



符号学习简介

归纳逻辑程序设计（一）

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

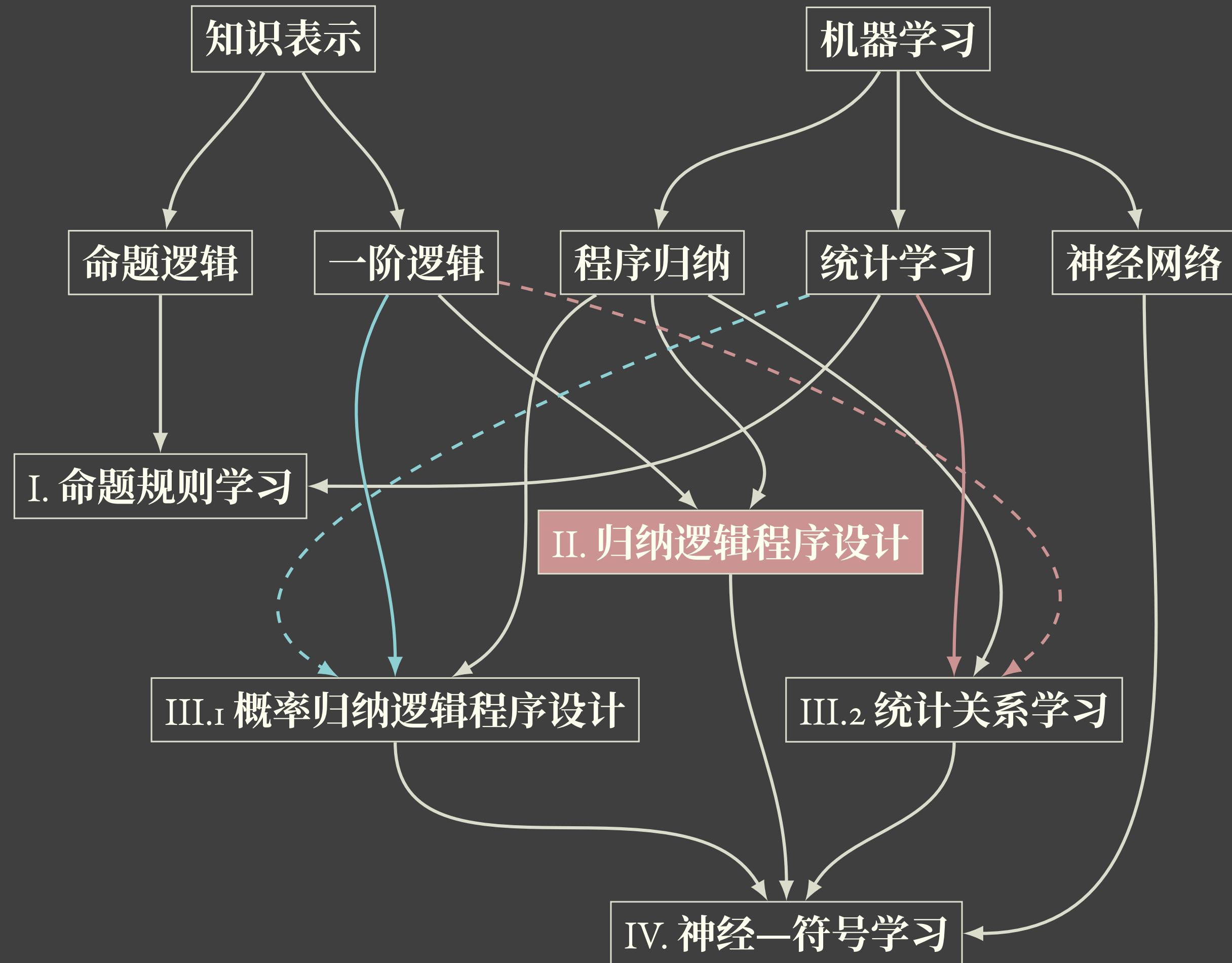
戴望州

南京大学智能科学与技术学院
2025年-秋季

<https://daiwz.net>



路径图





符号学习简介 · 归纳逻辑程序设计（一）

1. 归纳逻辑程序设计
2. 相对最小一般泛化 (RLGG)
3. 逆归结 (Inverse Resolution)

归纳逻辑程序设计



归纳 + 逻辑程序 + 设计

Inductive Logic Programming



演绎、归纳、反绎

“凡生前被火烧死者，其尸口鼻内有烟灰，两手脚皆拳缩……若死后烧者，其人虽手足拳缩，口内即无烟灰。”

