子公式，被称为是从G中选择出的（ selected atom ）
2. θ是A_m与A的MGU
3. G'是一个新的目标子句$\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta$，它是$G$与$C$的归结商

SLD-归结

SLD-推导（SLD-derivation）：若P是一个逻辑程序且G是一个目标子句。$\mathrm{P} \cup \{G\}$的一个SLD-推导由以下部分组成：

- 一个目标子句序列（可能无穷）$G_0 = G, G_1, \ldots$
- 一个程序规则序列$C_1, \ldots C_2, \ldots$
- 一个MGU序列$\theta_1, \theta_2, \ldots$

使得G_{i+1}是由G_i与C_{i+1}通过θ_{i+1}推导而来。

SLD-反驳（SLD-refutation）：$\mathrm{P} \cup \{G\}$的一个SLD-反驳是一个有穷的SLD-推导，且它的最后一个目标$G_n = \square$，我们称这个反驳的长度为n。

SLD-归结的答案

计算答案（computed answer）：若P是一个逻辑程序且G是一个目标子句。$P \cup \{G\}$的一个计算答案是一个G上的变元替换θ，它是由SLD-反驳过程中使用的所有MGU序列复合而来。

```
mother(ann,amy).       father(steve,amy).
mother(ann,andy).      father(steve,andy).
mother(amy,amelia).    father(gavin,amelia).
mother(linda,gavin).   father(andy,spongebob).

parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).

?- grandparent(ann, A), grandparent(linda, A).
% A = amelia
```

SLD-归结的有效性

定理：若P是一个逻辑程序且G是一个目标子句。那么所有通过SLD-反驳计算得出的答案θ都是正确答案，即$\forall(G)\theta$是P的逻辑推论（$P \models G\theta$）。

证明：令$G =\leftarrow A_1, \ldots, A_k$且$\theta = \theta_1, \ldots, \theta_n$为SLD-反驳中使用的MGU序列，利用关于反驳长度n的数学归纳法来证明$\forall(A_1 \wedge \ldots \wedge A_k)\theta_1 \ldots \theta_n$是P的一个逻辑推论。

1. 当长度$n = 1$时，说明当前只有一个原子可以归结，即$G =\leftarrow A_1$且P中包含一个单位子句（事实）$A \leftarrow$，有$A_1\theta_1 = A\theta_1$成立。由于$A_1\theta_1$是单位子句A的一个实例，显然它是P的一个逻辑推论（任意满足P的真值指派都令$A_1\theta_1$为真）。
2. 对所有长度为$n-1$的反驳来说结论成立，而反驳$G_0 =\leftarrow A_1, \ldots, A_k$时需要进行$n$步归结，用到的所有替换为$\theta_1 \ldots \theta_n$。令第一步归结中使用的$C_1 = A \leftarrow B_1, \ldots, B_q$，且SLD-反驳从$G_0$中选择了$A_m$。那么根据归纳假设，剩下的$n-1$步中得到的答案$\forall((A_1 \wedge \ldots \wedge A_{m-1} \wedge B_1 \wedge \ldots \wedge B_q \wedge A_{m+1} \wedge \ldots \wedge A_k)\theta_1 \ldots \theta_n)$是P的逻辑推论。那么若$q > 0$，则$\forall(B_1 \wedge \ldots \wedge B_q)\theta_1 \ldots \theta_n$也是P的逻辑推论。因此$\forall(A_m\theta_1 \ldots \theta_n) = \forall(A\theta_1 \ldots \theta_n)$也是P的逻辑推论。显然$\forall((A_1, \ldots, A_k)\theta_1 \ldots \theta_n)$还是P的逻辑推论，证毕。

符号学习简介・一阶逻辑与逻辑程序（下）

1. 逻辑程序
2. **逻辑程序的模型**
3. 元解释器

逻辑程序的语义

埃尔布朗域（Herbrand Universe）若P是一个逻辑程序，那么P的埃尔布朗域U_P是P中所有的具体项（ground terms），即P中所有常元和函数符号构成的、不含变元的项。

例如，对于以下逻辑程序P

```
p(X) :- q(f(X), g(X)).
r(a).
```

它的埃尔布朗域U_P为

```
a, f(a), g(a), f(f(a)), f(g(a)), g(f(a)), g(g(a)), ...
```

逻辑程序的语义

埃尔布朗基（Herbrand Base）若P是一个逻辑程序，那么P的埃尔布朗基B_P是P中所有的具体原子公式（ground atoms），即P中所有谓词和U_P构成的、不含变元的原子。

