

知识表示与推理 II

吉建民

USTC

`jianmin@ustc.edu.cn`

2025 年 4 月 8 日

Used Materials

Disclaimer: 本课件采用了部分网络资源

Table of Contents

情形演算: Situation Calculus

Golog

非单调推理: Nonmonotonic Reasoning

回答集编程: Answer Set Programming

稳定模型语义

ASP 基础

ASP 编程

Situation Calculus 背景

- ▶ 用逻辑的方法刻画 dynamical systems。需要考虑的问题：
 - ▶ 刻画行动效果的因果规则，行动执行条件；
 - ▶ 外部行动和事件；
 - ▶ 并发执行；
 - ▶ 复杂行动和程序；
 - ▶ 感知行动及其对 agent's mental state 的影响；
 - ▶ Agent 的 BDI 状态；
 - ▶ 不确定行动执行和效果，观察有噪音；
 - ▶ 离散时间或连续时间；
 - ▶ 行动规划，决策，信念修正；
 - ▶ 监控执行过程，识别失败，并从失败中恢复；等
- ▶ 从而可以通过逻辑推理处理，system control, simulation, and analysis.

使用逻辑来规划

- ▶ 需要用逻辑来刻画：
 - ▶ 世界、环境, world
 - ▶ 目标, goal
 - ▶ 行动如何影响世界, how actions change the world
- ▶ 行动对世界的影响: 新的事实变为真, 旧的事实变为假
- ▶ 基本概念:
 - ▶ 状态 (state): 一个系统在某时刻的情况
 - ▶ 行动 (action): 状态空间上的部分映射
- ▶ 情形演算: 特殊的一阶逻辑系统 (增加少量的二阶逻辑公理)

基本语法和预期解释

- ▶ 情形常元 (记为 $S_0, S_1, \dots$) 和情形变元 (记为 $s_0, s', s, s_1, \dots$), 用作状态的标识 (名字)
- ▶ 行动函数符号, 用于表示原子行动 (不可再分解的行动)。一个行动函数代表一个行动模式, 函数自变量为此行动主体 (和受体)
 - ▶ 例如: $move(agt, x, y, z)$ 个体变元 agt 代表这个行动的主体, 变元 x, y, z 为受体。表示 agt 将 x 从 y 上移到 z 上
- ▶ 行动复合函数 do , 二元, 将行动个体和情形映射为情形。 $do(a, s)$ 为在 s 上执行 a 的结果状态, 从而间接将行动解释为状态变换。 do 嵌套使用表示行动序列
 - ▶ 例如: $do(a_2, do(a_1, S_0))$ 表示在 S_0 中顺序执行 a_1, a_2 的结果状态
- ▶ 变式 (fluent) 代表其值依赖于情形的谓词和函数, 同时允许普通谓词和函数

情形演算基本假设

- ▶ “世界”的任何变化都是由有名行动产生的
- ▶ 每一行动有唯一的名字
- ▶ 任何情形 S ，只要给定 $a_1, \dots, a_n$ ，则后继状态

$$do(a_1, S)$$

$$do(a_2, do(a_1, S))$$

...

$$do(a_n, do(a_{n-1}, \dots, do(a_1, S) \dots))$$

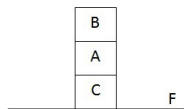
都是唯一确定的。因此，一个行动序列代表一个“历史”。

状态描述

- ▶ 采用逻辑表示法，描述状态、状态集

- ▶ 例如：（积木世界）图示状态 S_0

$$(F1) \quad on(B, A, S_0) \wedge on(A, C, S_0) \wedge \\ on(C, F, S_0) \wedge clear(B, S_0)$$



- ▶ $clear(x, s)$: 在 s 中 x 上有足够的（剩余）空间放置一块积木

- ▶ 类似表示“领域知识”（包括“领域不变性”）

$$(R1) \quad \forall s \, clear(F, s)$$

$$(R2) \quad \forall xys (on(x, y, s) \wedge \neg(y = F)) \supset \neg clear(y, s)$$