——宋慈 · 《洗冤集录》

知识：死后焚烧的尸体鼻腔无烟尘
结果：受害者鼻腔无烟尘
前提：受害者是死后被焚烧

知识：死后焚烧的尸体鼻腔无烟尘
前提：受害者是死后被焚烧
结果：受害者鼻腔无烟尘

前提：受害者是死后被焚烧
结果：受害者鼻腔无烟尘
知识：死后焚烧的尸体鼻腔无烟尘



演绎、归纳、反绎

若 H 是逻辑规则集合， B 和 E 为具体逻辑事实（ground facts）集合，且以下推理成立

$$B, H \models E$$

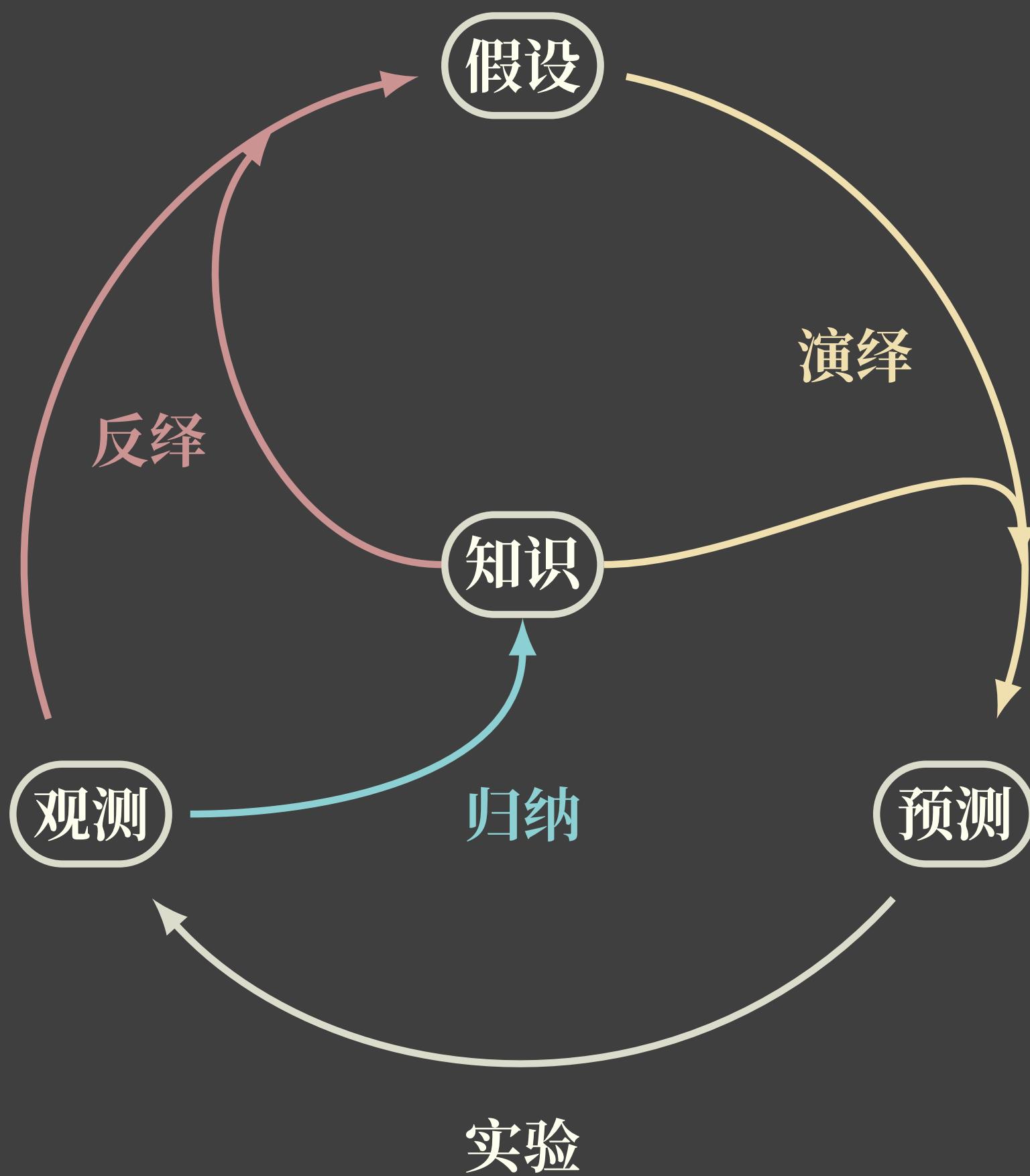
当已知和未知条件不同时，分为三种推理

- > 已知 $B \cup H$ ，推出 B 叫做**演绎**
- > 已知 $B \cup E$ ，推出 H 叫做**归纳**
- > 已知 $H \cup E$ ，推出 B 叫做**反绎**



皮尔斯的科学方法论

Scientific Methodology by Charles S. Pierce (1839/9/10 - 1914/4/19)





归纳逻辑程序设计

$$B, H \models E$$

- > **输入:** $B \cup E$
- > **输出:** H

Remarks:

1. B 、 H 、 E 均为逻辑程序
2. 一般情况下， H 为一阶规则集， E 为具体事实
3. 大部分时候 $E = E^+ \cup E^-$ ，此时需要

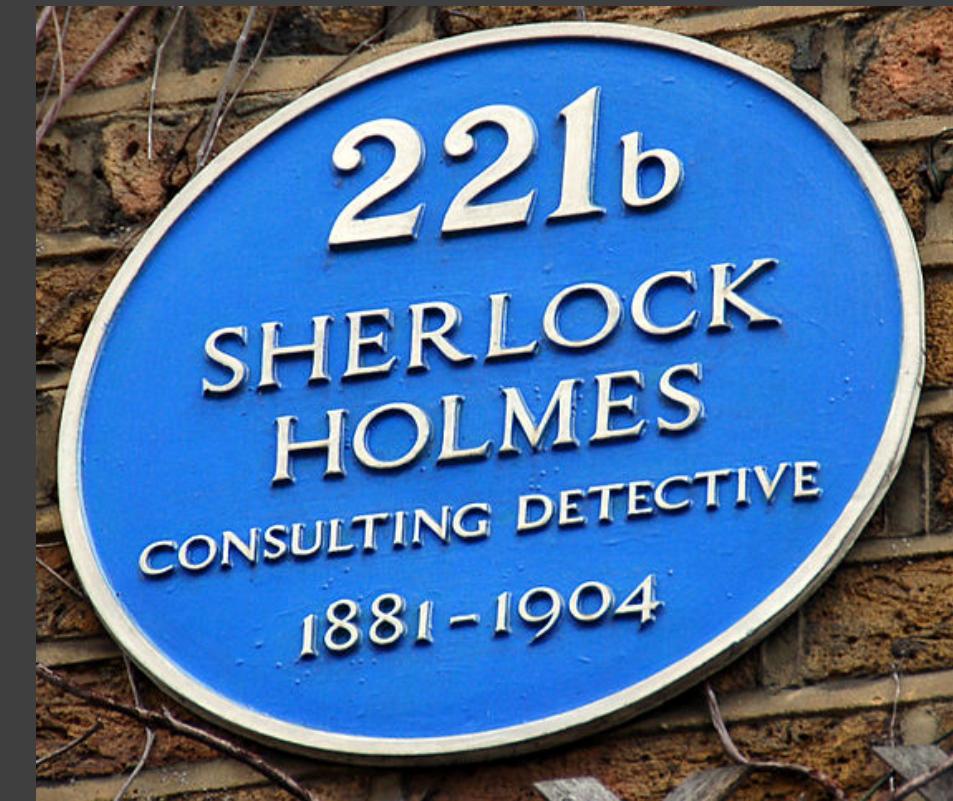
$$(B \cup H \models E^+) \wedge (B \cup H \not\models E^-)$$



符号学习简介 · 归纳逻辑程序设计（一）

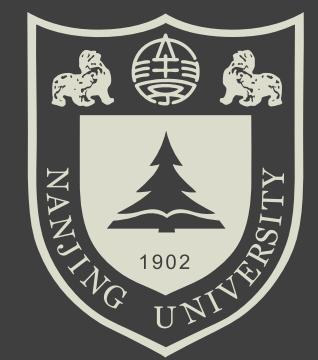
1. 归纳逻辑程序设计
2. 相对最小一般泛化 (RLGG)
3. 逆归结 (Inverse Resolution)

什么是泛化



命题1	孙悟空会变身超级赛亚人
命题2	赛亚人会变成超级赛亚人

命题3	福尔摩斯住在伦敦
命题4	福尔摩斯住在英国



对规则进行泛化

G. D. Plotkin. A Note on Inductive Generalization. *Machine Intelligence*, 5(1): 153-163, 1970.

观测实验结果，对规律进行总结：

- > “The result of heating this bit of iron to 419°C was that it melted”
- > “The result of heating that bit of iron to 419°C was that it melted”
- > “The result of heating any bit of iron to 419°C is that it melted”

表达为逻辑程序：

```
> melted(bit1) :- bit_of_iron(bit1), heated(bit1,419).  
> melted(bit2) :- bit_of_iron(bit2), heated(bit2,419).  
> melted(X) :- bit_of_iron(X), heated(X,419).
```



涵摄 (Subsumption)

叫做“ \preceq ”，也叫做“包摄”。

例如， $A \preceq B$ 读作“ A 摄涵 B ”或者“ B 摄于 A ”，意思是“ A 比 B 更一般”。

定义（摄涵和相容）：

- › 若 W_1, W_2 为项或者文字（在 Plotkin 的论文中被称为“词汇”，word），我们称 $W_1 \preceq W_2$ 当且仅当存在一个变量替换 θ ，使得 $W_1\theta = W_2$ ；
- › 若 C_1, C_2 为逻辑子句，我们称 $C_1 \preceq C_2$ 当且仅当存在一个变量替换 θ ，使得 $C_1\theta \subseteq C_2$ 。
- › 若 $A \preceq B \wedge B \preceq A$ ，则称 A 与 B 相等（equivalent），记为 $A \sim B$ 。
- › 两个词汇是相容的（compatible），当且仅当它们都是项，或者是谓词且符号 (\neg) 相同的文字。

- › $p(X, X, f(g(Y))) \preceq p(l(3), l(3), f(g(X)))$ ，只需令 $\theta = \{X/l(3), Y/X\}$
- › 对于子句（clause）来说也有这样的“更一般”偏序关系： $p(X) \vee p(f) \preceq p(f)$ ，只需令 $\theta = \{X/f\}$



