南京大学
NANJING UNIVERSITY

人工智能导论

推理与规划
(Reasoning & Planning)

郭兰哲

南京大学 智能科学与技术学院

https://www.lamda.nju.edu.cn/guolz/IntroAI/fall2025/index.html

Email: guolz@nju.edu.cn

大纲

- 推理任务
- 基于逻辑的智能体
- 命题逻辑：表示与推理算法
- 一阶逻辑：表示与推理算法
  - **SMT Solver: Z3**
  - **Prolog**
- 知识图谱
- 自动规划

Recall

- 推理任务：演绎推理(数学推理)、归纳推理(ARC)、反绎推理(归因)、因果推理
- 基于逻辑的智能体：感知(Perception)、知识库(Knowledge Base)、推理(Reasoning)
- 知识表示：逻辑(Logic)
  - 命题逻辑
  - 两种推理方式：
    - 枚举推理：准确、但复杂度高
    - 归结推理：将命题逻辑语句转化为合取范式，用归结+反证法证明

Recall

- **推理算法**：如何基于已知前提推导出新结论
- **直接思路**：把所有可能的推理都试一下，遍历所有已知条件，看看能否根据某些推理规则进行推理，如果可以就进行推理，并把新结论放到知识库中，一直进行下去
- 关键问题：
  - 每次选择哪些条件？
  - 每次使用哪些推理规则？
  - 归结算法：每次挑选有互补文字的条件，使用归结原理

霍恩子句与确定子句

许多实际情形并不需要用到归结的全部能力，一些真实世界的知识库中的语句满足某些限制，这使得它们可以使用更为受限而更高效的推断算法

- **确定子句（definite clause)：文字的析取式，其中只有一个为正文字**

  例如：$\neg B_{11} \vee P_{12} \vee P_{21}$不是确定子句，$\neg B_{11} \vee \neg P_{12} \vee P_{21}$是确定子句

- **霍恩子句（horn clause)：文字的析取式，其中最多只有一个为正文字**

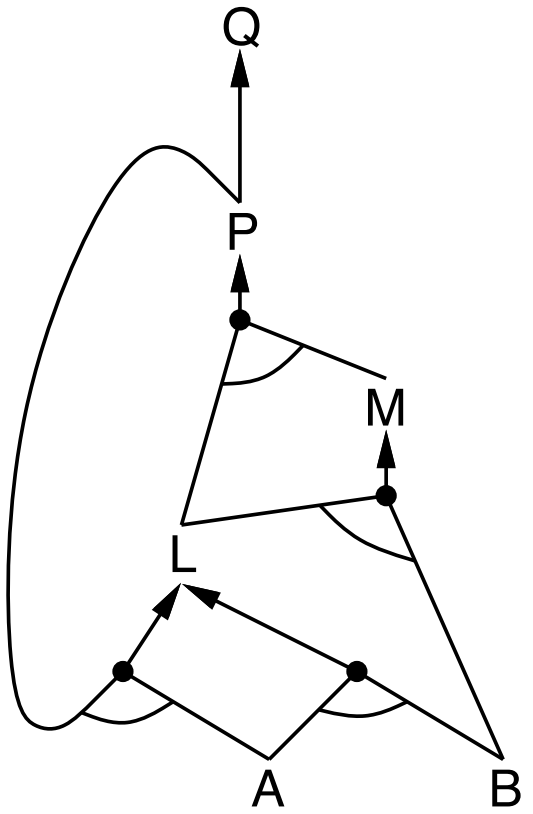
- **每个确定子句都可以写成一个蕴涵式**

$$\neg B_{11} \vee \neg P_{12} \vee P_{21} \qquad (B_{11} \wedge P_{12}) \rightarrow P_{21}$$

前向推理

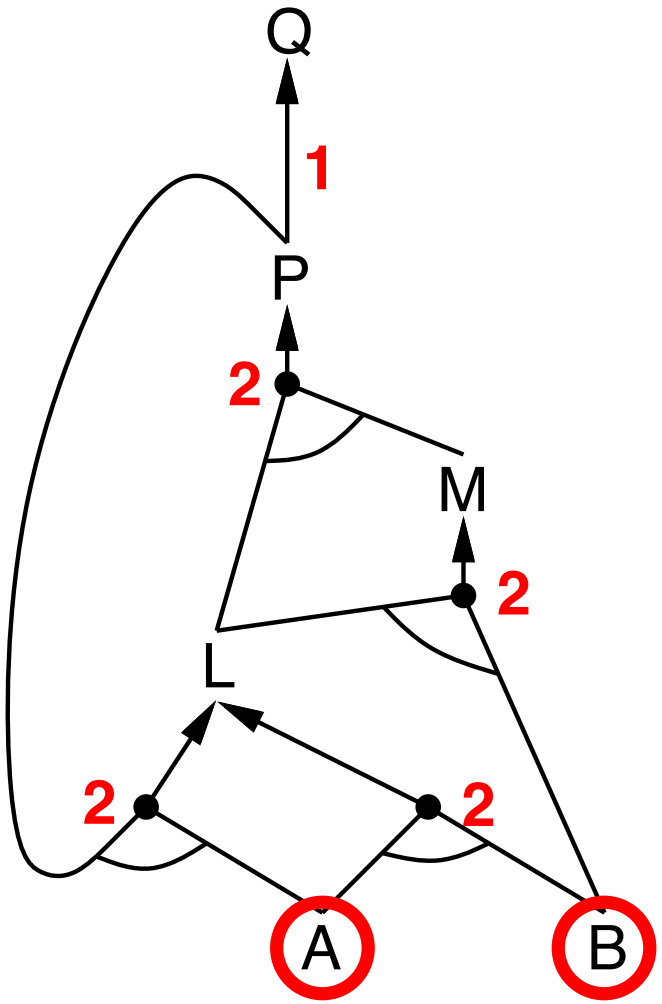
- 事实导向的推理(data driven)
- 找到前提满足知识库的规则，并把结论加入知识库
- 直到查询被添加或者无法进一步推断

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



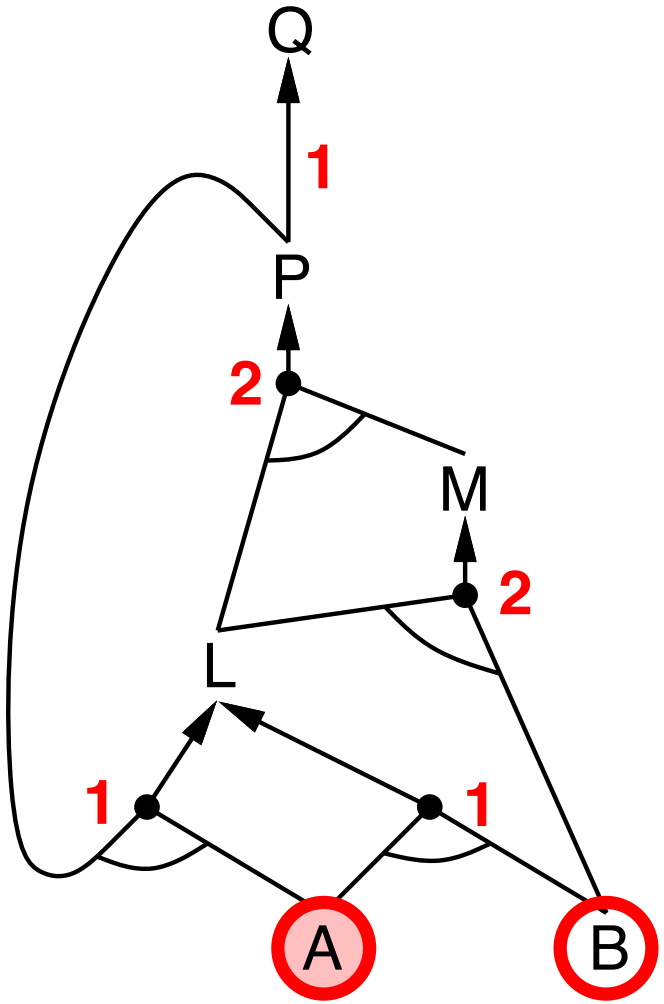
前向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



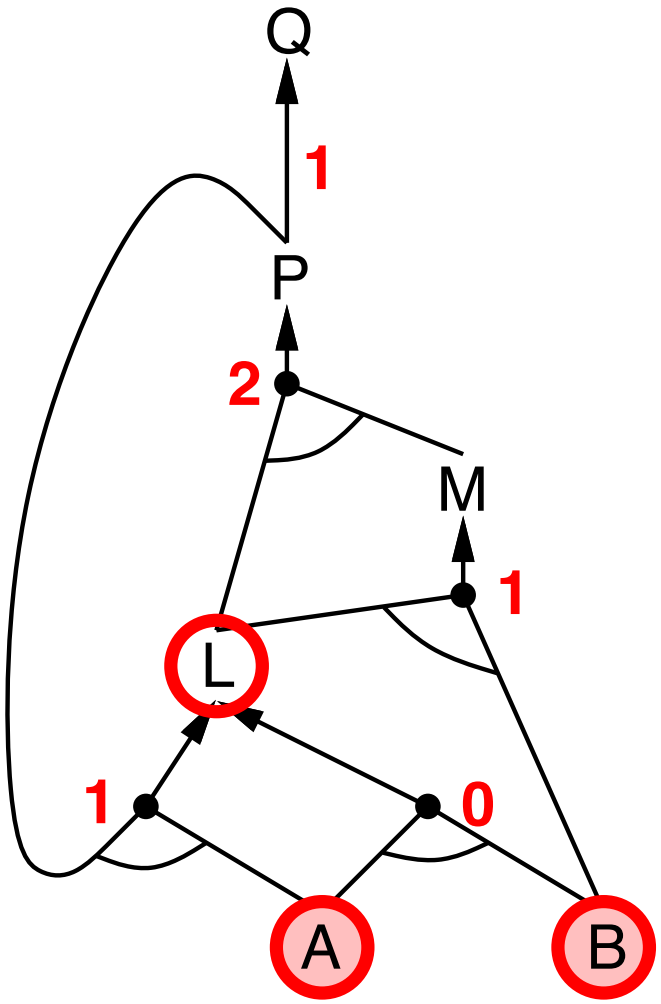
前向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



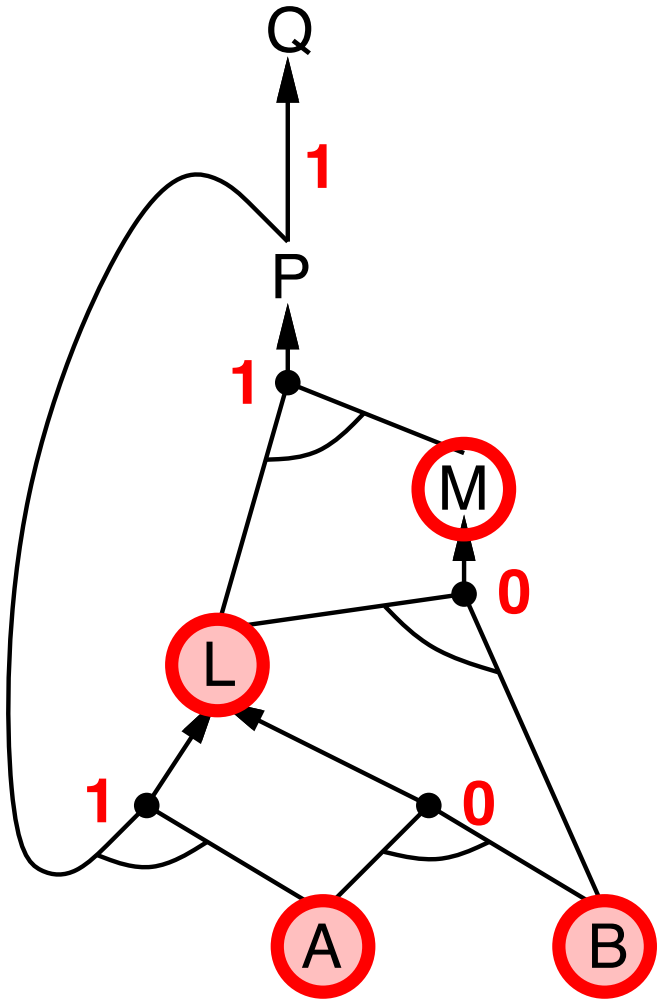
前向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