- ▶ 还可以方便的描述条件（目标等）

- ▶ 例如，给定目标“ A 在 F 上”表示为 $on(A, F, S)$

效果公设

- ▶ 效果公设：描述变式被行动改变的情况，是一个蕴含式，前、后件分别描述行动的前提条件和效果，又分为“正效果”（从假变真）和“负效果”（从真变假）
- ▶ 例如：给定行动“将 x 从 y 上移到 z 上”，用 $move(x, y, z)$ 表示：

$$(R3) \quad on(x, y, s) \wedge clear(x, s) \wedge clear(z, s) \wedge \neg(x = z) \supset on(x, z, do(move(x, y, z), s))$$

$$(R4) \quad on(x, y, s) \wedge clear(x, s) \wedge clear(z, s) \wedge \neg(x = z) \supset \neg on(x, y, do(move(x, y, z), s))$$

$$(R5) \quad on(x, y, s) \wedge clear(x, s) \wedge clear(z, s) \wedge \neg(x = z) \wedge \neg(y = z) \supset clear(y, do(move(x, y, z), s))$$

$$(R6) \quad on(x, y, s) \wedge clear(x, s) \wedge clear(z, s) \wedge \neg(x = z) \wedge \neg(z = F) \supset \neg clear(z, do(move(x, y, z), s))$$

框架公设

- ▶ S 经过行动 a 变为 S' , 依效果公设, 可以推出 S' 相对于 S 改变了的所有谓词变式
 - ▶ 例中, 令 $a = \text{move}(B, A, F)$, $S' = \text{do}(a, S_0)$, 依 (R3 - 6) 可推出: $\text{on}(B, F, S')$, $\neg \text{on}(B, A, S')$, $\text{clear}(A, S')$
- ▶ 再用领域知识, 可推出 S' 相对于 S 不变的变式, 例如:
 $\text{clear}(F, S')$
- ▶ 一般的, S' 总有一些未改变的变式无法由效果公设和领域知识推出 它们的不变不是由领域不变性决定的, 说明领域知识加效果公设不足以描述行动未改变的变式。“直接”办法是引入框架公设

$$(R7) \quad \text{on}(u, v, s) \wedge \neg(u = x) \supset \text{on}(u, v, \text{do}(\text{move}(x, y, z), s))$$

$$(R8) \quad \neg \text{on}(u, v, s) \wedge \neg(v = y) \supset \\ \neg \text{on}(u, v, \text{do}(\text{move}(x, y, z), s))$$

$$(R9) \quad \text{clear}(u, s) \wedge \neg(u = z) \supset \text{clear}(u, \text{do}(\text{move}(x, y, z), s))$$

$$(R10) \quad \neg \text{clear}(u, s) \wedge \neg(u = y) \supset \\ \neg \text{clear}(u, \text{do}(\text{move}(x, y, z), s))$$

情形演算中框架问题

- ▶ 框架问题 (Frame Problem): 每次状态改变只影响少量变式, 大量变式不变, 但仍需刻画。
- ▶ 情形演算中解决方案:
 - ▶ 只处理 deterministic actions without ramifications 的情况。
 - ▶ 使用 closed world assumption, 假设导致变式改变的所有行动 (因素) 都已经被刻画出来, 没说明的则保持不变。
- ▶ 使用 successor state axiom, 构造过程:
 - ▶ 已知 effect axioms:

$$\begin{aligned}\gamma_F^+(\vec{x}, a, s) &\supset F(\vec{x}, do(a, s)), \\ \gamma_F^-(\vec{x}, a, s) &\supset \neg F(\vec{x}, do(a, s)).\end{aligned}$$

- ▶ 根据 CWA 得到 Explanation closure axioms:

$$\begin{aligned}F(\vec{x}, s) \wedge \neg F(\vec{x}, do(a, s)) &\supset \gamma_F^-(\vec{x}, a, s), \\ \neg F(\vec{x}, s) \wedge F(\vec{x}, do(a, s)) &\supset \gamma_F^+(\vec{x}, a, s).\end{aligned}$$

情形演算中框架问题 (con't)

- ▶ 上述四个 axioms 可以用一个 successor state axiom 表达：

$$F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee [F(\vec{x}, s) \wedge \neg \gamma_F^-(\vec{x}, a, s)].$$

- ▶ 对于 function fluent f .

$$f(\vec{x}, do(a, s)) = y \equiv \gamma_f(\vec{x}, y, a, s) \vee [f(\vec{x}, s) = y \wedge \neg(\exists y') \gamma_f(\vec{x}, y', a, s)].$$

- ▶ 当然整个过程需要给出：Unique names axioms for actions.