最小一般泛化

若 K 是一个由词汇 (words) 构成的集合，那么 W 是 K 的最小一般泛化 (least general generalisation, LGG) 当且仅当

1. W 是 K 的泛化，即 $\forall V(V \in K \rightarrow W \preceq V)$
2. W 是其中最特殊的一个（即它是所有“更一般的词汇”的上界，即它是在 *subsumption* 关系中的下确界）： $\forall V \forall W'(V \in K \wedge W' \preceq V \rightarrow W' \preceq W)$

对任意非空有限词汇集合，若其中存在至少两个相容元素，那么它存在一个最小一般泛化，且能够通过以下算法得到。设此二者为 W_1, W_2 ，依次执行：

1. 令 $V_i = W_i, \theta_i = \emptyset, i = 1, 2$ 。
2. 找到 V_1 与 V_2 中的项 t_1 和 t_2 满足：位置相同且 $t_1 \neq t_2$ ，且要么 t_1 与 t_2 不相容、要么至少一个为变元。
3. 若无符合条件的项 t_1 和 t_2 则终止。 V_1 即是 $\{W_1, W_2\}$ 的 LGG，且 $V_1 = V_2, V_i \theta_i = W_i, i = 1, 2$ 。
4. 选择一个未在 $\{V_1, V_2\}$ 中出现的变量 X ，并找到所有 t_1 与 t_2 出现的位置，将它们置换为 X 。
5. 令 $\theta_i = \{X/t_i\} \circ \theta_i$ 。
6. 继续步骤 2。



最小一般泛化

$$\{p(f(a, g(Y)), X, g(Y)), p(h(a, g(X)), X, g(X))\}$$

I. 初始化

$$V_1 = p(f(a, g(Y)), X, g(Y))$$

$$V_2 = p(h(a, g(X)), X, g(X))$$

2. 令 $t_1 = f(a, g(Y))$, $t_2 = h(a, g(X))$ 且令 A 为新变量。由步骤 4:

$$V_1 = p(A, X, g(Y))$$

$$V_2 = p(A, X, g(X))$$

通过步骤 5 得到 $\theta_1 = \{A/f(a, g(Y))\}$, $\theta_2 = \{A/h(a, g(X))\}$ 。

3. 令 $t_1 = Y$, $t_2 = X$ 且令新变量为 B :

$$V_1 = p(Y, B, g(B)) = V_2$$

$$\theta_1 = \{B/Y, A/f(a, g(Z))\}$$

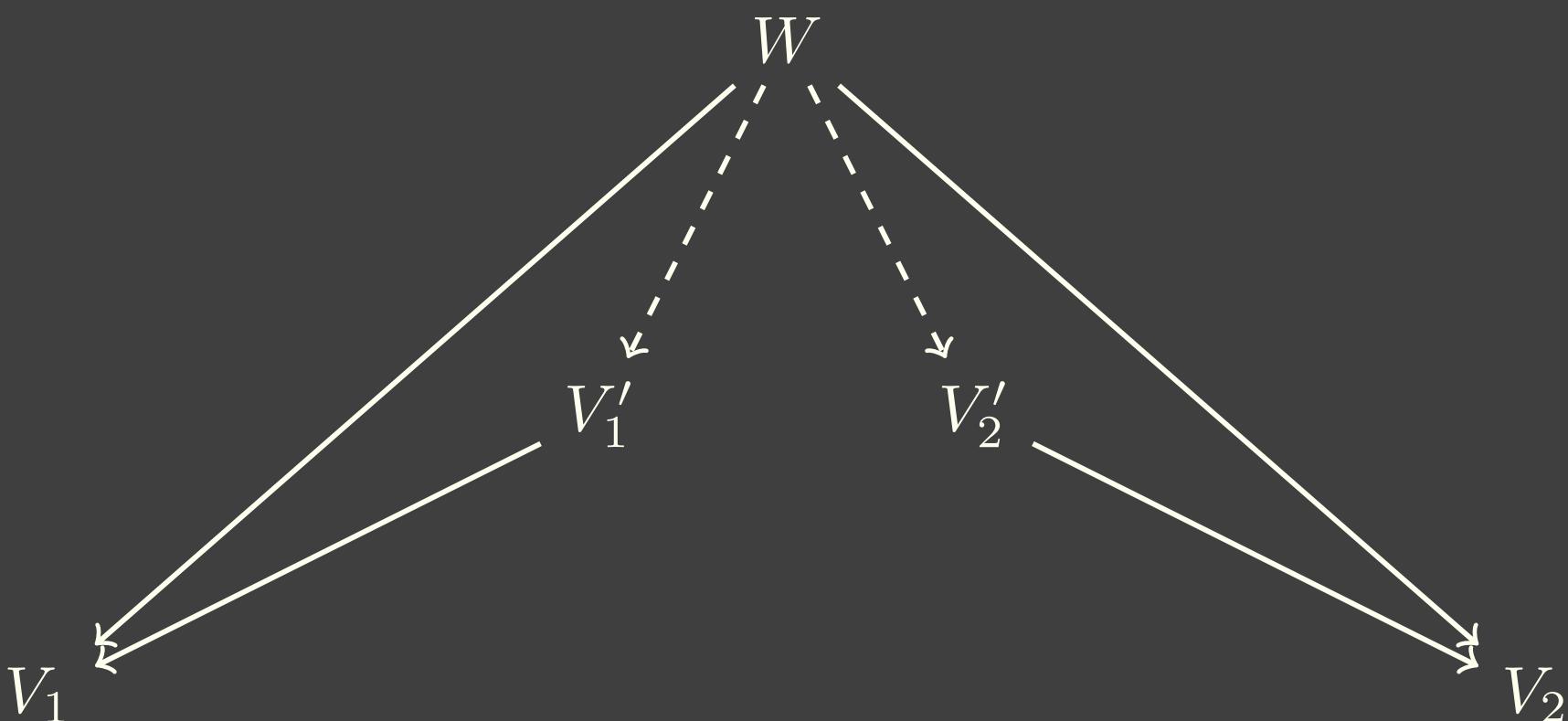
$$\theta_2 = \{B/X, A/h(a, g(Z))\}$$

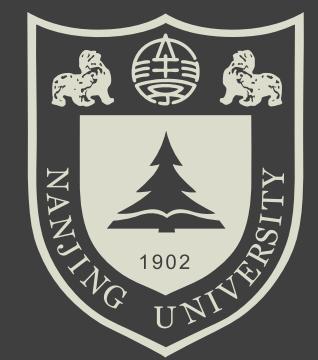


LGG 的存在性

证明思路：

1. 若 V_1 和 V_2 是不相等但相容的词汇，那么一定能找到一对 $\{t_1, t_2\}$ 满足条件 2。
2. 通过步骤 4 得到的 $\{V'_1, V'_2\}$ 必然有 $V'_i \prec V_i$ ，且能最终能令 $V_1^k = V_2^k$ 。
3. 若 V_1 和 V_2 能够找到一个下界 W 且存在 σ_i 令 $V\sigma_i = V_i$ 。那么也存在 σ'_i 令 $V\sigma'_i = V'_i$ 。