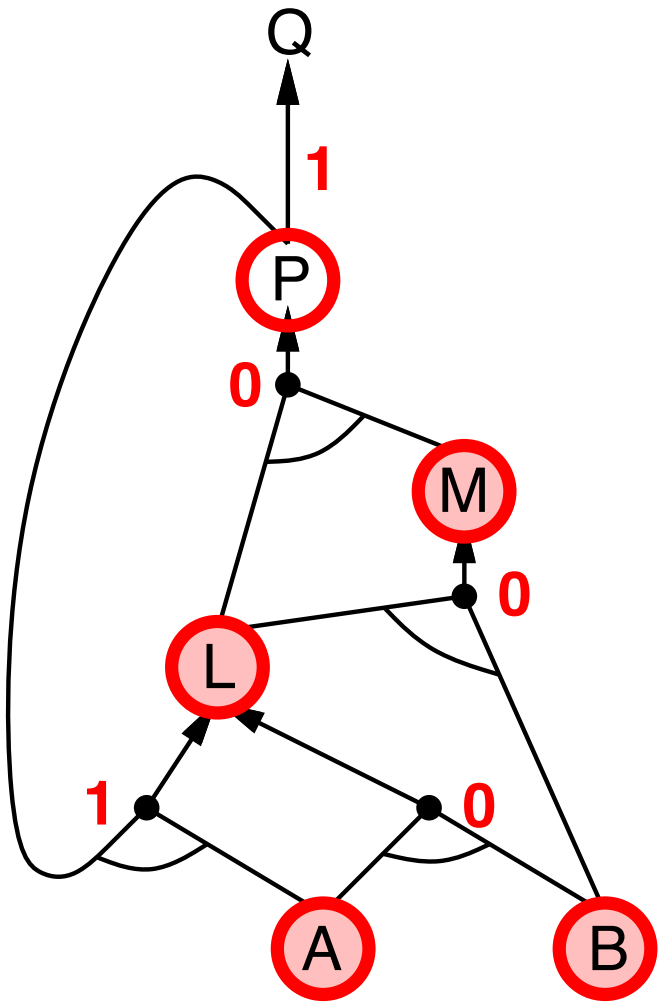
前向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



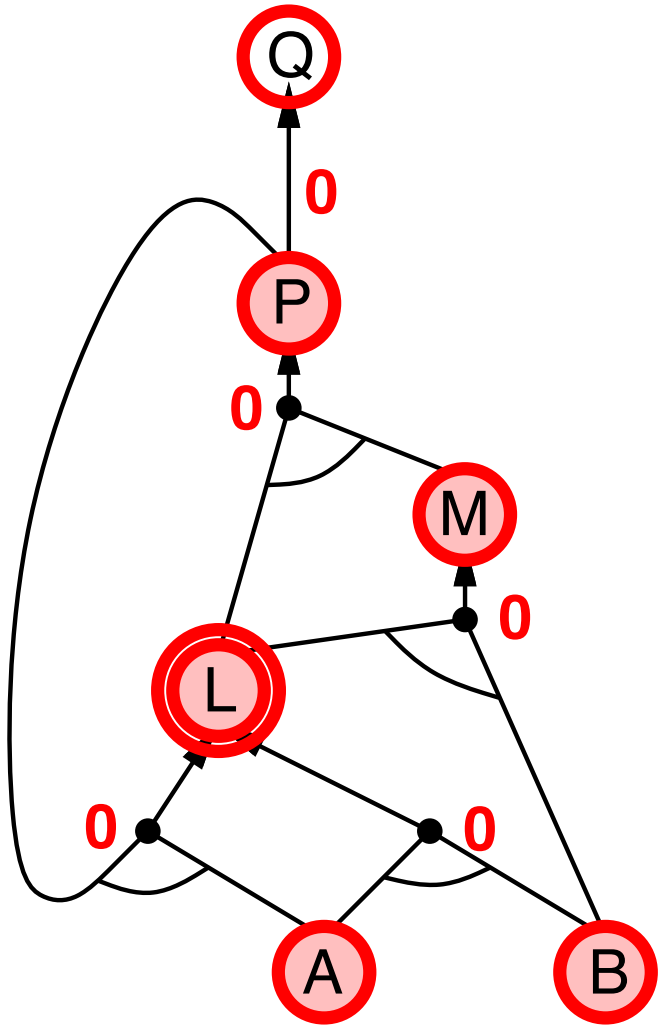
前向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



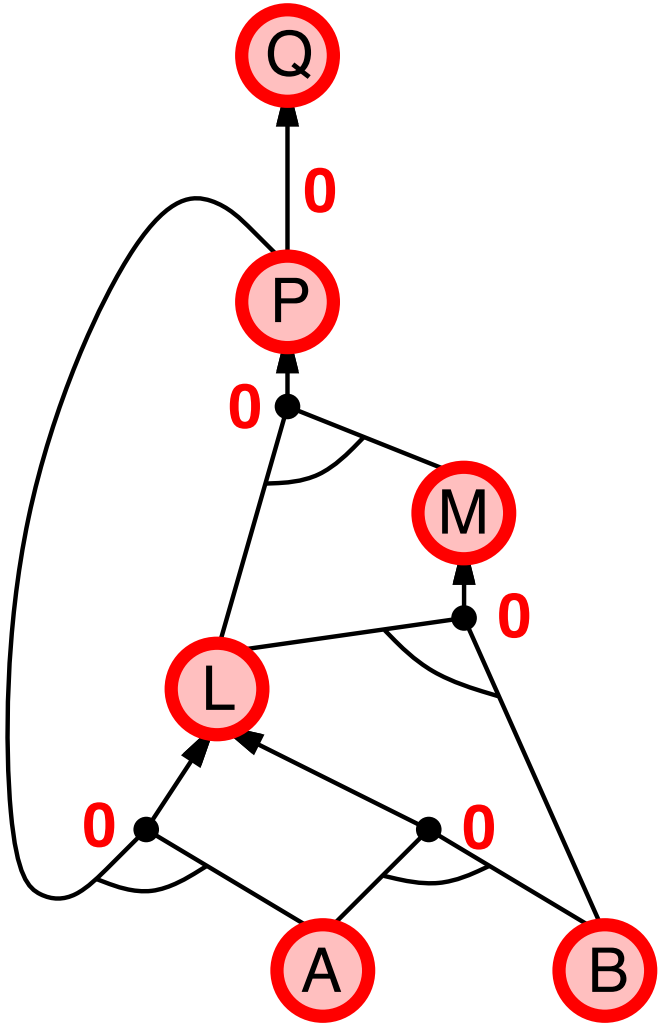
前向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



前向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

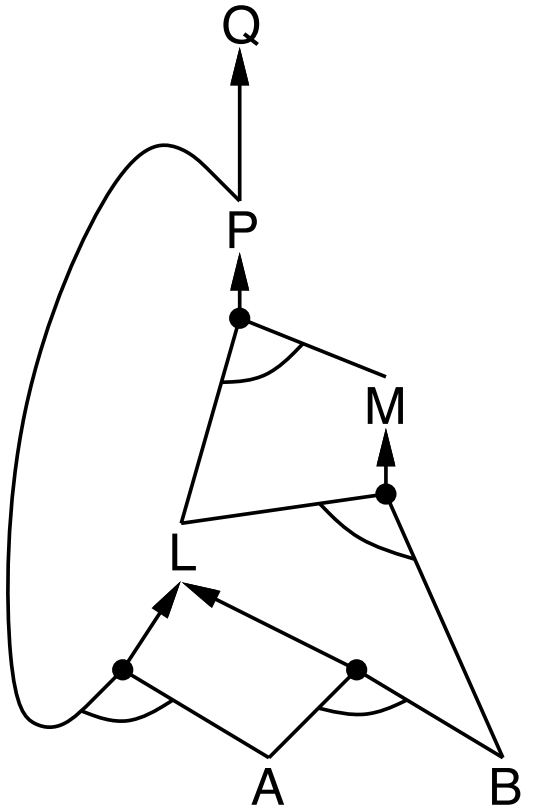


后向推理

目标导向的推理(Goal-Driven)

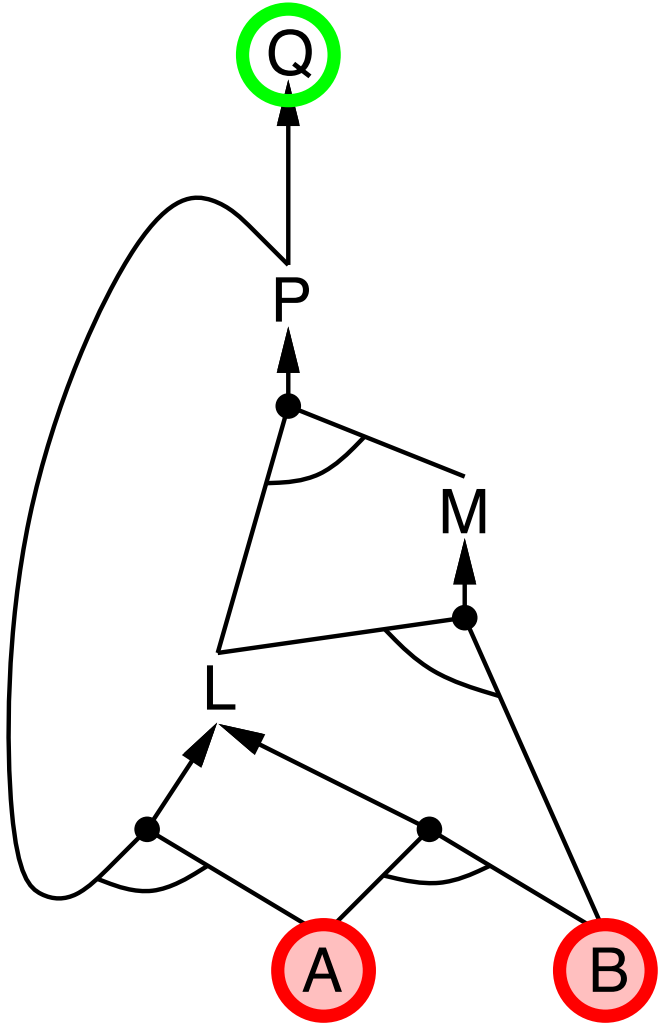
- 从查询q开始，反向运行，不断在知识库中查询结论为q的蕴涵式
- 如果这些蕴含式的所有前提都可以证明为真，则q为真

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



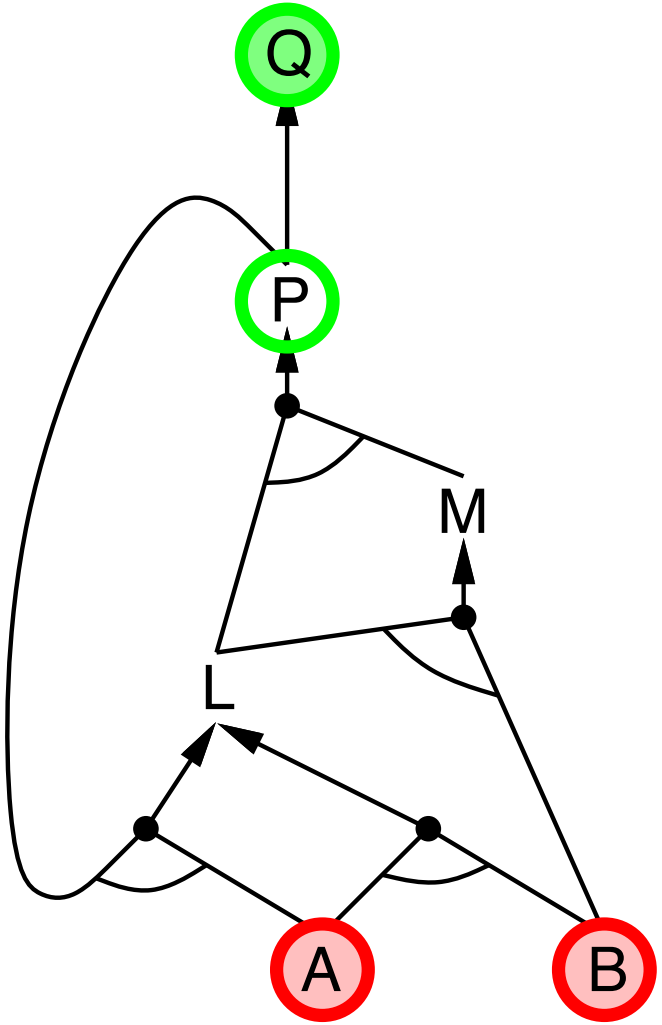
后向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



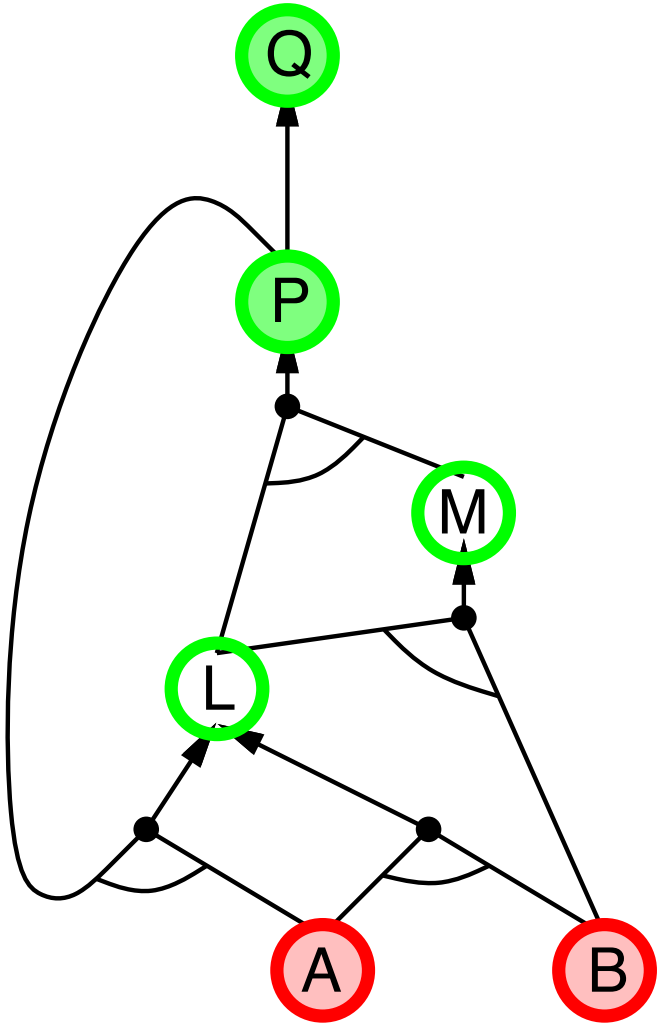
后向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