情形演算中条件问题

- 谓词 $Poss$ 用来刻画行动的前提，例如：

$$Poss(pickup(r, x), s) \supset [(\forall z) \neg holding(r, z, s)] \wedge \neg heavy(x) \\ \wedge nextTo(r, x, s),$$

$$Poss(repair(r, x), s) \supset hasGlue(r, s) \wedge broken(x, s).$$

- 条件问题 (Qualification Problem)：我们很难刻画行动能执行的所有前提。
- 情形演算中解决方案：
 - 通过 action precondition axioms 来刻画行动前提：

$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s),$$

其中 $\Pi_A(\vec{x}, s)$ 为一阶公式，不出现 do 。

- 在 SC 中选择 ignore all the “minor” qualifications，而将 “important” qualifications 作为充分必要条件。

情形演算简介

- Foundational Axioms for Situations:

$$\begin{aligned}do(a_1, s_1) &= do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2, \\(\forall P). P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] &\supset (\forall s)P(s), \\ \neg s \sqsubset S_0, \\ s \sqsubset do(a, s') &\equiv s \sqsubset s'.\end{aligned}$$

- 一个 basic theories of Action 为:

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}.$$

- 其中 Σ 为上面的四条 foundational axioms,
- $\mathcal{D}_{ss}$ 为相关的 successor state axioms,
- $\mathcal{D}_{ap}$ 为相关的 action precondition axioms,
- $\mathcal{D}_{una}$ 为相关的 unique names axioms for actions,
- $\mathcal{D}_{S_0}$ 为刻画 S_0 的公式。

情形演算应用

- ▶ Situation Calculus 在 Planning 上应用:
 - ▶ The planning problem: Given an axiomatized initial situation, and a goal statement, find an action sequence that will lead to a state in which the goal will be true.
 - ▶ 在 SC, plans can be synthesized as a side-effect of theorem proving.
 - ▶ $Axioms \vdash (\exists s) G(s)$.
 - ▶ $executable(s) =_{df} (\forall a, s^*). do(a, s^*) \sqsubseteq s \supset Poss(a, s^*)$.
 - ▶ σ is a plan for G iff $\mathcal{D} \models executable(\sigma) \wedge G(\sigma)$.
- ▶ SC 在 Database Transactions 上应用:
 - ▶ 可以考虑 projection problem in Database, 即推理计算执行某些行动后的效果。

Table of Contents

情形演算：Situation Calculus

Golog

非单调推理：Nonmonotonic Reasoning

回答集编程：Answer Set Programming

稳定模型语义

ASP 基础

ASP 编程

Golog 背景

- ▶ Complex actions and procedures 用来 high level control of robots and industrial processes, intelligent software agents, discrete event simulation, etc.
- ▶ Golog 目的: provide a situation calculus based account of complex actions and procedures.
- ▶ Golog is a program language based on the situation calculus which allows for expressing and reasoning with complex actions and procedures.
- ▶ a Golog program δ is based on Situation Calculus.
- ▶ $Do(\delta, s, s')$ will be macro-expand to a SC formulas that says that it is possible to reach s' from s by executing a sequence of actions specified by δ .

Golog $\equiv$ complex actions

- ▶ Primitive actions: a

$$Do(a, s, s') =_{df} Poss(a[s], s) \wedge s' = do(a[s], s)$$

- ▶ Test actions: $\phi?$

$$Do(\phi?, s, s') =_{df} \phi[s] \wedge s = s'$$

- ▶ Sequence: $\delta_1; \delta_2$

$$Do(\delta_1; \delta_2, s, s') =_{df} (\exists s''). Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$$

Golog $\boxplus$ complex actions (con't)

- ▶ Nondeterministic choice of two actions: $\delta_1 \mid \delta_2$

$$Do(\delta_1 \mid \delta_2, s, s') =_{df} Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$$