子句的最小一般泛化

对任意非空有限的子句集合，其LGG可以通过如下方式计算。不失一般性，当集合为 $\{C_1, C_2\}$ 时：

1. 令 $lgg(C_1, C_2) = \emptyset$, $\theta_i = \emptyset$, $i = 1, 2$ 。
2. 找到 C_1 与 C_2 中相容的逻辑文字 $W_1 \in C_1$ 和 $W_2 \in C_2$ 。
3. 若无复合条件的逻辑文字对，则终止并输出 $lgg(C_1, C_2)$ 。
4. $lgg(C_1, C_2) = lgg(W_1, W_2) \cup lgg(C_1, C_2)$, $\theta_i = \theta'_i \circ \theta_i$ 。这里的 θ'_i 为计算 W_i 时使用的替换。

```
> C1: parent(tom,eve) :- father(tom,eve), male(tom).  
> C2: parent(tom,ann) :- father(tom, ann), male(tom), female(ann).  
> lgg(C1,C2): parent(tom,X) :- father(tom,X), male(tom).
```



相对最小一般泛化

Relative Least General Generalisation: 考虑领域知识时的最小一般泛化。

假设给定背景知识（逻辑程序） B 与两个正样例 $\{e_1, e_2\}$ ，欲找到一条规则 C 覆盖它们，即

$$(B \wedge C \vdash e_1) \wedge (B \wedge C \vdash e_2)$$

那么，对于 e_i 有：

$$\begin{aligned} & B \wedge C \vdash e_i \\ & C \vdash B \rightarrow e_i \\ & \vdash C \rightarrow (B \rightarrow e_i) \end{aligned}$$

也就是说， $C \preceq (e_1 \leftarrow B)$ 且 $C \preceq (e_2 \leftarrow B)$ 。

那么， C 可以是 $lgg(e_1 \leftarrow B, e_2 \leftarrow B)$ 。

> 这里的 $e_i \leftarrow B$ 叫 e_i 关于 B 的饱和规则（saturation）。



相对最小一般泛化

```
% Background knowledge
append([1,2],[3,4],[1,2,3,4]) .
append([a],[],[a]) .
append([],[],[]) .
append([2],[3,4],[2,3,4]) .

% Examples
E1 = append([1,2],[3,4],[1,2,3,4]) .
E2 = append([a],[],[a]) .

% Saturation
C1 = append([1,2],[3,4],[1,2,3,4]) :-
    append([1,2],[3,4],[1,2,3,4]),
    append([a],[],[a]),
    append([],[],[]),
    append([2],[3,4],[2,3,4]) .
C2 = append([a],[],[a]) :-
    append([1,2],[3,4],[1,2,3,4]),
    append([a],[],[a]),
    append([],[],[]),
    append([2],[3,4],[2,3,4]) .
```



RLGG 的剪枝

1. **逻辑归约** (Logical Reduction) : 给定背景知识 B , 当 $B \wedge C \preceq C - \{L\}$ 时, 我们说子句 C 中一个逻辑文字 L 是冗余的。此时可以删去 C 中的 L 。
 - » 归约后会产生等价规则 (仅变量名不同), 同样需要去掉
2. **函数归约** (Functional Reduction) : 保证规则头中输入和输出的变元在规则体中是相连的, 多余的变量可以删除。例如
$$p(X, Y) \leftarrow q(X, Z), \textcolor{brown}{r}(A, B), s(Z, Y).$$
3. **利用负样例**: 不停地尝试删除 RLGG 中的文字, 除非删除操作导致它覆盖负样例。



RLGG 的问题

- I. 埃尔布朗模型 $M_P = B$ 无穷大，导致 RLGG 的长度无穷。
2. 即便在 B 有限的情况下， $lgg(C_1, C_2)$ 的大小也是 $|C_1| \cdot |C_2|$ 。那么 $rlgg_B(\{e_1, e_2, \dots, e_n\})$ 的大小上界为 $|B|^n + 1$ 。
3. 当目标概念必须通过多条规则描述时，RLGG 的表达能力不够。



S. H. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, pp. 368-381, Tokyo, Japan, 1990.

为了改进RLGG，GOLEM对它进行了以下改进：

1. **约束背景知识大小**：通过直接推论算子 T_P^n ，提出了 **h -easy models** 概念，即利用 B 经过 h 次直接推论（归结）得到的 T_B^h 作为饱和规则用的规则体。我们记这个模型为 $M_B(h)$ 。
2. **约束背景知识结构**：要求所有背景知识中的一阶规则均为**生成式规则**（Generative rule），即规则头的变元是规则体变元子集的规则。
 - » 这么做的好处是对任意 $h \in \mathbb{N}$ 有 $M_B(h)$ 有穷。
3. **约束饱和规则结构**： ij -决定性 (ij -determinacy)



i, j -决定性

决定项 (determinate terms)：对于 B 中的规则 $A \leftarrow B_1, \dots, B_m, B_{m+1}, \dots, B_n$, 令 t 为 B_{m+1} 中的项。我们称 t 对于 B_{m+1} 是决定的，当且仅当对于任意一个令 $\{B_1, \dots, B_m\}\theta \subseteq M_B$ 且令 $A\theta$ 成为 ground atom 时的替换 θ , t 中的剩余变元仅存在唯一一个替换 δ 令 $B_{m+1} \in M_B$ 。

- > 简而言之, t 可以看作 B_{m+1} 中谓词关于其他被替换的项 s_1, \dots, s_j 的函数, 我们称 t 的度 (degree) 为 j 。
- > 根据变量之间的依赖性, 我们称决定项只由规则头变元确定的文字深度 (depth) 为 I , 否则其深度等于所有确定它决定项的其他文字的最大深度加 I 。



i, j -决定性

```
multiply(A,B,C) :-  
    decrement(B,D),  
    multiply(A,D,E),  
    plus(A,E,C).
```

- › D 和 E 均为决定项；
- › D 的度为 1, decrement(B,D) 深度为 1；
- › E 的度为 2, multiply(A,D,E) 深度为 2；
- › plus(A,E,C) 深度为 1。



i, j -决定性

i, j -决定性 (ij -determinacy) 一个逻辑程序中的子句 $A \leftarrow B_1, \dots, B_m, B_{m+1}, \dots, B_n$ 是 ij -决定的当且仅当：

1. 其所有文字中的变元最大度为 j 。
2. 若 $n = 0$, 则它是 $0j$ -决定的。
3. 若 $A \leftarrow B_1, \dots, B_m$ 中所有文字最大深度为 $i - 1$, 即 $(i - 1)j$ -决定的, 且所有 B_{m+1}, \dots, B_n 中的项均为决定项。



i, j -决定性

1, 1-决定的:

```
class(A,mammal) :- has_milk(A).
```

1, 2-决定的:

```
double(A,B) :- plus(A,A,B).
```

2, 1-决定的:

```
grandfather(A,B) :- father(A,C), father(C,B).
```



GOLEM 的 RLGG 大小

定理：对于逻辑程序 B ，若其中有 m 个谓词，它们的最大元数（arity）为 f 。 $\{e_1, \dots, e_n\}$ 为一组 ground atoms，它们的最小一般泛化 $lgg(\{e_1, \dots, e_n\})$ 至多只包含 t 个文字。那么，一个关于 $M_B(h)$ 、 i, j -决定的 RLGG 的规则体中至多包含 $O((tfm)^{j^i})$ 个文字。

- › 与不受限的 RLGG 大小 ($O(|M_B(h)|^n)$) 相比，这个上界**与样本量、模型大小无关**。
- › 看起来这个上界很大，但对于很多任务来说 $i = j = 2$ 已经足够。
 - » 例如快排、列表append运算、member函数等等。