例如，对于以下逻辑程序P

```
p(X) :- q(f(X), g(X)).
r(a).
```

它的埃尔布朗基B_P为

```
p(a). q(a,a). r(a). p(f(a)). p(g(a)). q(f(a),f(a)). q(f(a),g(a)). ...
```

考虑埃尔布朗基的任意子集$I_P \in 2^{B_P}$，它们通过包含关系“$\subseteq$”构成一个格，其中$\top = 2^{B_P}$且$\bot = \emptyset$。

逻辑程序的语义

埃尔布朗解释（Herbrand Interpretation）若P是一个逻辑程序，那么P的埃尔布朗解释I_P是对B_P的一个真值指派，它也可以表示为2^{B_P}的一个子集（出现在该子集中的具体原子赋值为$true$）。

埃尔布朗模型（Herbrand Model）若P是一个逻辑程序，那么P的埃尔布朗模型是满足（satisfies）P的一个埃尔布朗解释，即满足P中所有子句的一个真值指派。

例如，对于以下逻辑程序P

```
p(X) :- q(f(X), g(X)).
r(a).
```

它的两个埃尔布朗模型为

```
MP1 = r(a), p(a), q(f(a),g(a)).
MP2 = r(a).
```

最小埃尔布朗模型

最小埃尔布朗模型（Least Herbrand Model）若P是一个逻辑程序，且$\{\mathrm{M}_i\}$是P的一个非空的埃尔布朗模型集合，那么$\cap_i \mathrm{M}_i$也是P的一个埃尔布朗模型。当$\{\mathrm{M}_i\}$是P所有的埃尔布朗模型时，它们的交集即是最小埃尔布朗模型，记为M_P。

定理：若P是一个逻辑程序，那么$\mathrm{M}_\mathrm{P} = \{A \in \mathrm{B}_\mathrm{P} \mid \mathrm{P} \vdash A\}$。

最小埃尔布朗模型

直接推论算子（Immediate consequence operator）若P是一个逻辑程序，直接推论算子是一个映射 $\mathrm{T_P} : 2^{\mathrm{B_P}} \mapsto 2^{\mathrm{B_P}}$，它有如下定义：令I是一个埃尔布朗解释，那么：

$$\mathrm{T_P(I)} = \{A \in \mathrm{B_P} \mid A \leftarrow A_1 \wedge \ldots \wedge A_n \text{是P中一条规则的具体实例且} \{A_1, \ldots, A_n\} \subseteq \mathrm{I}\}$$

定理：由于$\mathrm{T_P}$是单调的，根据不动点定理，存在不动点令$\mathrm{T_P}(I) = I$。而P的最小埃尔布朗模型$\mathrm{M_P}$则是$\mathrm{T_P}$的最小不动点（least fixed point）。

- 单调：$I_1 \subseteq I_2 \rightarrow \mathrm{T_P}(I_1) \subseteq \mathrm{T_P}(I_2)$

最小埃尔布朗模型的计算

只要找到最小不动点，就能找到逻辑程序的模型

```
mother(ann,amy).   mother(ann,andy).   ...
father(steve,amy). father(steve,andy). ...
% Rules
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

直接推论$\mathrm{T_P}^n$（**有可能无穷**）

```
1: [parent(steve,amy), ..., parent(linda,gavin)]
2: [parent(steve,amy), ..., parent(linda,gavin),
    ancestor(steve,amy), ..., ancestor(linda,gavin)]
3: [parent(steve,amy), ..., parent(linda,gavin),
    ancestor(steve,amy), ..., ancestor(linda,gavin),
    ancestor(steve,amelia, ..., ancestor(linda,amelia))]
4+: no change.
```

SLD-归结的完备性

通过SLD-归结推理出来的结果 vs T_P计算出来的结果

可行原子集合（success set）若P是一个逻辑程序，它的可行原子集合为：

$$\{A \mid \forall A \in B_P \text{使得} P \cup \{\leftarrow A\} \text{有一个SLD-反驳}\}$$

定理：逻辑程序的可行原子集合与它的最小埃尔布朗模型相同

SLD-归结的完备性

定理：若P是一个逻辑程序且G是一个目标子句。若$\mathrm{P} \cup \{G\}$是不可满足的，那么它存在一个SLD-反驳。

符号学习简介·一阶逻辑与逻辑程序（下）

1. 逻辑程序
2. 逻辑程序的模型
3. **元解释器**

元解释器

- **解释器**（interpreter）：“A program that evaluates programs”
- **元解释器**（meta-interpreter）：可以用于解释自身实现语言构成的程序
  - 我们可以用Prolog写一个解释Prolog语言的程序！

为什么是Prolog?

1. Prolog是一种同像性（homoiconic）语言
   - Prolog程序就是Prolog terms
2. Prolog有许多解释器能利用的隐式功能
   - 基于SLD-归结的深度优先搜索
3. Prolog语句非常简单
   - `Head :- Body`

元调用（Meta-call）

若我们有一个程序，