- ▶ Nondeterministic choice of action arguments: $(\pi x)\delta(x)$

$$Do((\pi x)\delta(x), s, s') =_{df} (\exists x) Do(\delta(x), s, s')$$

- ▶ Nondeterministic iteration:

$$\begin{aligned} Do(\delta^*, s, s') =_{df} & (\forall P). \{ (\forall s_1) P(s_1, s_1) \\ & \wedge (\forall s_1, s_2, s_3) [Do(\delta, s_1, s_2) \wedge P(s_2, s_3) \supset P(s_1, s_3)] \} \supset P(s, s') \end{aligned}$$

Golog 的 program

- ▶ Conditions: $\text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2 \text{ endIf} =_{df} [\phi?; \delta_1] \mid [\neg\phi?; \delta_2]$.
- ▶ While-loops: $\text{while } \phi \text{ do } \delta \text{ endWhile} =_{df} [\phi?; \delta]^*; \neg\phi?$.
- ▶ Predicate P : $\text{Do}(P(t_1, \dots, t_n), s, s') =_{df} P(t_1[s], \dots, t_n[s], s, s')$.
- ▶ Program:
 $\text{proc } P_1(\vec{v}_1) \delta_1 \text{ endProc}; \dots; \text{proc } P_n(\vec{v}_n) \delta_n \text{ endProc}; \delta_0$

$$\text{Do}(\{\text{proc } P_1(\vec{v}_1) \delta_1 \text{ endProc}; \dots; \text{proc } P_n(\vec{v}_n) \delta_n \text{ endProc}; \delta_0\}, s, s') =_{df} \\ (\forall P_1, \dots, P_n). [\bigwedge_{i=1}^n (\forall s_1, s_2, \vec{v}_i). \text{Do}(\delta_i, s_1, s_2) \supset P_i(\vec{v}_i, s_1, s_2)] \supset \text{Do}(\delta_0, s, s')$$

- ▶ 其中 $P_i(\vec{v}_i)$ 相当于子程序声明， δ_i 为子程序主体。 δ_0 为当前程序的主体。
- ▶ 此定义为二阶公式，含义为：When $P_1, \dots, P_n$ are the smallest binary relations on situations that are closed under the evaluation of their procedure bodies $\delta_1, \dots, \delta_n$, then any transition (s, s') obtained by evaluation the main program δ_0 is a transition for the evaluation of the program.

Golog 的性质

- ▶ Golog 程序不能用 planning 方式计算。
 $Axioms \models (\exists \delta, s). Do(\delta, S_0, s) \wedge Goal(s)$ 是不允许的, 因为 δ 对应 SC 中的公式, 而不是里面的 object (项)。 δ 可能的组合方式也是无穷的。
- ▶ Golog 程序可以证明很多性质:
 - ▶ Correctness: $Axioms \models (\forall s). Do(\delta, S_0, s) \supset P(s)$.
 - ▶ Termination: $Axioms \models (\exists s) Do(\delta, S_0, s)$.
- ▶ 总的来说, 现有定义方式 is well-suited to applications where a program δ is given and the job is to prove it has some property.
- ▶ 运行 Golog program 的主要任务是: 找到一个状态 s , 使得 s 是 program 的 terminating situation, 即, 从 S_0 出发的行动序列。即: $Axioms \models (\exists s) Do(\delta, S_0, s)$.

Golog 的性质 (con't)

- ▶ Golog program 一般对应 SC 中 second-order sentence.
- ▶ 利用 Prolog 可以写出一个 Golog interpreter.
- ▶ 可以用 Golog 定义 complex action schemas, without specifying in detail how to perform these actions. 例如:

while $[(\exists block) ontable(block)]$ *do* $(\pi b) remove(b)$ *endWhile*

- ▶ Classical planning 可以表达为一个 Golog program:
while $\neg Goal$ *do* $(\pi a)[Appropriate(a)?; a]$ *endWhile*