GOLEM 算法

- I. 随机抽取 s 对正样例。
2. 对每一个样本对 S 生成 $rlgg(S)$ ，并选择 RLGG 覆盖正样例最多的样例对 S_{best} 。
3. 当覆盖正样例个数增加的情况下：
 - I. 采样 s 个正样例
 2. 找到一个 e_{best} 令 $rlgg(S_{best} \cup \{e_{best}\})$ 覆盖最多正样例
 3. $S_{best} = S_{best} \cup \{e_{best}\}$
4. 利用逻辑归约和负样例对最后生成的 RLGG 做剪枝。



符号学习简介 · 归纳逻辑程序设计（一）

1. 归纳逻辑程序设计
2. 相对最小一般泛化 (RLGG)
3. 逆归结 (Inverse Resolution)



特化与泛化

特化	泛化
演绎	归纳
变量代换	最小一般泛化



另一种归纳方式

特化	泛化
演绎	归纳
变量代换（合一）	最小一般泛化（逆合一，anti-unification）
归结	逆归结



逆归结

输入：

```
C = child(X,Y) :- father(X,Y).  
C2 = parent(X,Y) :- father(X,Y).
```

输出：

```
C1 = child(X,Y) :- parent(X,Y).
```



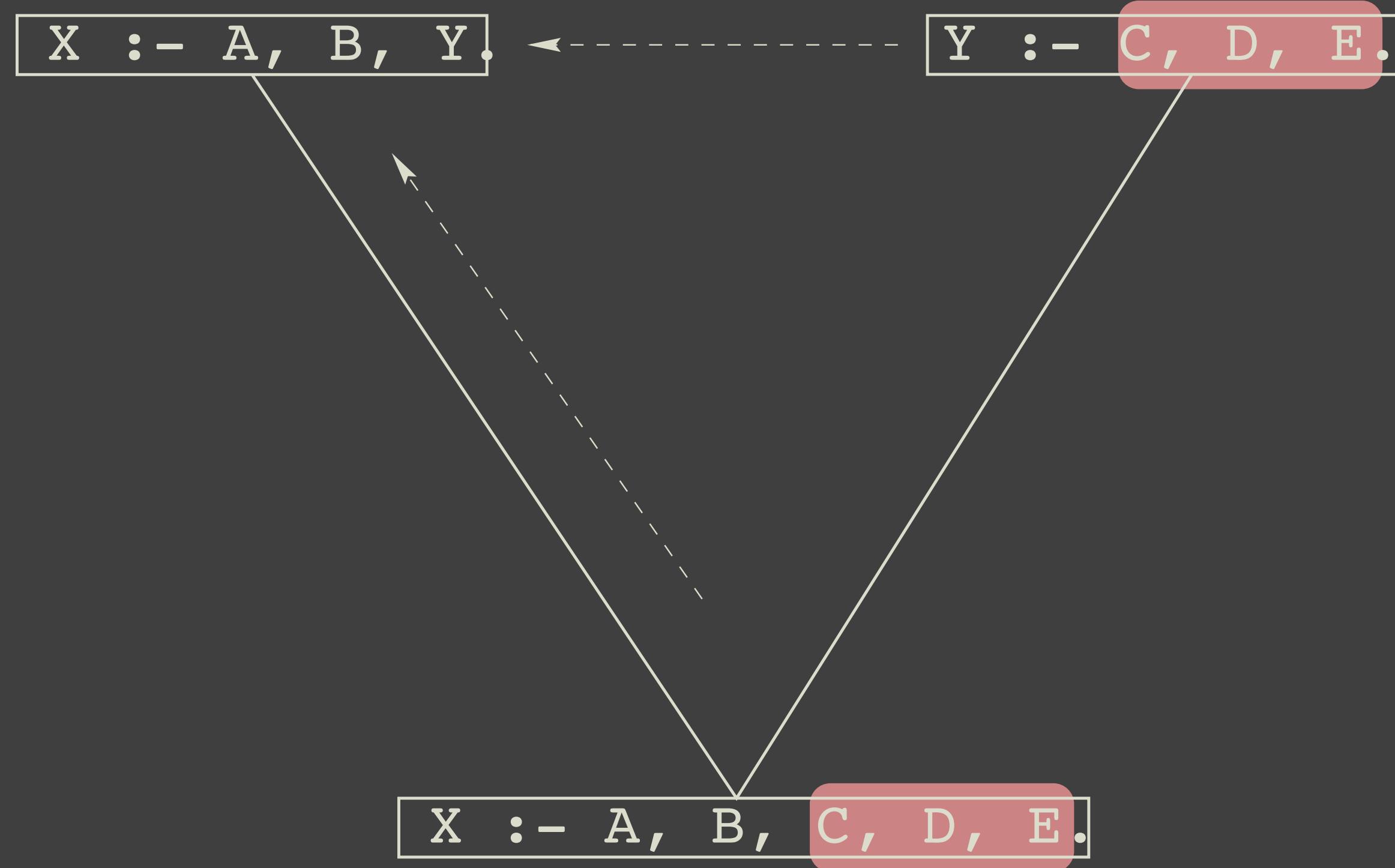
S. H. Muggleton. Duce, an oracle based approach to constructive induction. In *IJCAI-87*, pp 287-292, 1987.

DUCE是一个通过逆归结进行命题规则学习的方法，它包含下面几种归纳操作：

- > **V型操作**
 - » 吸收 (absorption)
 - » 辨识 (identification)
- > **W型操作**
 - » 内构 (inter-construction)
 - » 互构 (intra-construction)
- > **非逆归结操作**
 - » 截断 (truncation)
 - » 二分 (dichotomization)