```
natnum(0).
natnum(s(X)) :-
    natnum(X).
```

Prolog能够动态地运行它：

```
?- Goal = natnum(X), call(Goal).
Goal = natnum(0), X = 0 ;
Goal = natnum(s(0)), X = s(0) ;
Goal = natnum(s(s(0))), X = s(s(0)) ;
Goal = natnum(s(s(s(0)))), X = s(s(s(0))) ;
Goal = natnum(s(s(s(s(0))))), X = s(s(s(s(0)))) ;
...
```

自我解析

也叫做自省（reflection and introspection）

若我们有一个程序，

```
:- dynamic complicated_clause/1. % 需要声明这是公共谓词

complicated_clause(A) :-
    goal1(A),
    goal2(A),
    goal3(A).
complicated_clause(1).
```

Prolog能够用`clause/2`解析它：

```
?- clause(complicated_clause(Z), Body).
Body =  (goal1(Z), goal2(Z), goal3(Z)) ;
Z = 1, Body = true.
```

最基础的元解释器

通过meta-call和`clause/2`实现SLD-归结：

```
prove(true).
prove((A,B)) :-
    prove(A),
    prove(B).
prove(Goal) :-
    Goal \= true,
    Goal \= (_,_),
    clause(Goal, Body),
    prove(Body).
```

用在`natnum/1`中：

```
?- prove(natnum(X)).
X = 0 ;
X = s(0) ;
X = s(s(0)) ;
... .
```

最基础的元解释器

通过meta-call和`clause/2`实现SLD-归结：

```
prove(true).
prove((A,B)) :-
    prove(A),
    prove(B).
prove(Goal) :-
    Goal \= true,
    Goal \= (_,_),
    clause(Goal, Body),
    prove(Body).
```

解释不了自己：

```
?- prove(prove(natnum(X))).
ERROR: No permission to access private_procedure `(\=)/2'
```

自定义语法

比如，新定义一个谓词`head_body(Head, Goals)`

```
head_body(natnum(0), []).
head_body(natnum(s(X)), [natnum(X)]).
```

新的SLD-归结元解释器：

```
prove([]).
prove([G|Gs]) :-
    head_body(G, Goals),
    prove(Goals),
    prove(Gs).
```

自定义语法2

利用list difference定义谓词`head_body(Head, Goals0, Goals)`

- `Head`成立当且仅当`Goals0-Goals`成立

```
head_body(natnum(0), Rs, Rs).
head_body(natnum(s(X)), [natnum(X)|Rs], Rs).
```

新的SLD-归结元解释器:

```
prove([]).
prove([G|Gs]) :-
    head_body(G, Goals, Gs),
    prove(Goals).
```

解释解释器的解释器

`prove/1`可以表达为：

```
head_body(prove([]),Rs,Rs).
head_body(prove([G|Gs]),[head_body(G,Goals,Gs),prove(Goals)|Rs], Rs).
```

`head_body/3`自己也可以表达为：

```
head_body(head_body(Head,Goals0,Goals), Rs, Rs) :-
    head_body(Head, Goals0, Goals).
```

解释解释器的解释器

```
% Meta-Interpreter
prove([]).
prove([G|Gs]) :-
    head_body(G, Goals, Gs),
    prove(Goals).

head_body(prove([]),Rs,Rs).
head_body(prove([G|Gs]),[head_body(G,Goals,Gs),prove(Goals)|Rs],Rs).

head_body(head_body(Head,Goals0,Goals), Rs, Rs) :-
    head_body(Head, Goals0, Goals).

% Program
head_body(natnum(0), Rs, Rs).
head_body(natnum(s(X)), [natnum(X)|Rs], Rs).

?- prove([prove([natnum(X)])]).
X = 0 ;
X = s(0) ;
X = s(s(0)) ;
X = s(s(s(0))) .
```

小结

逻辑程序部分的小结

1. 逻辑程序（definite program）的组成：
   - 规则（Horn clause）
   - 事实（unit clause）
2. 归结原理与SLD-归结
   - 问句（goal clause）
   - 反驳（refutation）
3. SLD-归结的有效性与完备性
4. SLD-归结的实现

逻辑程序软件

1. 环境：SWI-Prolog， Scryer Prolog
2. Prolog 教程
   - Learn Prolog Now
   - SWI-Prolog Manual
   - The Power of Prolog
     - Meta-Interpreters