Table of Contents

情形演算：Situation Calculus

Golog

非单调推理：Nonmonotonic Reasoning

回答集编程：Answer Set Programming

稳定模型语义

ASP 基础

ASP 编程

非单调推理

- ▶ 经典逻辑（即单调推理）：以一个无矛盾的公式系统 Γ 为基础。如果 $\Gamma \models A$ ，则 $\Gamma \cup \Gamma' \models A$
- ▶ 非单调推理：人类思维本质上是非单调的。由于对客观条件掌握的不充分，当有新的事实被认识时，可能导致原来的某些结论被推翻。 $\Gamma \models A$ ，但可能 $\Gamma \cup \Gamma' \not\models A$
- ▶ 非单调推理应用：
 - ▶ 封闭世界假设（CWA）：当前不是已知的事物都假定为假
 - ▶ 失败即否定（Negation As Failure）：推不出的假定为假
 - ▶ 框架问题（Frame Problem）：没提到的假设不受影响
 - ▶ 分支问题（Ramification Problem）：行动的间接效果
 - ▶ 常识推理（Commonsense Reasoning）

非单调推理例子

- ▶ 鸟会飞，特威蒂是鸟，所以，特威蒂会飞。
- ▶ 鸟会飞，特威蒂是鸟，但特威蒂不会飞，所以，特威蒂不会飞。
- ▶ 鸟会飞，企鹅不会飞，特威蒂是鸟，特威蒂是企鹅，所以，特威蒂？(会飞还是不会飞？)
- ▶ 企鹅是鸟，鸟会飞，企鹅不会飞，特威蒂是鸟，特威蒂是企鹅，所以，特威蒂不会飞。
- ▶ 贵格会教徒是和平主义者，共和党人不是和平主义者，尼克松是贵格会教徒，所以，尼克松是和平主义者。
- ▶ 贵格会教徒是和平主义者，共和党人不是和平主义者，尼克松是贵格会教徒，尼克松是共和党人，所以，尼克松是？(是不是和平主义者？)

主要工作

- ▶ 非单调推理的主要工作:

- ▶ McCarthy 的“限定理论” (Circumscription): ϕ :
 $Bird(x) \wedge \neg Abnormal(x) \supset Flies(x)$, $CIRC(\phi, Abnormal)$.
- ▶ Reiter 的“缺省逻辑” (Default Logic):

$$\frac{\alpha(x) : \beta_1(x), \dots, \beta_m(x)}{\gamma(x)}$$

- ▶ Moore 的“自认知逻辑” (Autoepistemic logic): 采用模态词 L , 刻画认知。
- ▶ 非单调推理的核心: Assumption = Minimal Knowledge

Table of Contents

情形演算：Situation Calculus

Golog

非单调推理：Nonmonotonic Reasoning

回答集编程：Answer Set Programming

稳定模型语义

ASP 基础

ASP 编程

逻辑程序回顾

- ▶ 逻辑程序（正程序）为规则的集合

$$A \leftarrow A_1, A_2, \dots, A_m.$$

- ▶ 逻辑程序（正程序）有唯一的极小 Herbrand 模型

逻辑程序中引入 'not'

- ▶ 逻辑程序中规则定义为如下形式：

$$A \leftarrow A_1, \dots, A_m, \text{not } A_{m+1}, \dots, \text{not } A_n.$$

- ▶ 引入 'not'，从而引入非单调的表达能力。
- ▶ 'not' 表示“推不出”，逻辑程序下的推出（推不出）关系。
- ▶ 对于正程序有唯一的极小 Herbrand 模型，对于一般逻辑程序极小 Herbrand 模型不是唯一的。（哪些才是合理的？）

稳定模型语义 (Stable Model Semantics)

The stable model semantics for logic programming (Gelfond & Lifschitz 1988).

- ▶ Assumption = Minimal Knowledge.
- ▶ 对于含 'not' 的规则，我们不得不预先合理的假定一些知识 (assumption)。所谓的合理在这里解释为：与最后推导出的知识 (minimal knowledge) 一致。
- ▶ 对于任意原子的集合 S ， P^S 定义为 P 中按下列方式处理得到的程序：
 1. 如果规则体中有 $\text{not } a$ ，并且 $a \in S$ ，则删除这条规则；
 2. 删除剩余的含 not 文字。
- ▶ S 是 P 的稳定模型 iff S 是 P^S 的极小模型。