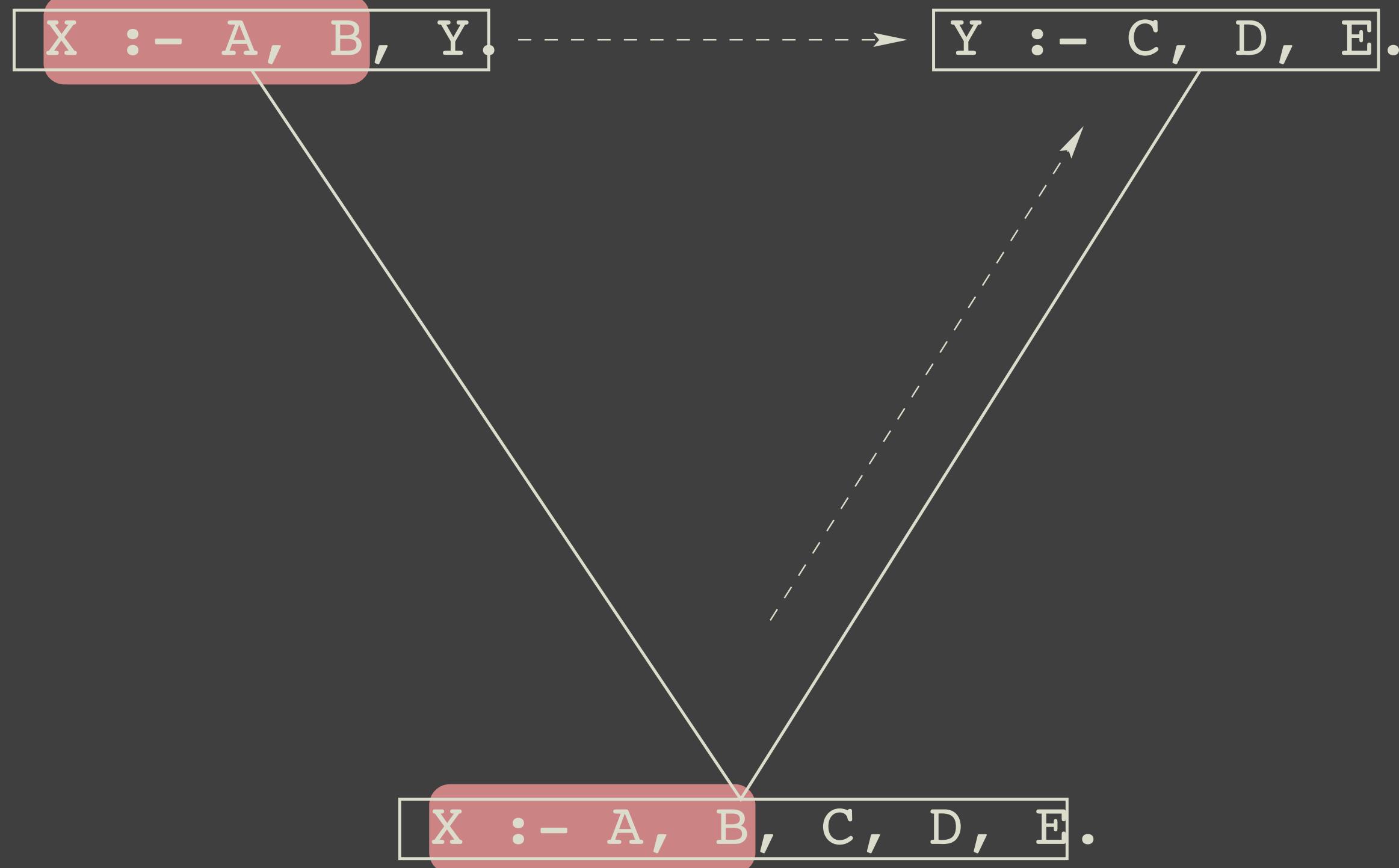
吸收



- › 替换子句中的部分文字，并生成一个新概念（谓词）`Y`用于定义它们
- › 对`x`的泛化：当存在其他规则同样定义`Y`时，同样能被用于证明`x`



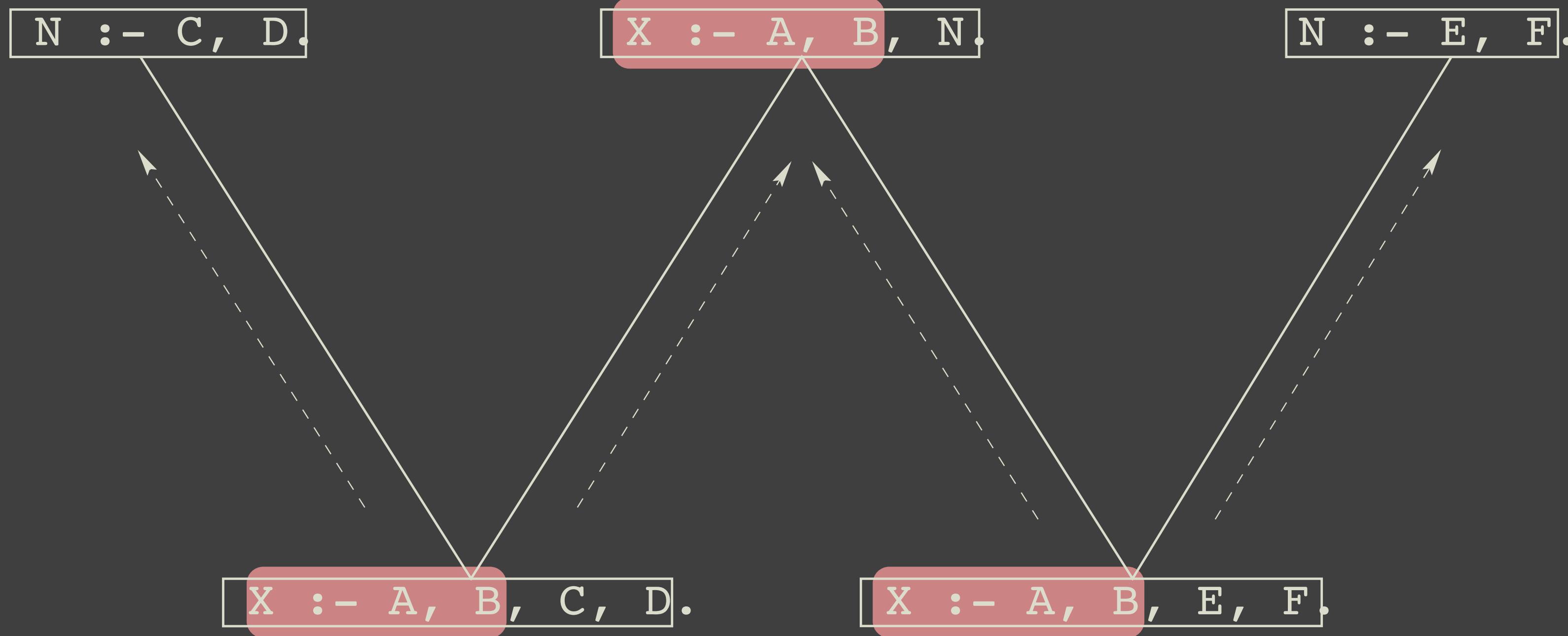
辨识



- > 构造一条新规则，把 x 中单独的文字 y 分离出来
- > 对程序的其他部分泛化：若关于 y 的规则体在其他子句中也出现，则同样可以用 y 替换



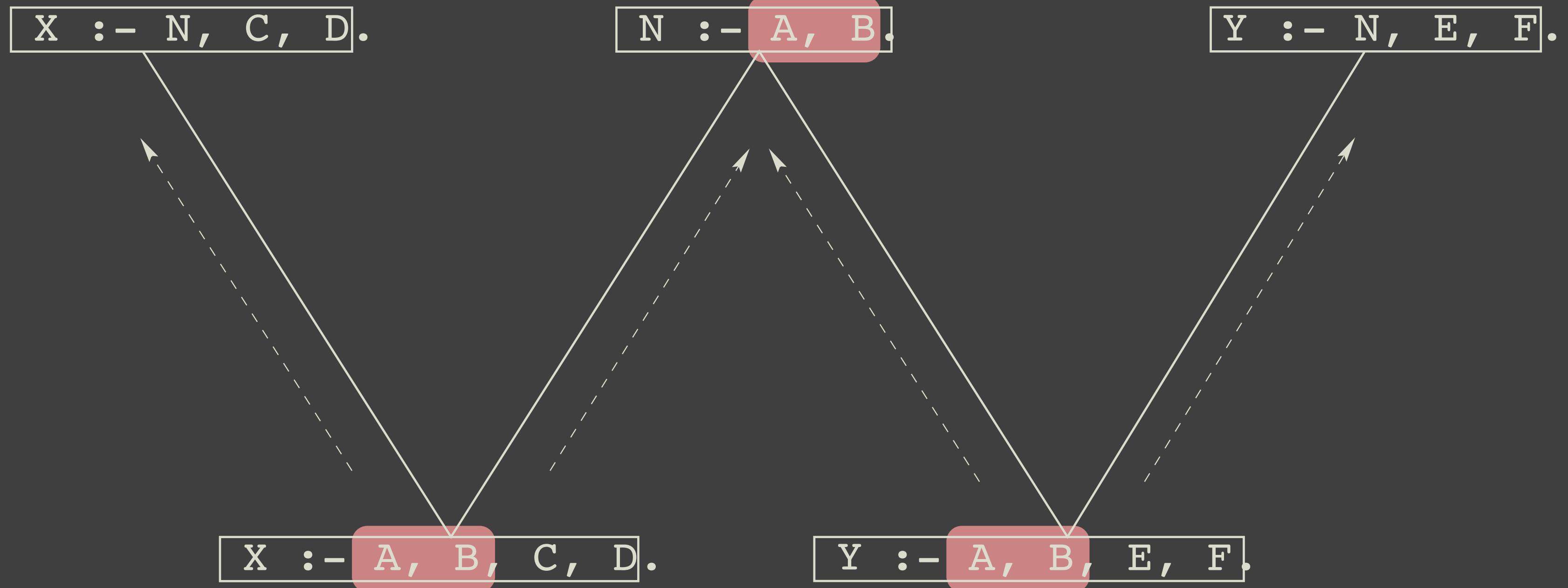
内构



- > 找到关于同一概念两条规则中的共同部分，提取其中不同的文字作为子概念 N
- > 无法泛化程序，但可以简化它



互构



- > 找到关于不同概念两条规则中的共同部分作为子概念 N
- > 同样无法泛化程序，但可以简化



截断

截断 (truncation) : 类似于LGG, 由

```
X :- A, B, C, D.  
X :- A, B, E, F.
```

得到

```
X :- A, B.
```



二分

二分 (dichotomization) : 由规则不同的部分构造一个否定概念，输入

```
X      :- A, B, C, D.  
Not_X :- A, B, E, F.
```

得到

```
X      :- A, B, N.  
Not_X :- A, B, Not_N.  
N      :- C, D.  
Not_N :- E, F.
```



区分性假设

DUCE要求归纳出的子句与另一个归结子句没有公共文字，即它们是有区分性的（separable）。

这会导致：

```
A :- B, C.
```

无法从

```
A :- D, E, F.  
B :- D, E.  
C :- D, F.
```

中归纳出。而DUCE只可能归纳出下面二者之一

```
A :- B, F.  
A :- C, E.
```



DUCE算法

输入：一个命题逻辑程序，其中每个子句描述一个样例。

输出：一个精化后的命题逻辑程序，能够区分正负样例。

算法：

1. 找到所有可逆归结的操作。
2. 选择最能够缩小原命题规则长度的操作，询问用户（oracle）是否需要执行。
3. 若用户选择执行，则将归结商替换为归纳得出的结果。
4. 执行步骤1，直到无法继续缩小任一规则长度。



DUCE的例子

```
blackbird(blockhead) :- beak=t, color=black, legs=2, tail=t, wings=t.  
  
chimp(maggie) :- colour=brown, hairy=t, legs=2, tail=t, wings=f.  
  
eagle(egg) :- beak=t, colour=golden, legs=2, tail=t, wings=t.  
  