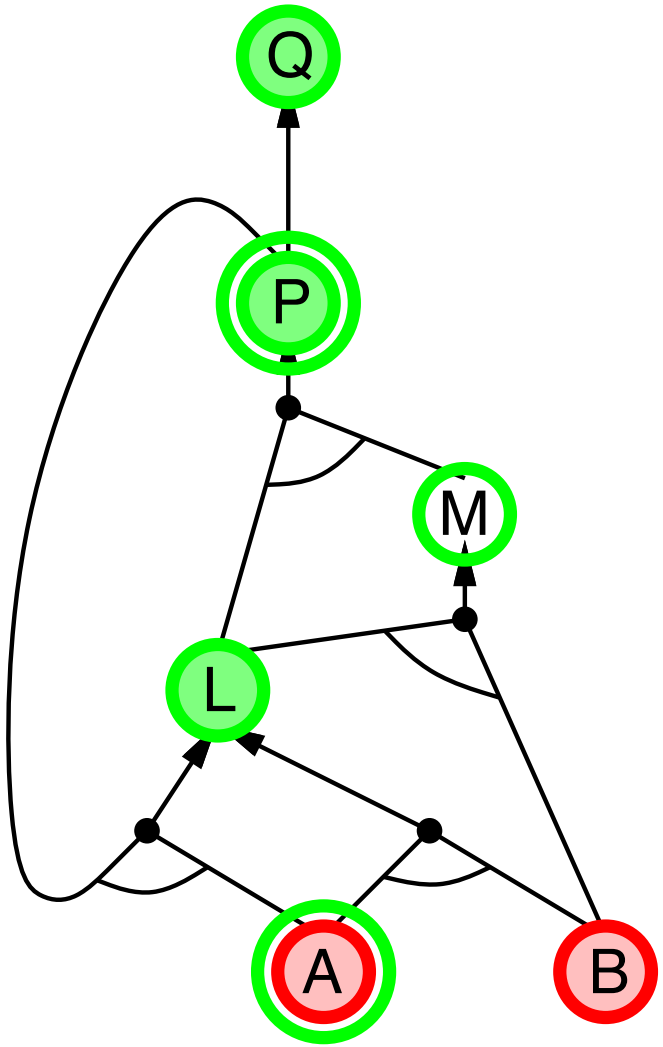
后向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



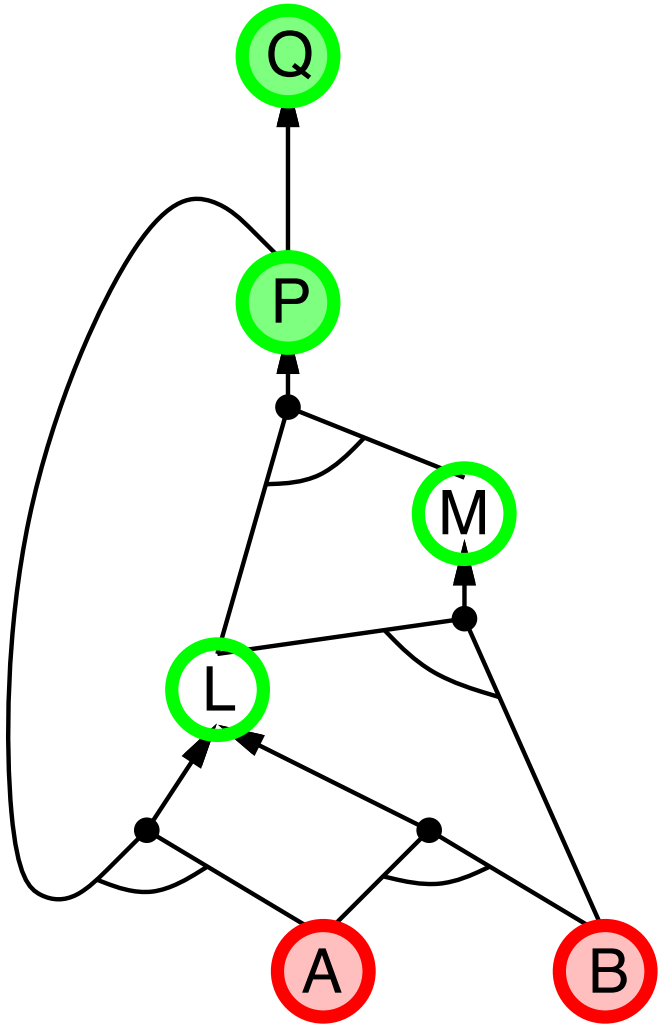
后向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



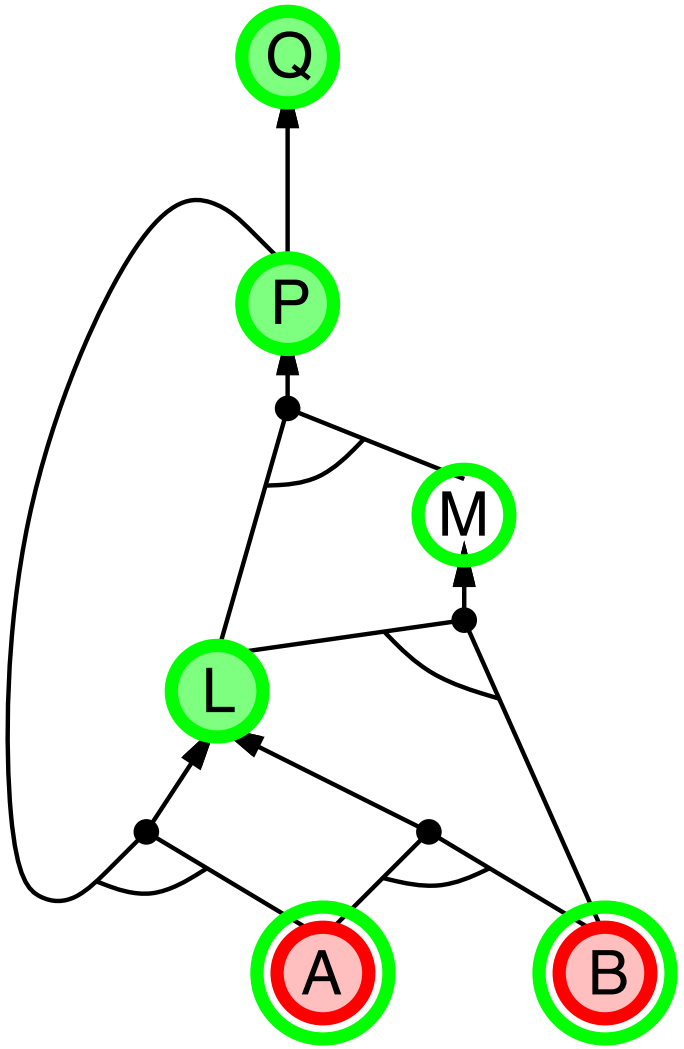
后向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



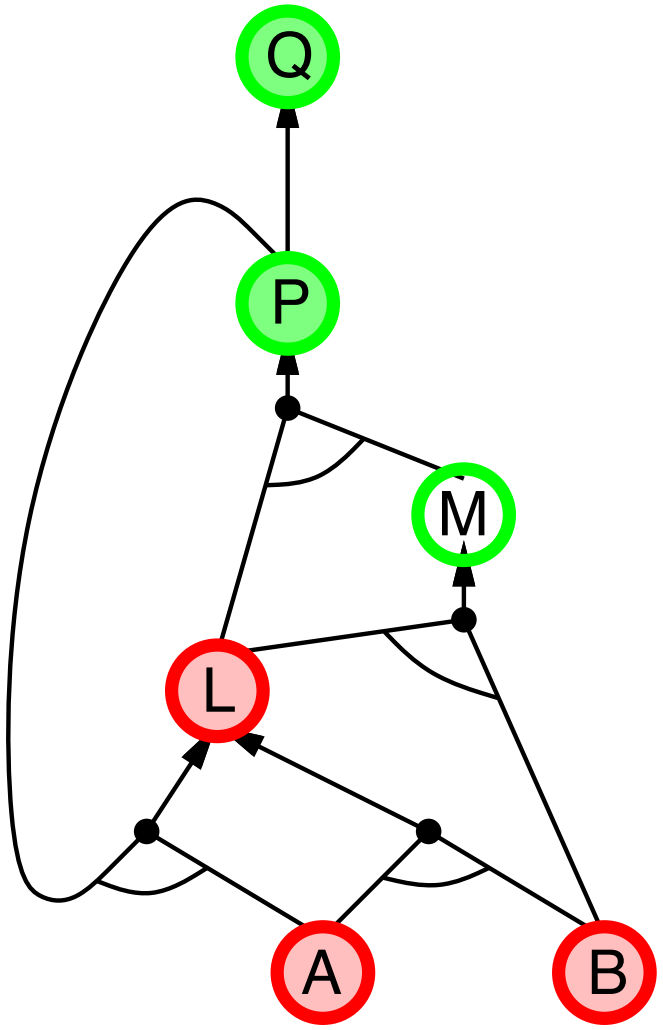
后向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



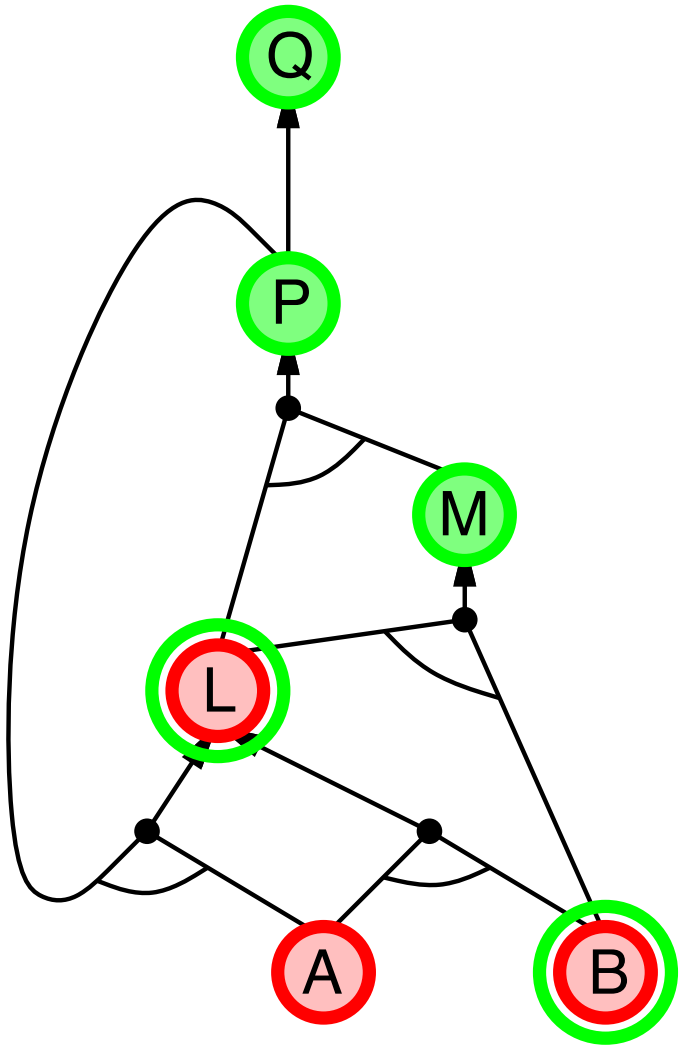
后向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



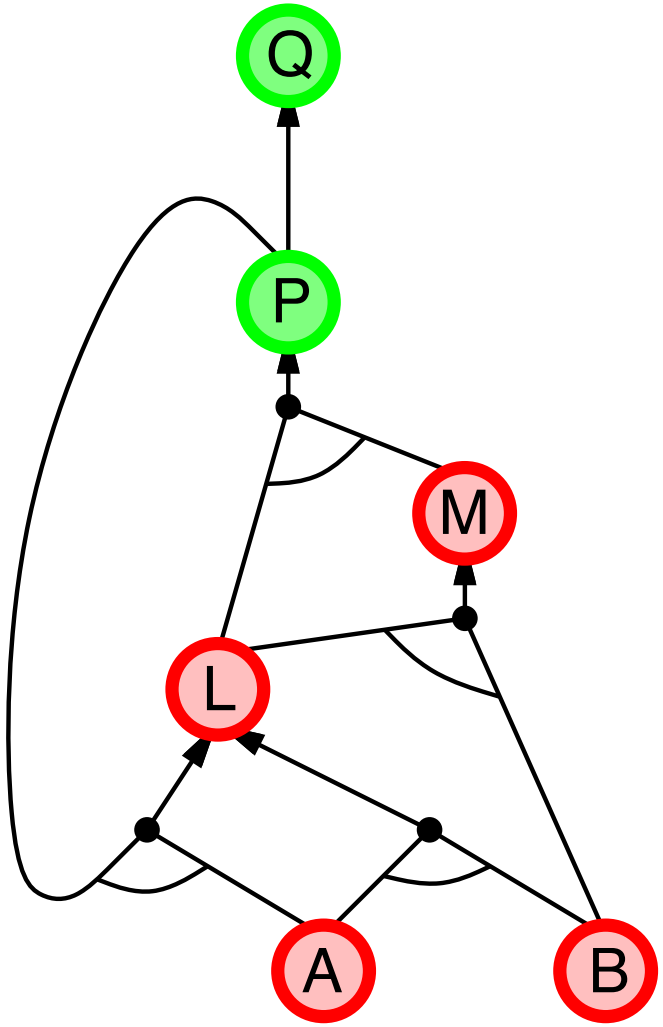
后向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