$$S = AS(P^S)$$

Stable model 性质

定理

S 是程序 P 的 *Herbrand* 模型当且仅当 S 是 P^S 的 *Herbrand* 模型。

定理

程序 P 的任何一个稳定模型都是 P 的一个极小 *Herbrand* 模型。

- ▶ 正程序有唯一的稳定模型，即该程序的极小模型；
- ▶ 有些程序没有稳定模型，例如 $\{p \leftarrow \text{not } p.\}$ ；
- ▶ 有些程序有多个稳定模型，例如 $\{p \leftarrow \text{not } q. q \leftarrow \text{not } p.\}$ ；
- ▶ 有些程序只有空集作为稳定模型，例如 $\{p \leftarrow p.\}$ ；
- ▶ 一个程序的稳定模型彼此互不为子集。

稳定模型语义举例

- ▶ 企鹅与鸟的例子：

$fly(X) \leftarrow bird(X), not\ nfly(X).$

/ 通常，鸟会飞 */*

$bird(X) \leftarrow penguin(X).$ */* 企鹅是鸟 */*

$nfly(X) \leftarrow penguin(X), not\ fly(X).$

/ 通常，企鹅不会飞 */*

$penguin(tweety).$ */* tweety 是企鹅 */*

- ▶ tweety 会不会飞？
- ▶ 此程序的所有稳定模型为：

$\{ bird(tweety), penguin(tweety), fly(tweety) \},$
 $\{ bird(tweety), penguin(tweety), nfly(tweety) \}.$

经典否定

- ▶ 前面的介绍中，没有涉及经典否定 $\neg$ 。在程序中其实可以加入经典否定：

$$L \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n.$$

其中， L 和 L_i 是文字，即原子或原子的否定 ($\neg$)。

- ▶ 对于含经典否定的正逻辑程序，仍然可以定义极小 Herbrand 模型，只不过此时 Herbrand 解释对应为文字的集合。
- ▶ 程序的 Herbrand 模型是一致的（不同时存在 p 和 $\neg p$ ）或者是文字的全体集合 Lit 。在此约束下，逻辑程序保持前面介绍的各种性质。
- ▶ 程序的 Answer set 为一个文字集 S ，满足 $S = AS(P^S)$ 。

逻辑程序与其常例

- ▶ 一个逻辑程序规则 R 在一个 Herbrand 解释下为真，当且仅当它的所有常例解释为真。
- ▶ 在有限常元的情况下，如果没有函数词，Herbrand 域是有限的，则规则的所有常例也是有限的。
- ▶ 逻辑程序和它的常例有相同的稳定模型。
- ▶ 在标准 ASP 程序里面，函数词禁止出现，我们只考虑常例程序，而含变量的程序作为其常例的缩写。ASP 本质是命题的。
- ▶ 经典非在 ASP 里面只是语法糖，可以用其他方式替代，以后也不再涉及。

ASP 程序

- ▶ 一个 ASP 程序是下列形式规则的集合：

$$H \leftarrow a_1, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n.$$

其中 a_i 为原子（常原子）， H 为空或者是一个原子。

- ▶ 当 H 为空时，我们称这个规则为“约束”。

$$\leftarrow a_1, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n.$$

- ▶ 约束的直观含义：约束体中的文字不应该同时满足。

Answer Sets

- ▶ 对于任意原子的集合 S , P^S 定义为 P 中按下列方式处理得到的程序:
 1. 如果规则体中有 $not\ a$, 并且 $a \in S$, 则删除这条规则;
 2. 删除剩余的含 not 文字。
- ▶ 令 P' 为 P 中不含约束的部分, S 是 P 的 Answer set, 当且仅当 S 是 P'^S 的极小模型并且 S 满足 P 中的所有约束。
- ▶ 约束可以下面规则替换 (f 为新原子):

$$f \leftarrow a_1, \dots, a_m, not\ a_{m+1}, \dots, not\ a_n, not\ f.$$

ASP 基本性质

- ▶ 逻辑程序可以有多个，一个，或者没有 answer set。
- ▶ 逻辑程序的 answer set 一定是头原子集合的子集。
- ▶ Minimality: 程序 P 的 answer set 也是 P 的极小模型。
- ▶ Supportedness: 原子 a 属于程序 P 的一个 answer set S , 则存在 P 的一个规则 r 使得 $body(r)$ 在 S 中为真且 $head(r) = a$ 。
- ▶ answer set $\neq$ minimal + supported.
- ▶ 原子 a 属于程序 P 的所有 answer set, 程序 P 的 answer set 都是程序 $P \cup \{a.\}$ 的 answer set, 但反过来不一定成立。例如程序 P_1 :