elephant(adult) :- colour=grey, legs=4, size=big, tail=t, trunk=t, wings=f.  
  
elephant(baby) :- colour=grey, legs=4, size=small, tail=t, trunk=t, wings=f.  
  
falcon(flap) :- beak=t, colour=brown, legs=2, size=big, tail=t, wings=t.  
  
gorilla(ronnie) :- colour=black, hairy=t, legs=2, tail=f, wings=f.  
  
lemur(alone) :- colour=grey, legs=2, tail=t, wings=f.  
  
man(harry) :- colour=brown, hairy=f, legs=2, size=big, tail=f, wings=f.  
  
man(clap) :- colour=pink, hairy=f, legs=2, size=small, tail=f, wings=f.  
  
sparrow(sparky) :- beak=t, colour=brown, legs=2, size=small, tail=t, wings=t
```



DUCE的例子

```
!- induce.  
    TRUNCATION - (-12)  
    Is (elephant :- legs=4) a valid rule? (y/n/i) n  
    TRUNCATION - (-11)  
    Is (elephant :- legs=4, wings=f) a valid rule? (y/n/i) n  
    TRUNCATION - (-11)  
    Is (elephant :- legs=4, trunk=t) a valid rule? (y/n/i) y  
!- induce.  
    TRUNCATION - (-9)  
    Is (man :- hairy=f, legs=2, tail=f, wings=f) a valid rule? (y/n/i) y  
!- induce.  
    INTER-CONSTRUCTION - (-1)  
    Rule:  
        (? :- legs=2, wings=f)  
        What shall I call <?>? (name/n/i) primate  
!- induce.  
    INTER-CONSTRUCTION - (-7)  
    Rule:  
        (? :- beak=t, legs=2, tail=t, wings=t)  
        What shall I call <?>? (name/n/i) bird  
!- induce
```