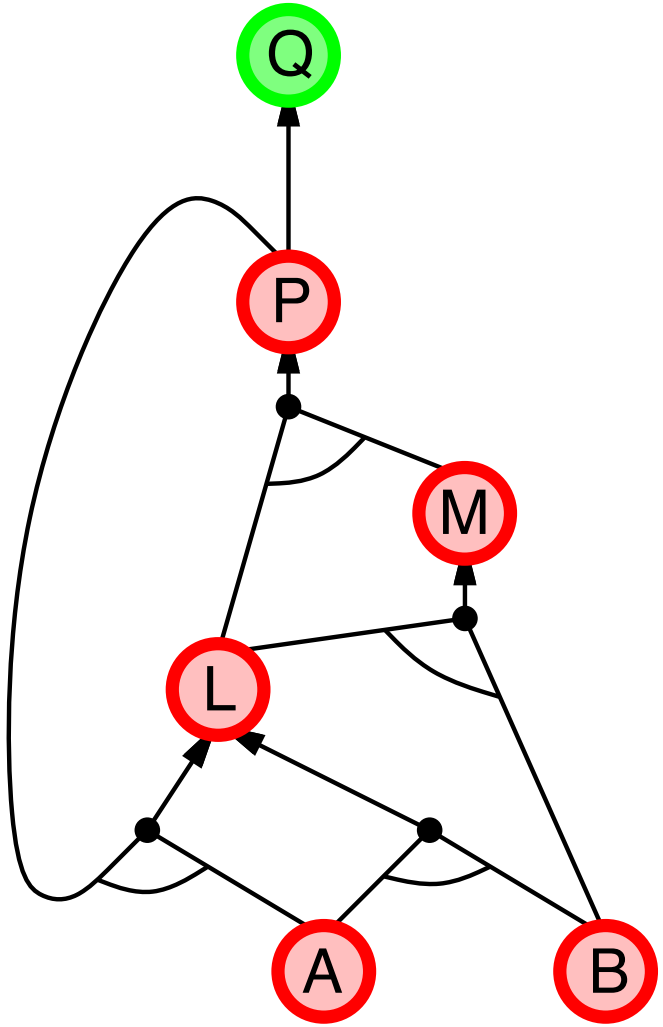
后向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



后向推理

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



命题逻辑的局限性

在命题逻辑中，每个陈述句是最基本的单位(即原子命题)，无法对原子命题进行分解，

因此在命题逻辑中，**不能表达局部与整体、一般与个别的关系**

- 例如，对于苏格拉底论断

✓ α：所有的人总是要死的

✓ β：苏格拉底是人

✓ γ：所以苏格拉底是要死的

无法在命题逻辑基础上推导出：$\alpha \wedge \beta \rightarrow \gamma$

虽然是正确的，但是无法通过命题逻辑来进行推理判断

一阶逻辑(First-Order Logic)

一阶逻辑/谓词逻辑

在谓词逻辑中，将原子命题进一步细化，分解出个体、谓词（predicate）和量词（quantifier），来表达个体与总体的内在联系和数量关系

- 个体：个体是指所研究领域中可以独立存在的具体或抽象的概念
- 具体或特定的个体：“张三”“李四”“1”“2”“3”
- 抽象或泛指的个体：x, y, z

谓词逻辑

- 谓词：谓词是用来**刻画个体属性**或者**描述个体之间关系存在性**的元素，其值为真或为假

  - 包含一个参数的谓词称为一元谓词，表示一元关系，通常是用来刻画个体是否包含特定的属性：如$p(x)$: x是质数，表示某个数字是否是质数
  - 包含多个参数的谓词称为多元谓词，表示个体间的多元关系，通常用于描述个体之间是否存在特定的关联，如Father(x, y)表示x是y的父亲

思考：谓词和函数有什么区别？

谓词逻辑：量词

◘ 全称量词 (universal quantifier, ∀)

全称量词用符号∀表示，表示一切的、凡是的、所有的、每一个等。

$\forall x$表示定义域中的所有个体，$\forall x P(x)$表示定义域中的所有个体具有性质P

◘ 存在量词(existential quantifier, ∃)

存在量词用符号∃表示，表示存在、有一个、某些等。

$\exists x$表示定义域中存在一个或若干个个体，

$\exists x P(x)$表示定义域中存在一个个体或若干个体具有性质P

◘ 全称量词和存在量词统称为量词

谓词逻辑

全称量词与存在量词之间的关系

- $\forall x P(x) \equiv \neg \exists x \neg P(x)$
- $\forall x \neg P(x) \equiv \neg \exists x P(x)$
- $\neg \forall x P(x) \equiv \exists x \neg P(x)$
- $\exists x P(x) \equiv \neg \forall x \neg P(x)$

谓词逻辑

- 约束变元：在全称量词或存在量词约束条件下的变量称为约束变元
- 自由变元：不受全称量词或存在量词约束的变量称为自由变元

自由变元即可以存在于量词的约束范围之内，也可以存在于量词的约束范围之外

$$(\forall x)(A(x) \vee B) \equiv (\forall x)A(x) \vee B$$

$$(\forall x)(A(x) \wedge B) \equiv (\forall x)A(x) \wedge B$$

$$(\exists x)(A(x) \vee B) \equiv (\exists x)A(x) \vee B$$

$$(\exists x)(A(x) \wedge B) \equiv (\exists x)A(x) \wedge B$$

谓词逻辑

在约束变元相同的情况下，**全称量词对合取满足分配律**，**存在量词对析取满足分配律**

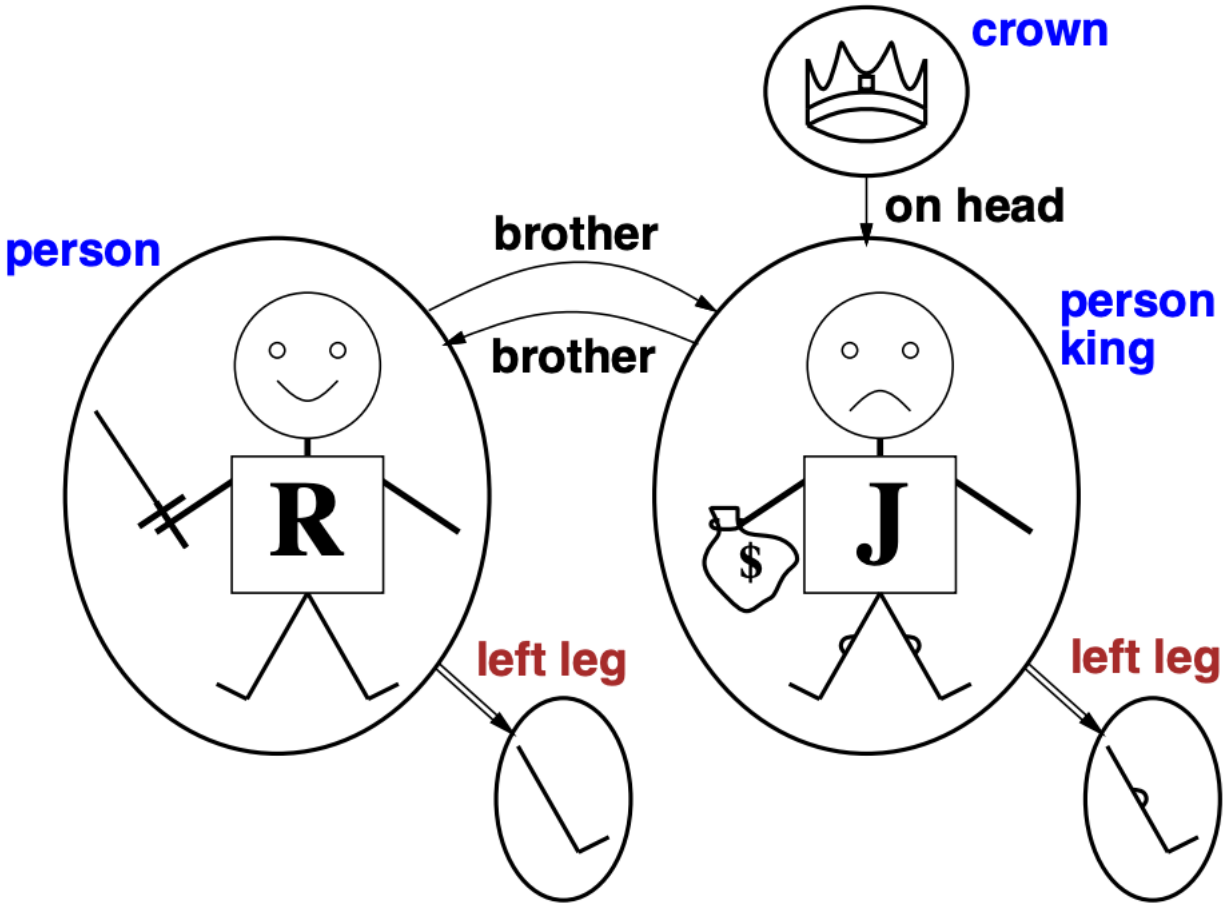
$$(\forall x)(A(x) \wedge B(x)) \equiv (\forall x)A(x) \wedge (\forall x)B(x)$$

$$(\exists x)(A(x) \vee B(x)) \equiv (\exists x)A(x) \vee (\exists x)B(x)$$

$$(\forall x)(A(x) \vee B(x)) \equiv (\forall x)A(x) \vee (\forall x)B(x) \quad \text{（不成立）}$$

$$(\exists x)(A(x) \wedge B(x)) \equiv (\exists x)A(x) \wedge (\exists x)B(x) \quad \text{（不成立）}$$

谓词逻辑：事实符号化



$$King(x)$$

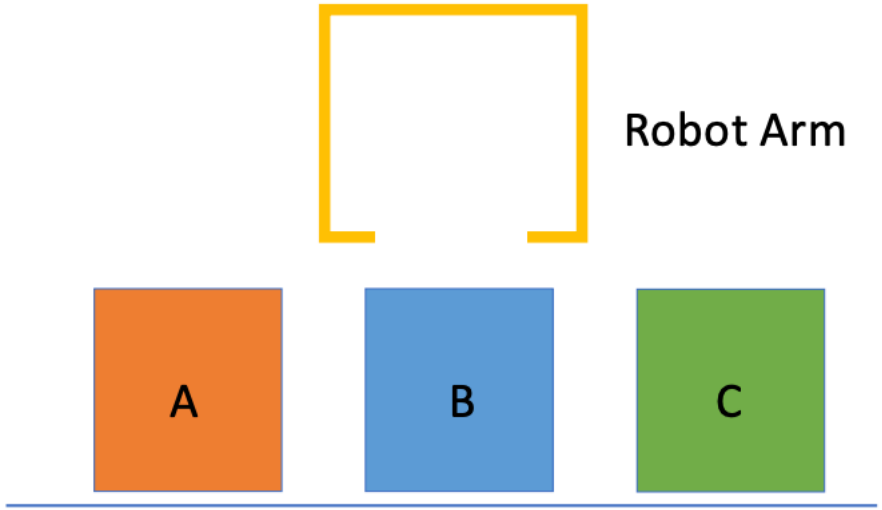
$$Person(x)$$

$$Head\_on(Crown, x)$$

$$\forall x(King(x) \rightarrow Person(x)$$

$$\forall x(King(x) \rightarrow Head\_on(Crown, x)$$

谓词逻辑：事实符号化



$On\_table(A)$

$On\_table(B)$

$On\_table(C)$

Hand-Empty()

On(A, B)

……

谓词逻辑：推理规则

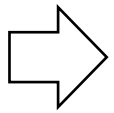
问题1：如何解决量词？

设$A(x)$是谓词公式，x和y是变元，a是常量符号，则存在如下推理规则：

- 全称量词消去(Universal Instantiation, UI): $(\forall x)A(x) \rightarrow A(y)$
- 存在量词消去(Existential Instantiation, EI): $(\exists x)A(x) \rightarrow A(c)$
  - c必须是一个新的、从未使用过的符号

退化到命题逻辑推理

- $(\forall x)King(x) \wedge Greedy\ (x) \rightarrow Evil(x)$
- $King(John)$
- $Greedy(John)$
- $Brother(Richard, John)$



- $King(John) \wedge Greedy\ (John) \rightarrow Evil(John)$
- $King(Richard) \wedge Greedy(Richard) \rightarrow$

  $Evil(Richard)$

问题：效率不高，生成$King(richard) \wedge Greedy(richard) \rightarrow Evil(richard)$

这样的语句并没有意义

合一(Unification)

命题逻辑中的归结

- 子句 1: $Rich \vee Happy$
- 子句 2: $\neg Rich$
- 归结直接消去 $Rich$ 和 $\neg Rich$
- 结果：$Happy$