$\leftarrow not\ a.$

$a \leftarrow c.$

$c \leftarrow not\ b.$

$b \leftarrow not\ c.$

ASP 求解问题

- ▶ ASP 求解问题，只需要将问题本身描述出来。
- ▶ ASP 描述问题的方式：Generate-and-test，先构造问题可能的解，再用刻画约束解的条件。
- ▶ 对问题表达为通用的问题描述部分和问题实例，这样同一段问题描述的代码可以处理不同的问题实例。
- ▶ ASP 计算复杂性为 NP-complete.
- ▶ 主流的几个 ASP 求解器：
 - ▶ smodels: <http://www.tcs.hut.fi/Software/smodels/>
 - ▶ cmodels: <http://www.cs.utexas.edu/~tag/cmodels/>
 - ▶ clasp: <http://www.cs.uni-potsdam.de/clasp/>
 - ▶ dlv: <http://www.dbai.tuwien.ac.at/proj/dlv/>

n 着色问题

- ▶ 图的 n 着色问题：用 n 种颜色给图的各顶点着色，相邻顶点不能有相同颜色。
- ▶ ASP 编码：

$$\begin{aligned}color(V, C) &\leftarrow vertex(V), col(C), not othercolor(V, C). \\othercolor(V, C) &\leftarrow vertex(V), col(C), col(C1), C1 = C1, \\&\quad color(V, C1). \\&\leftarrow arc(V1, V2), col(C), color(V1, C), color(V2, C).\end{aligned}$$

n 皇后问题

- ▶ n 皇后问题：在 n 行 n 列的棋盘上，如果两个皇后位于棋盘上的同一行或者同一列或者同一对角线上，则称他们为互相攻击。现要求找出使 n 元棋盘上的 n 个皇后互不攻击的所有布局。
- ▶ ASP 编码：

$queen(R, C) \leftarrow not\ no_queen(R, C), num(R), num(C).$

$no_queen(R, C) \leftarrow not\ queen(R, C), num(R), num(C).$

$occupied(R) \leftarrow queen(R, C), num(R), num(C).$

$\leftarrow not\ occupied(R), num(R).$

$\leftarrow num(R), num(C), num(C1), C! = C1, queen(R, C), queen(R, C1).$

$\leftarrow num(R), num(R1), num(C), R! = R1, queen(R, C), queen(R1, C).$

$\leftarrow num(R), num(R1), num(C), num(C1), R! = R1, queen(R, C),$
 $queen(R1, C1), abs(R - R1) == abs(C - C1).$

哈密尔顿回路

- ▶ 哈密尔顿回路：经过图中所有顶点有且仅有一次的回路。
- ▶ ASP 编码：

$hc(V1, V2) \leftarrow arc(V1, V2), not otherroute(V1, V2).$

$otherroute(V1, V2) \leftarrow arc(V1, V2), arc(V1, V3), hc(V1, V3),$
 $V2! = V3.$

$otherroute(V1, V2) \leftarrow arc(V1, V2), arc(V3, V2), hc(V3, V2),$
 $V1! = V3.$

$reached(V2) \leftarrow arc(V1, V2), hc(V1, V2), reached(V1),$
 $not initialnode(V1).$

$reached(V2) \leftarrow arc(V1, V2), hc(V1, V2), initialnode(V1).$
 $initialnode(0).$

$\leftarrow vertex(V), not reached(V).$

小结

- ▶ 逻辑程序的基本动机：只需要描述问题，程序自动求解。
- ▶ ASP 的语义基于稳定模型语义。
- ▶ ASP 是一种非单调推理工具，其核心
assumption = minimal knowledge.
- ▶ ASP 编程的基本方式是：Generate-and-test.
- ▶ ASP 本身没有多项式算法可以求解。