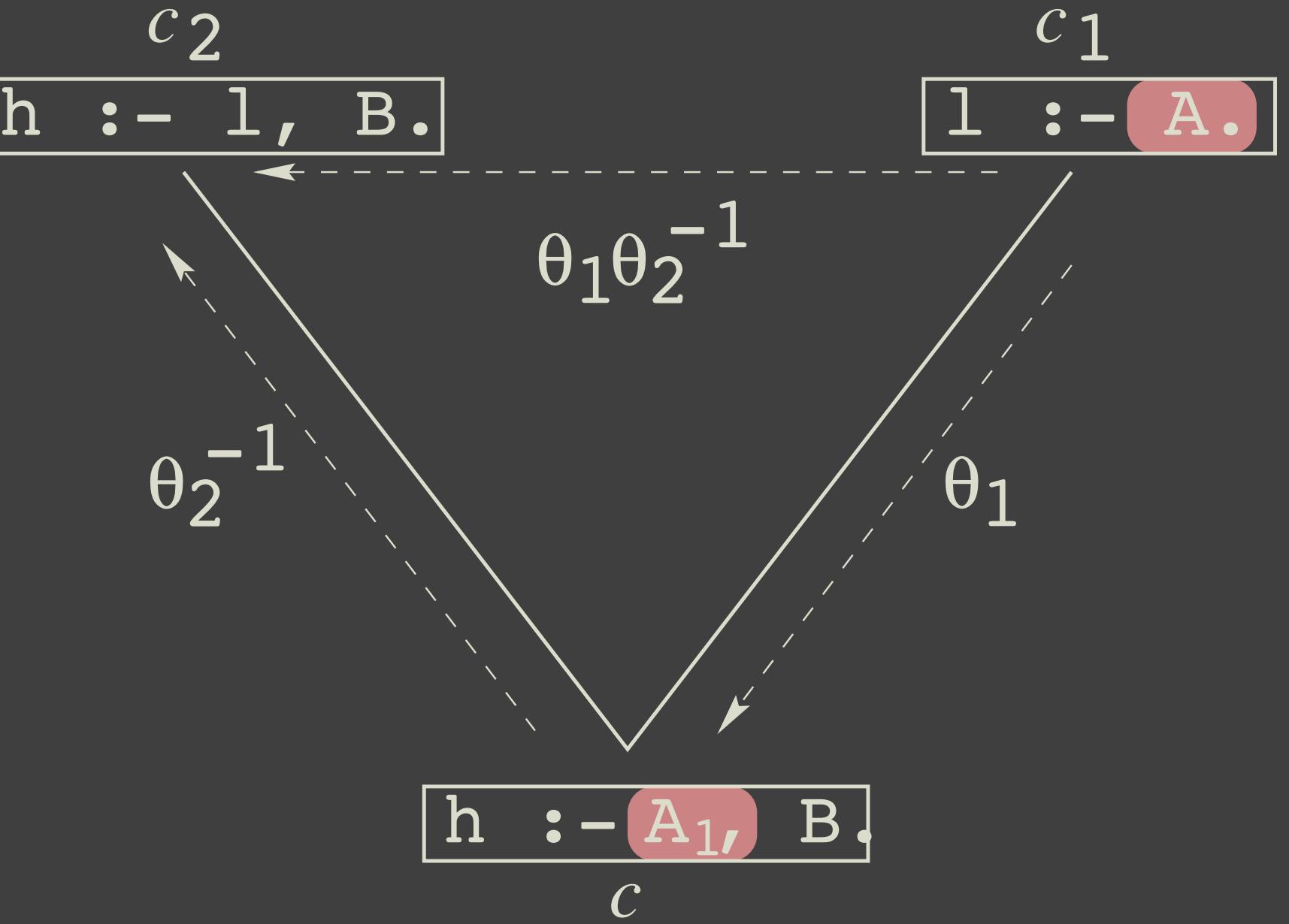


DUCE的例子

```
% 归纳得出的结果:  
bird :- beak=t, legs=2, tail=t, wings=t.  
% covers [blockhead, egg, flap, sparky]  
blackbird :- bird, colour=black.  
% covers [blockhead]  
chimp :- primate, colour=brown, hairy=t, tail=t.  
% covers [maggie]  
eagle :- bird, colour=golden.  
% covers [egg-the-eagle]  
elephant :- legs=4, trunk=t.  
% covers [adult, baby]  
falcon :- bird, colour=brown, size=big.  
% covers [flap]  
gorilla :- primate, colour=black, hairy=t, tail=f.  
% covers [ronnie]  
lemur :- primate, colour=grey, tail=t.  
% covers [lemur]  
man :- primate, hairy=f, tail=f.  
% covers [clap, harry]  
primate :- legs=2, wings=f.  
% covers [maggie, clap, ronnie, harry, alone]
```



一阶逆归结（吸收）

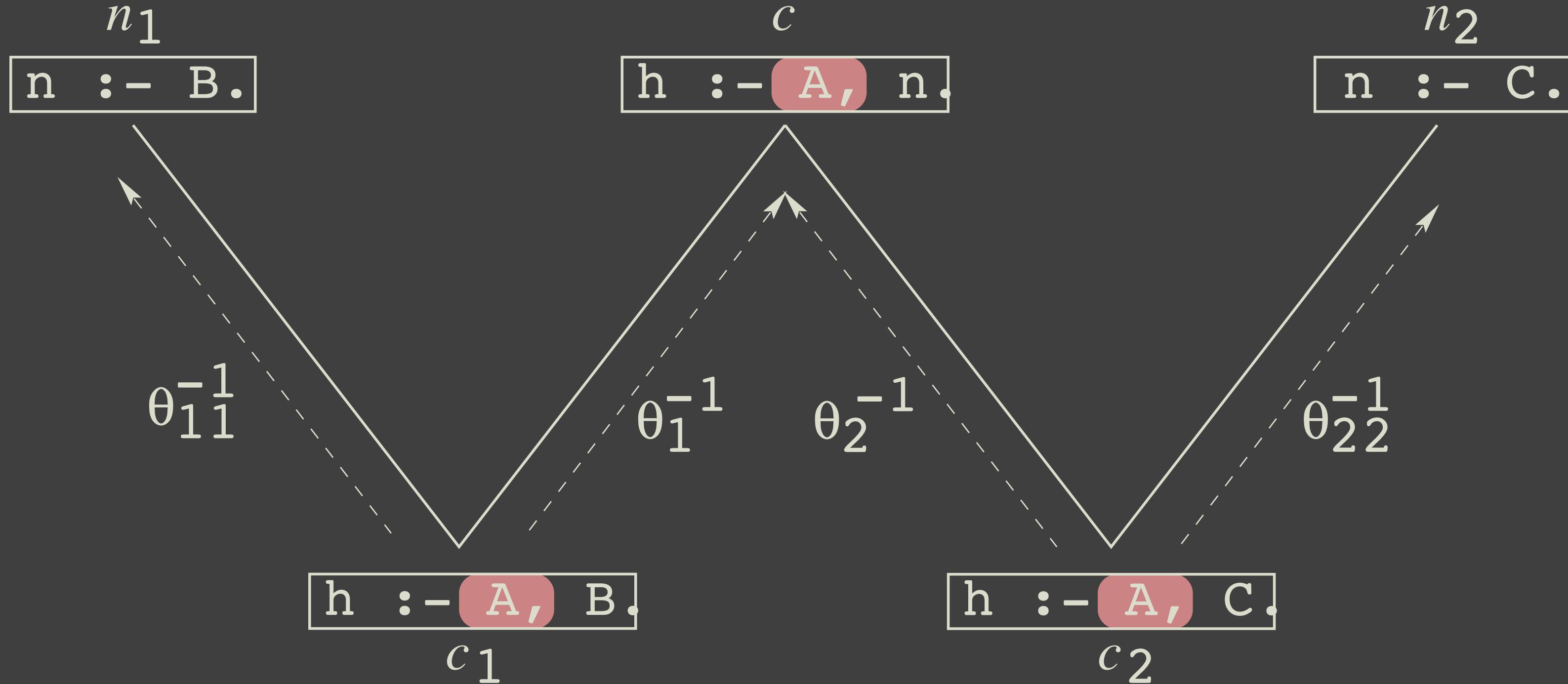


$$c_2 = (c - (c_1 - \{l\}) \cup \{\neg(l)\})\theta_1\theta_2^{-1}$$

- > 子句 c 包含一个 c_1 规则体的特化 ($A \preceq A_1$)
- > 对 h 的泛化：把 A_1 替换成 c_1 的规则头。



一阶逆归结（内构）



- 一阶内构在选择变元替换构造新谓词 n 时只需要保留相关参数，即同时出现在 n 和 $c \setminus n$ 中的参数。



一阶吸收的不可判定性

一阶吸收的过程需要决定以下内容：

1. 归结项 ℓ 的选择可能多样。
2. 替换 θ_1 : c_1 的规则头中某些变量可能未出现在规则体 A 中, 与 A_1 合一时的 MGU 可能过于一般。
3. 替换 θ_2^{-1} : $(c - (c_1 - \{l\}) \cup \{\neg(l)\})$ 的结果同样不唯一。
4. 若忽略区分性假设, A 与 B 可有多种交集。



S. H. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pp. 339-352, 1988.

为了缩小一阶逆归结的搜索空间，CIGOL对几种逆归结操作进行约束：

- > **吸收**: c_1 只能是原子公式（ A 为空）
- > **内构**: 可以同时对超过 2 个公式进行内构
- > **截断**: 计算两条公式的LGG



小结

符号学习

归纳逻辑程序设计（一）

<https://daiwz.net>



小结

- > 归纳逻辑程序设计
 - » 巨大的搜索空间
- > 演绎vs归纳
 - » 合一与LGG
 - » LGG的存在性
 - » RLGG和饱和规则
 - » 归结与逆归结
 - » 命题逻辑中的逆归结
 - » 一阶逻辑中的逆归结
- > 两种方法的优势与缺陷
 - » 逆归结：谓词发明，但不可判定且速度非常慢
 - » RLGG：速度比逆归结快，但是？