一阶逻辑的问题：

- 子句 1: $Rich(x) \vee Happy(x)$
- 子句 2: $\neg Rich(Bill)$

合一(Unification)
通过变量替换让两个不同的逻辑
表达式变得相同

$$Unify(Rich(x), Rich(Bill)) = \{x/Bill\}$$

- Unify(Knows(John, x), Knows(John, Jane))={x/Jane}
- Unify(Knows(John, x), Knows(y, Bill))={x/Bill, y/John}
- Unify(Knows(John, x), Knows(y, Mother(y)))={y/John, x/Mother(John)}
- Unify(Knows(John, x), Knows(x, Eliza))=failure

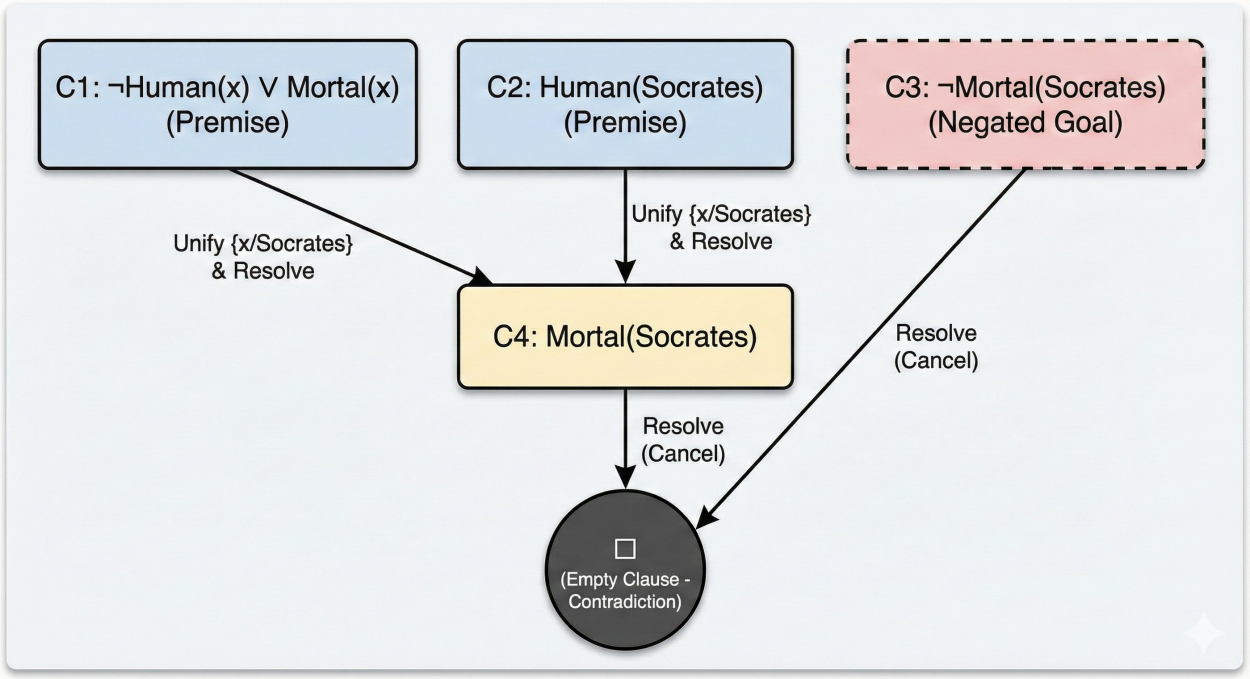
适用于一阶逻辑的归结算法

- $\forall x\big(Human(x) \rightarrow Mortal(x)\big)$
- $Human(Socrates)$
- $Mortal(Socrates)$

转换为CNF

- $\neg Human(x) \vee Mortal(x)$
- $Human(Socrates)$
- $\neg Mortal(Socrates)$

证明前提1+前提2+$\neg Mortal(Socrates)$矛盾



其他逻辑形式

- 命题逻辑(Propositional Logic)：无量词，只能表示真假命题
- 一阶逻辑(First-Order Logic)：引入量词、谓词
- 二阶逻辑(Second-Order Logic)：量词可以作用于谓词
- 时序逻辑(Temporal Logic)：引入时间算子
- 模糊逻辑(Fuzzy Logic)：真值$\in [0,1]$，而非Ture/False布尔值
- ……

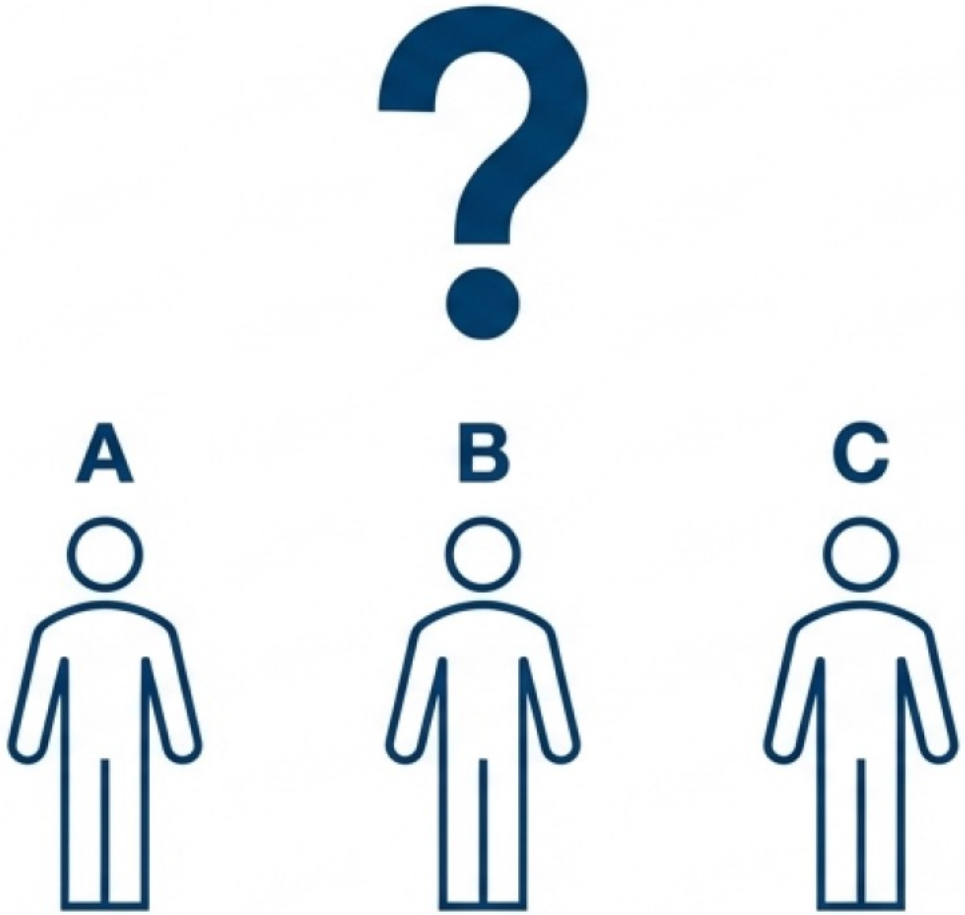
大纲

- 推理任务
- 基于逻辑的智能体
- 命题逻辑：表示与推理算法
- 一阶逻辑：表示与推理算法
  - **SMT Solver：Z3**
  - **逻辑编程语言：Prolog**
- 知识图谱
- 自动规划

Z3求解器：从逻辑谜题到AI验证的艺术

将“思考”转化为“计算”的终极工具

谁是窃贼？



场景：

警察抓获了三名嫌疑人A，B，C，已知其中有且只有一人是窃贼

口供记录：

- A说：“我不是窃贼”
- B说：“C是窃贼”
- C说：“窃贼在A和B之间”

关键线索：

在这三个人中，只有窃贼说了谎话

核心问题： 窃贼究竟是谁？我们如何将这个问题“翻译”成机器能懂的语言

布尔可满足性(SAT)

核心定义：

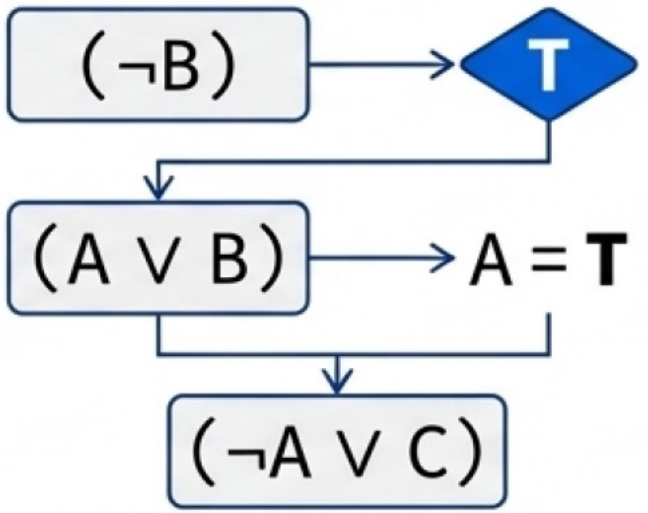
给定一个由变量与逻辑运算符(与、或、非)组成的布尔公式，是否存在一组真/假复制，能让整个公式结果为真?

地位：

计算机科学中第一个被证明是**NP-Complete**的问题

示例剖析：

$$(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B)$$



要使公式为真，$\neg B$必须为真，即**B=False**

代入$A \vee B$，则**A**必须为真，即**A=True**

入$\neg A \vee C$，则**C**必须为真，即**C=True**

结论： 存在解(A=T, B=F, C=T)，该问题是可满足的(SAT)。

模理论可满足性(SMT)

核心思想：

SMT=SAT求解引擎+理论约束(如算术、数组、字符串等)

解释：

SAT只能处理True/False，SMT将SAT的逻辑搜索能力扩展至数学领域，如整数、实数、数组等

理论约束示例

- 数值约束

$$0 < x < 5$$

- 算术约束

$$x + y < 10$$

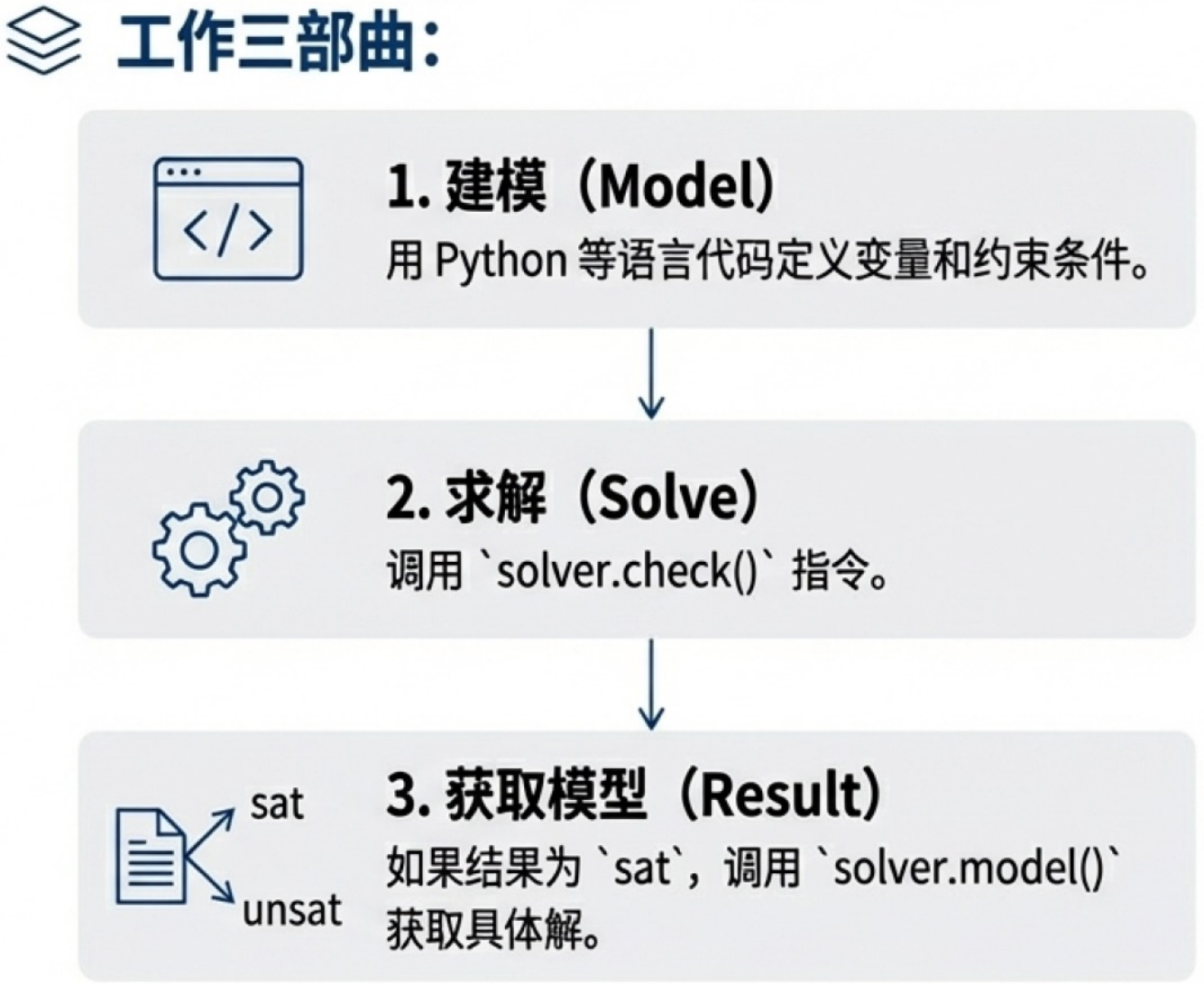
- 数组

$$a[i+1] = 2 * a[i] + 1$$

- ……

SMT求解器： Z3 Solver

- 微软研究院开发
- 工业界和学术界最常用的SMT求解器
- 应用：软件验证、硬件设计、AI推理验证
- 入门资料：https://ericpony.github.io/z3py-tutorial/guide-examples.htm



Z3实战(一)：破解窃贼之谜

逻辑建模

变量：

A，B，C代表“A/B/C”是窃贼

约束1：

只有一个窃贼

约束2：

一个人是窃贼，当且仅当他说的是谎话

A的口供：$\neg A$

B的口供：C

C的口供：$A \vee B$

Python+Z3代码

```python
# 1. 定义布尔变量: A_is_thief 表示 A 是窃贼，以此类推
A, B, C = Bools('A B C')

solver = Solver()

# 2. 约束: 只有 1 个人是窃贼
solver.add(
    Or(And(A, Not(B), Not(C)),And(Not(A), B, Not(C)), And(Not(A), Not(B), C)
    ))

# 3. 建模供词 (True/False)
# A 说 "我不是窃贼" -> Not(A)
# B 说 "C 是窃贼"   -> C
# C 说 "A或B是窃贼" -> Or(A, B)
statement_A = Not(A)
statement_B = C
statement_C = Or(A, B)

# 4. 核心逻辑:
# "只有窃贼说了谎话" 意味着:
# (A是窃贼 <-> A的话为假) AND (B是窃贼 <-> B的话为假) ...

solver.add(A == Not(statement_A))
solver.add(B == Not(statement_B))
solver.add(C == Not(statement_C))

# 5. 求解
if solver.check() == sat:
    print(solver.model())
else:
    print("矛盾供词，无解。")
```

输出：[A = False, B = False, C = True]，结论：C是窃贼

Z3实战(二)：求解高阶版鸡兔同笼

问题描述

寻找一组正整数解，满足以下所有条件：

- $x > 0, y > 0, z > 0$
- $3x + 2y - z = 13$
- $x * y + z = 20$

Python+Z3代码

```python
# 1. 定义整数变量
x, y, z = Ints('x y z')

solver = Solver()

# 2. 添加约束
solver.add(x > 0, y > 0, z > 0)          # 范围约束
solver.add(3*x + 2*y - z == 13)          # 线性约束
solver.add(x * y + z == 20)              # 非线性约束

# 3. 求解
if solver.check() == sat:
    m = solver.model()
    print(m)
else:
    print("该方程组无整数解。")
```

输出：[x = 1, y = 10, z = 10]

要点：Z3能够轻松处理包含非线性算术的复杂约束问题

Z3实战(三)：自动定理证明

核心原理：反证法

- 目标：对于所有a, b，证明

  $(a+b)^2 = a^2 + 2ab + b^2$

- 向Z3提问：是否存在一组a, b，使得结论不成立

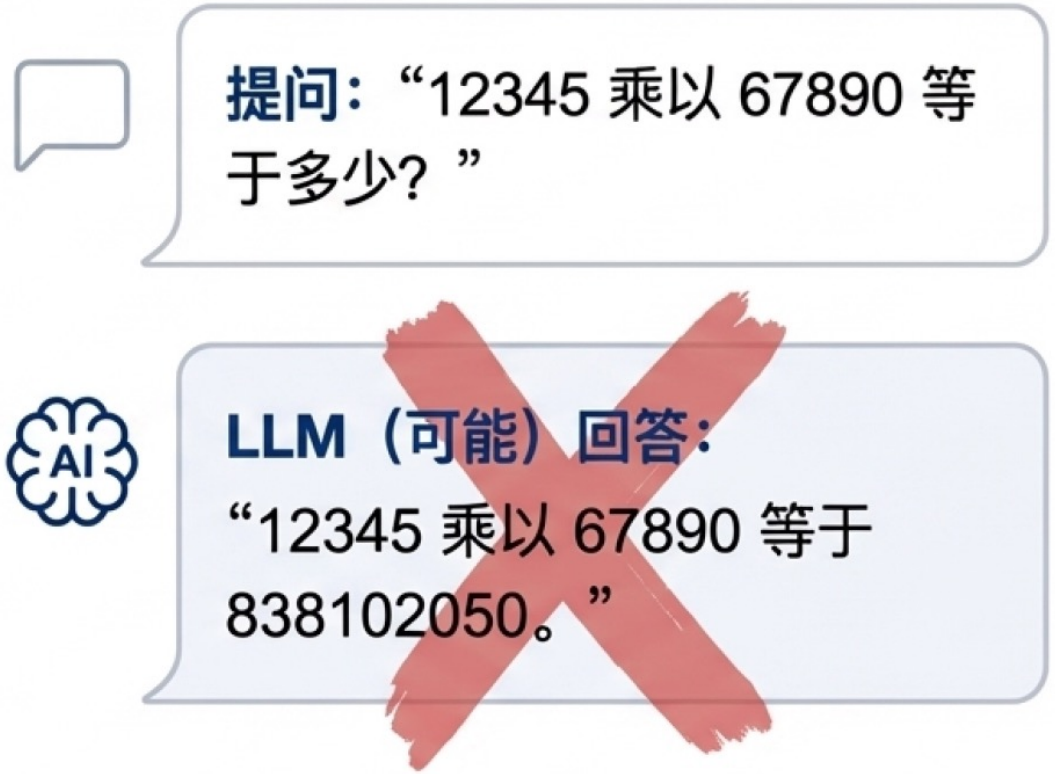
- 判定：如果Z3判定unsat(无解)，则原命题成立，否则找到了反例，原命题不成立

Python+Z3代码

```python
# 1. 定义实数变量
a, b = Reals('a b')

solver = Solver()

# 2. 定义我们要证明的定理
theorem = ((a + b)**2 == a**2 + 2*a*b + b**2)

# 3. 反证法核心:
# 我们告诉 Z3: "请帮我找一个反例，使得定理不成立"
# 即: 添加 Not(theorem)
solver.add(Not(theorem))

# 4. 检查是否有反例
result = solver.check()

if result == unsat:
    print("结论: 找不到反例 -> 定理对于所有实数 a, b 成立! ")
elif result == sat:
    print("结论: 定理是错的。反例为:", solver.model())
```

输出：找不到反例，结论成立

新时代的挑战：LLM的计算幻觉



LLM在需要绝对精确的计算与严谨逻辑时，常常会犯错或前后矛盾

应用场景(一)：LLM调用工具求解

解决方案(Program of Thought)：LLM不再直接计算，而是与Z3协作

翻译官

将用户的自然语言提问翻译成精确的Z3代码

运行代码，得出100%可靠可验证的结果

解说员

将Z3返回的精确结果组织称流程的自然语言，回答用户

应用场景(二)：AI生成内容的形式化验证

常见场景

- AI生成的代码是否存在安全漏洞(如数组越界、除零错误)?
- AI生成的数学推理过程是否严谨无误?

验证流程

- LLM 生成代码或数学证明
- LLM自动提取代码的逻辑/路径条件
- Z3 检查是否都满足规范
- 若不满足，说明该代码可能存在漏洞 → 反馈给 LLM 自修正(self-refine)

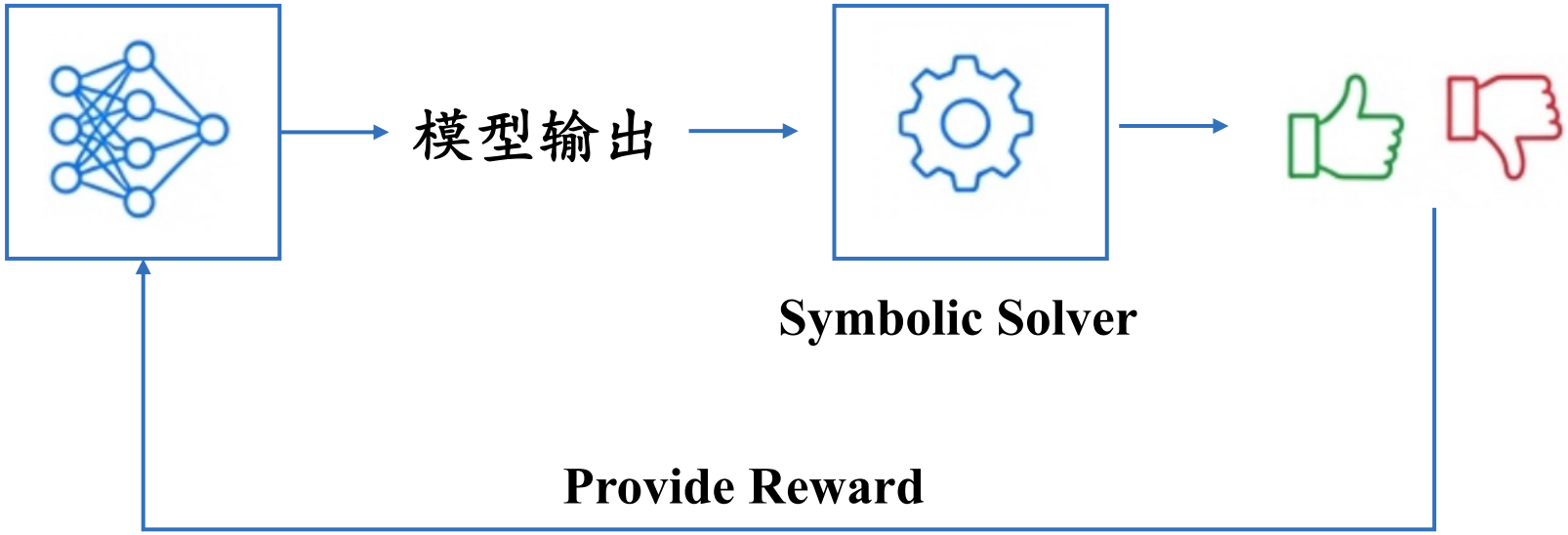
AI 生成的代码 / 证明

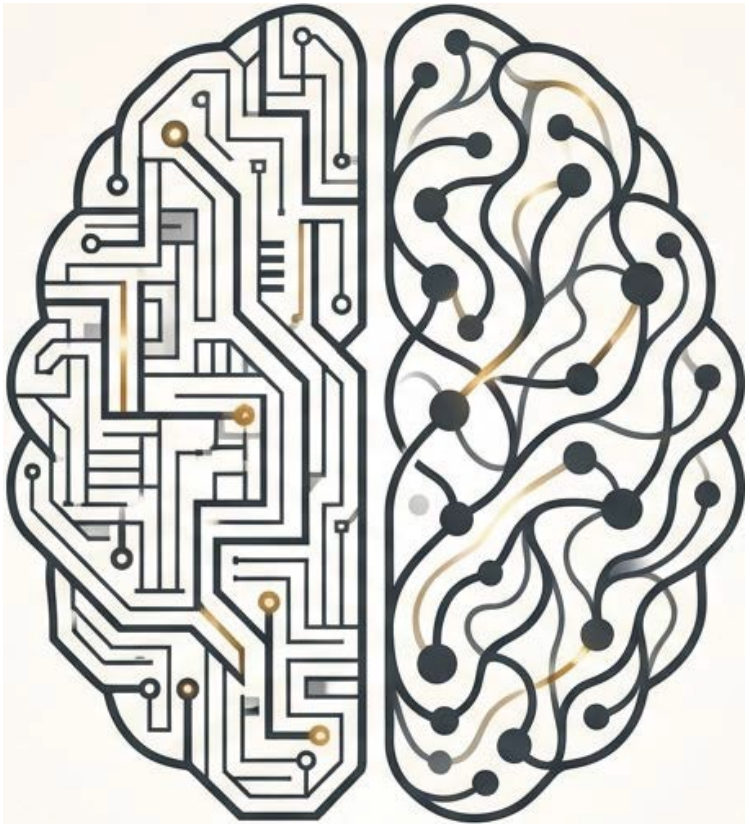
```
def process_data(data):
  if data:
    result = 100 / data[0]
    # … more logic
    return result

# 规范 (Specification):
# pre: data is not null
# pre: len(data) > 0
# pre: data[0] != 0
# post: result is valid
```

应用场景(三)：RLVR

对于数学推理、逻辑推理等问题，引入符号求解器，验证大模型输出，

从而提供奖励信号，用于强化微调





Thinking in Logic: An Introduction to Prolog

--逻辑编程的艺术

命令式 vs 声明式

命令式编程 (Imperative) - Python, C++

告诉计算机执行的步骤(算法)

Tell the computer ***how to do*** something

Program = **Data Structure** + **Algorithm**

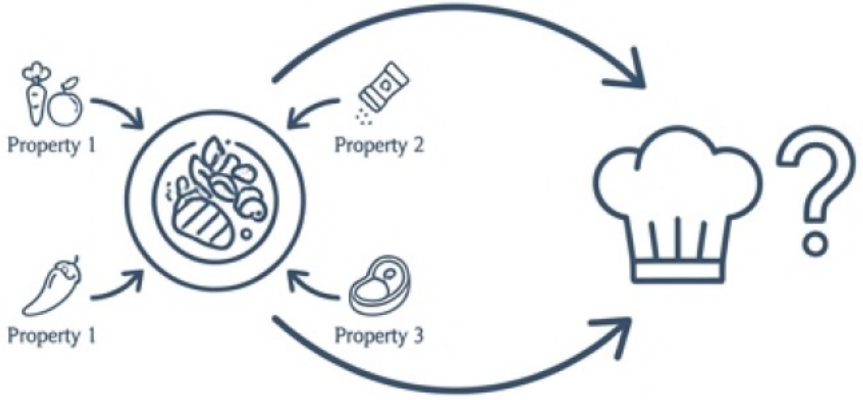
要制作蛋糕，先打破鸡蛋，然后加面粉，然后搅拌...

声明式编程 (Imperative) - Prolog

告诉世界是什么(知识库)

Tell the computer ***what is true***

Program = **Logic** + **Control**



蛋糕是美味的，如果一个东西包含面粉、鸡蛋和糖，并且经过烘烤，那么它就是蛋糕，计算机自己去寻找符合条件的东西

构建知识库：事实(Facts)与规则(Rules)

事实(Facts)

语法： predicate(atom1, atom2).

- human(socrates).
- parent(john, mary).
- likes(alice, pizza).

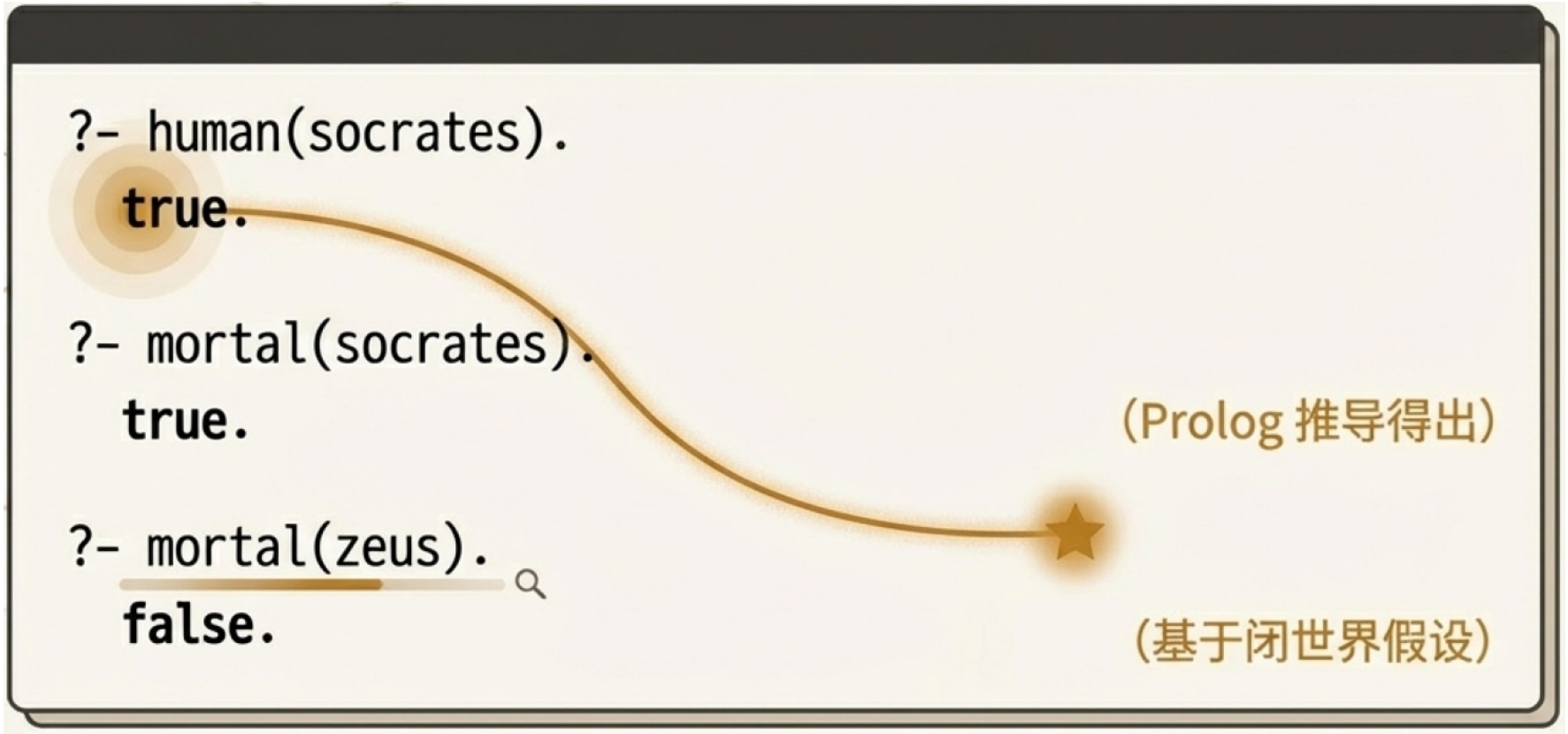
规则(Rules)

语法： Head :- Body.
(Head为真，如果Body为真)

- mortal(X) :- human(X).
- grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

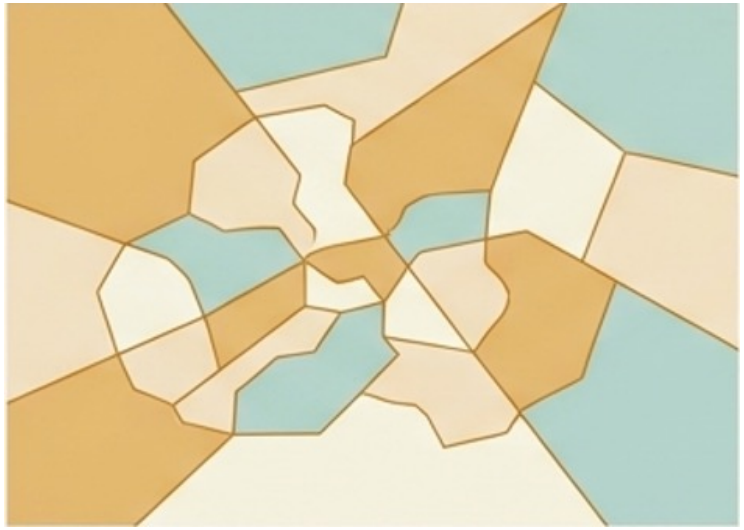
与知识库对话：查询(Queries)

向系统提问，Prolog会基于已知的事实和规则进行推理



应用场景(一)：求解经典逻辑难题

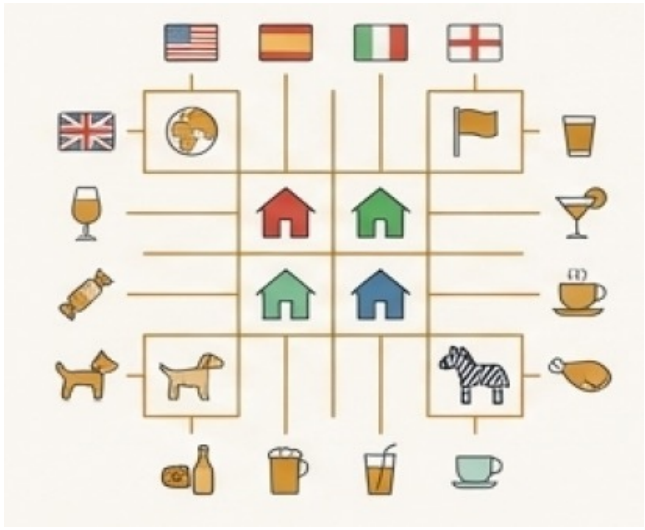
许多复杂问题，其本质就是一组逻辑约束



地图着色问题

“相邻区域的颜色不能相同”

约束满足



爱因斯坦斑马谜题

“英国人住在红房子里，西班牙人养狗…”

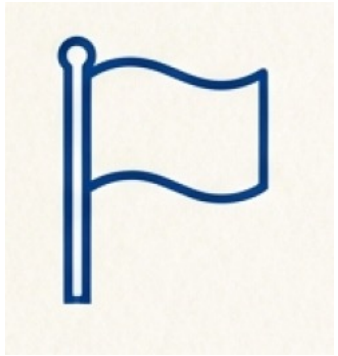
逻辑推理

五栋房子之谜

一场纯粹的逻辑对决

背景：五个维度，一个真相

在一条街上，有五栋颜色不同的房子，每一栋房子里都住着一个不同国籍的人，他们每个人喝不同的饮料，抽不同品牌的香烟，并养着不同的宠物



国籍

房子颜色

饮料

香烟

宠物

没有任何两栋房子在任何一个维度上是相同的

现有线索

1. 英国人住在红房子里。
2. 西班牙人养狗。
3. 绿房子里的人喝咖啡。
4. 乌克兰人喝茶。
5. 绿房子紧挨在象牙色房子的右边。
6. 抽Old Gold牌香烟的人养蜗牛。
7. 黄房子里的人抽Kools牌香烟。
8. 中间的房子里的人喝牛奶。
9. 挪威人住在第一间房子。
10. 抽Chesterfields牌香烟的人住在养狐的人的隔壁。
11. 抽Kools牌香烟的人住在养马的人的隔壁。
12. 抽Lucky Strike牌香烟的人喝橙汁。
13. 日本人抽Parliaments牌香烟。
14. 挪威人住在蓝色房子的隔壁。

Prolog编程

```
% 1. 定义整体结构：5 个房子
% 结构原型：house(颜色, 国籍, 宠物, 饮料, 香烟)
length(Houses, 5),
Houses = [house(_,_,_,_,_), house(_,_,_,_,_), house(_,_,_,_,_),
          house(_,_,_,_,_), house(_,_,_,_,_)],

% 2. 应用约束 (Constraints)
% [线索 1] 英国人住在红房子里
member(house(red, brit, _, _, _), Houses),
% [线索 2] 瑞典人养狗
member(house(_, swede, dog, _, _), Houses),
% [线索 3] 丹麦人喝茶
member(house(_, dane, _, tea, _), Houses),
% [线索 4] 绿房子紧挨着白房子，且在白房子的左边 (left_of)
left_of(house(green, _, _, _, _), house(white, _, _, _, _), Houses),
% [线索 5] 绿房子主人喝咖啡
member(house(green, _, _, coffee, _), Houses),
% [线索 6] 抽 Pall Mall 香烟的人养鸟
member(house(_, _, bird, _, pall_mall), Houses),
```

Prolog编程

```
% [线索 7] 黄房子主人抽 Dunhill 香烟
member(house(yellow, _, _, _, dunhill), Houses),
% [线索 8] 住在中间房子的人喝牛奶
Houses = [_, _, house(_, _, _, milk, _), _, _],
% [线索 9] 挪威人住在第一间房子
Houses = [house(_, norwegian, _, _, _) | _],
% [线索 10] 抽 Blends 香烟的人住在养猫的人隔壁 (next_to)
next_to(house(_, _, _, _, blends), house(_, _, cat, _, _), Houses),
% [线索 11] 养马的人住在抽 Dunhill 香烟的人隔壁
next_to(house(_, _, horse, _, _), house(_, _, _, _, dunhill), Houses),
% [线索 12] 抽 Blue Master 的人喝啤酒
member(house(_, _, _, beer, blue_master), Houses),
% [线索 13] 德国人抽 Prince 香烟
member(house(_, german, _, _, prince), Houses),
% [线索 14] 挪威人住在蓝房子隔壁
next_to(house(_, norwegian, _, _, _), house(blue, _, _, _, _), Houses),
% [线索 15] 抽 Blends 的人有一个喝水的邻居
next_to(house(_, _, _, _, blends), house(_, _, _, water, _), Houses),
% 4. 确保斑马(Zebra)和水(Water)存在于解中
member(house(_, _, zebra, _, _), Houses),
member(house(_, _, _, water, _), Houses).
```

Prolog编程

?- solve(Houses).

solve(Houses).

Houses =

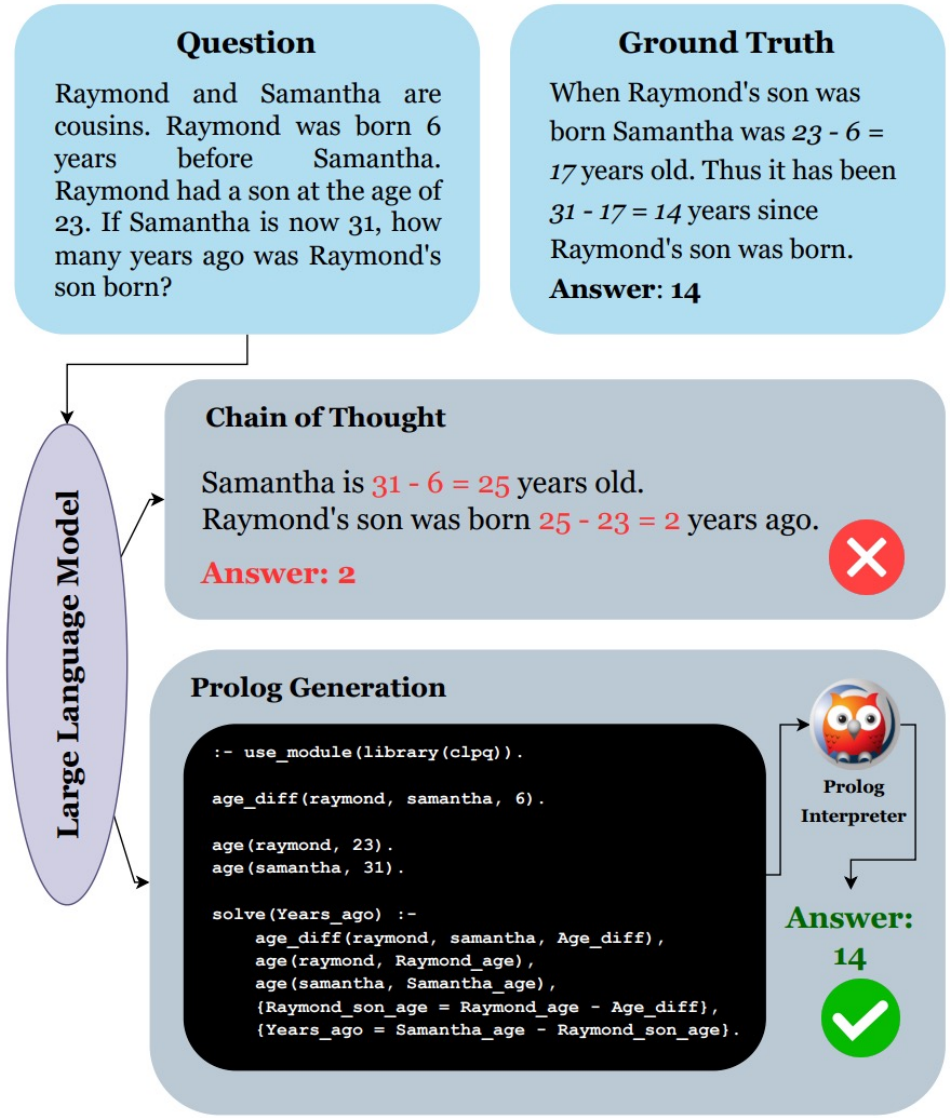
颜色	国籍	宠物	饮料	香烟
yellow	norwegian	cat	water	dunhill
blue	dane	horse	tea	blends
red	brit	bird	milk	pall_mall
green	german	zebra	coffee	prince
white	swede	dog	beer	blue_master

应用场景(二)：专家系统

```
hypothesis(流感) :-
    fluid,                  % 流鼻涕
    headache,               % 头痛
    fever,                  % 发烧
    body_ache.              % 身体酸痛

hypothesis(普通感冒) :-
    fluid,                  % 流鼻涕
    sneezing,               % 打喷嚏
    sore_throat,            % 喉咙痛
    \+ fever.               % 没有发烧 (\+ 表示 NOT)

hypothesis(麻疹) :-
    fever,                  % 发烧
    rash,                   % 皮疹
    conjunctivitis.         % 结膜炎 (红眼)

hypothesis(水痘) :-
    fever,                  % 发烧
    rash,                   % 皮疹
    body_ache.              % 身体酸痛

hypothesis(新冠肺炎_COVID19) :-
    fever,                  % 发烧
    cough,                  % 咳嗽
    shortness_of_breath,    % 呼吸急促
    loss_of_taste.          % 味觉丧失
```

```
start
----------------------------------------
欢迎使用简易医学诊断系统。
请根据提示回答 yes (y) 或 no (n)。
----------------------------------------
请问您有以下症状吗: has_fluid ? (y/n):
y
请问您有以下症状吗: has_headache ? (y/n):
y
请问您有以下症状吗: has_fever ? (y/n):
y
请问您有以下症状吗: has_body_ache ? (y/n):
y

诊断结果: 您可能患有 ** 流感 **
建议您尽快咨询专业医生。
true
```

应用场景(三)：LLM + Prolog




```prolog
% Raymond was born 6 years before Samantha.
age_difference(RaymondAge, SamanthaAge) :-
    RaymondAge is SamanthaAge + 6.

% Samantha is now 31.
samantha_age(31).

% Raymond had a son at age 23, so son's birth year relative to Raymond:
son_birth_year(RaymondAge, SonAge) :-
    SonAge is RaymondAge - 23.

% Query: how many years ago was Raymond's son born?
years_ago_son_born(YearsAgo) :-
    samantha_age(SamanthaAge),
    age_difference(RaymondAge, SamanthaAge),
    son_birth_year(RaymondAge, SonAge),
    YearsAgo is SonAge.
```

```
years_ago_son_born(X).

X = 14
```

```
?- years_ago_son_born(X).
```

Arithmetic Reasoning with LLM: Prolog Generation & Permutation. https://arxiv.org/pdf/2405.17893

Prolog简史

1965：Robinson开发出归结(resolution)程序

1973：马赛的Colmeraur用Fortan开发出Prolog语言

1974：Kowlaski的工作将谓词逻辑作为编程语言

提出著名的原则：程序=逻辑+控制

标志着逻辑编程(Logic Programming)的诞生

1977：爱丁堡大学在DEC10计算机上开发了Prolog解释器

1980：英国帝国理工学院为个人计算机开发了微Prolog语言

1981：日本第五代计算机系统采用Prolog作为主要程序设计语言

……

阿兰·罗宾逊 (1930—2016)

Programming Languages — J. J. Horning Editor

Algorithm = Logic + Control

Robert Kowalski
Imperial College, London

An algorithm can be regarded as consisting of a logic component, which specifies the knowledge to be used in solving problems, and a control component, which determines the problem-solving strategies by means of which that knowledge is used. The logic component determines the meaning of the algorithm whereas the control component only affects its effkiency. The effkiency of an algorithm can often be improved by improving the control component without changing the logic of the algorithm. We argue that computer programs would be more often correct and more easily improved and modified if their logic and control aspects were identified and separated in the program text.

开启逻辑编程之旅

常用工具

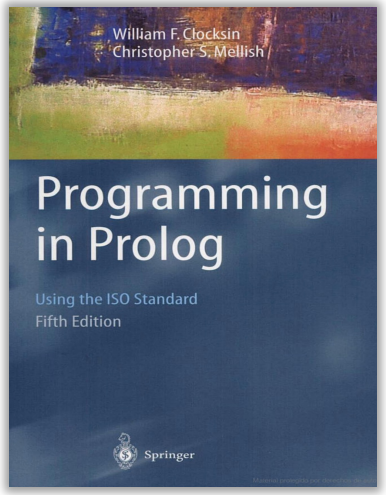
- SWI-Prolog
- 在线版本:SWI-Prolog Online (SWISH)
- 无需安装，直接在浏览器编写和运行

SWI Prolog

https://www.swi-prolog.org/
https://swish.swi-prolog.org/

深入学习

- 《Programming in Prolog》
  (Clocksin & Mellish)
  逻辑编程领域经典教材



https://athena.ecs.csus.edu/~mei/logicp/Programming_in_Prolog.pdf

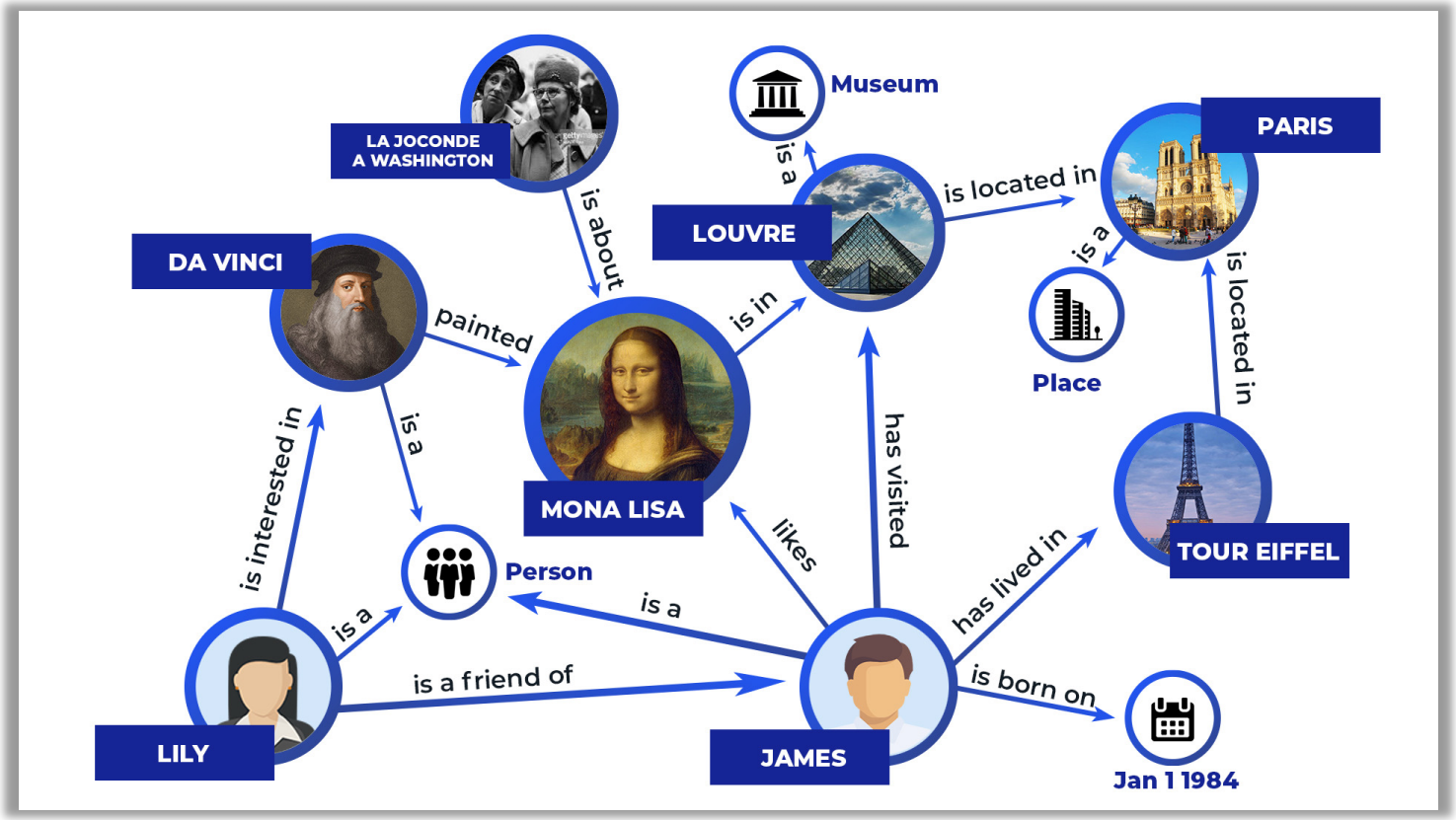
小结

- 掌握逻辑推理的基本类型
- **掌握逻辑智能体的三个核心要素：符号、推理、 感知**
- 掌握命题逻辑、一阶逻辑的基本推理规则
- 了解基本的逻辑推理算法，如归结、前向链接、后向链接等
- **学会使用Z3等SMT Solver 、学会使用Prolog语言**

知识图谱(Knowledge Graph)

知识图谱

知识图谱是一种以图结构来描述知识和建模事物之间关联关系的技术方法



知识图谱于 2012 年 由 Google 提出，其初衷是利用网络多源数据构建的知识库来增强语义搜索，提升搜索质量和体验

知识图谱已被广泛应用于智能搜索、智能问答、个性化推荐、内容分发等领域

从语义搜索认识知识图谱

李奥納多·達文西

译者 ⋮

李奧納多·達文西，又譯達芬奇，全名李奧納多·迪·塞爾·皮耶羅·達文西，是義大利文藝復興時期佛羅倫斯共和國的博學者：在繪畫、音樂、建築、數學、幾何學、解剖學、生理學、動物學、植物學、天文學、氣象學、地質學、地理學、物理學、光學、力學、發明、土木工程等領域都有顯著的成就。 维基百科

出生信息： 1452 年 4 月 15 日，義大利Anchiano

逝世于： 1519 年 5 月 2 日，法國昂布瓦斯克勞斯·呂斯城堡

系列作品： 聖母像， 麗達與天鵝

建筑作品： Leonardo's Rivellino

画风： 文藝復興， 義大利文藝復興， 文藝復興全盛期

祖先： 瑟皮耶羅·達文西， 卡特琳娜·達文西， 安東尼奧·達·溫奇， Lucia di ser Piero di Zoso da Bacchereto

艺术作品


蒙娜麗莎
1503 年


最後的晚餐
1498 年


救世主
1500 年


維特魯威人

展出地点


Royal Collection…


Museo Galileo
佛羅倫斯


羅浮宮
巴黎


國家藝廊
華盛頓哥倫比亞特區

用户还搜索了


米開朗基羅


拉斐爾


李奧納多·狄卡皮歐


文森·梵谷

南京大學

網站 | 行車路線 | 儲存 | 致電

中国南京市的大学

南京大學，簡稱南大，位於中國江蘇省南京市，該校歷史或可追溯至三國吳永安元年，歷史上曾歷經多次變遷，亦是中國第一所集教學和研究於一體的現代大學。中華民國政府撤離南京後，中華人民共和國成立前夕由「國立中央大學」易名「國立南京大學」，翌年逕稱「南京大學」，沿用至今。 维基百科

地址： 中国江苏省南京市鼓楼区 邮政编码: 210093

电话号码： +86 25 8359 3186

总裁： Chen Jun

招生人数： 35,434 (2007 年)

创立于： 1902 年

子公司： 南京大學物理學院， 更多

省份： 江蘇省

免责声明

近期活动

10月20日 週五　International Conference on Integrated Circ…

知名校友

查看更多项目 (超過 45 項)

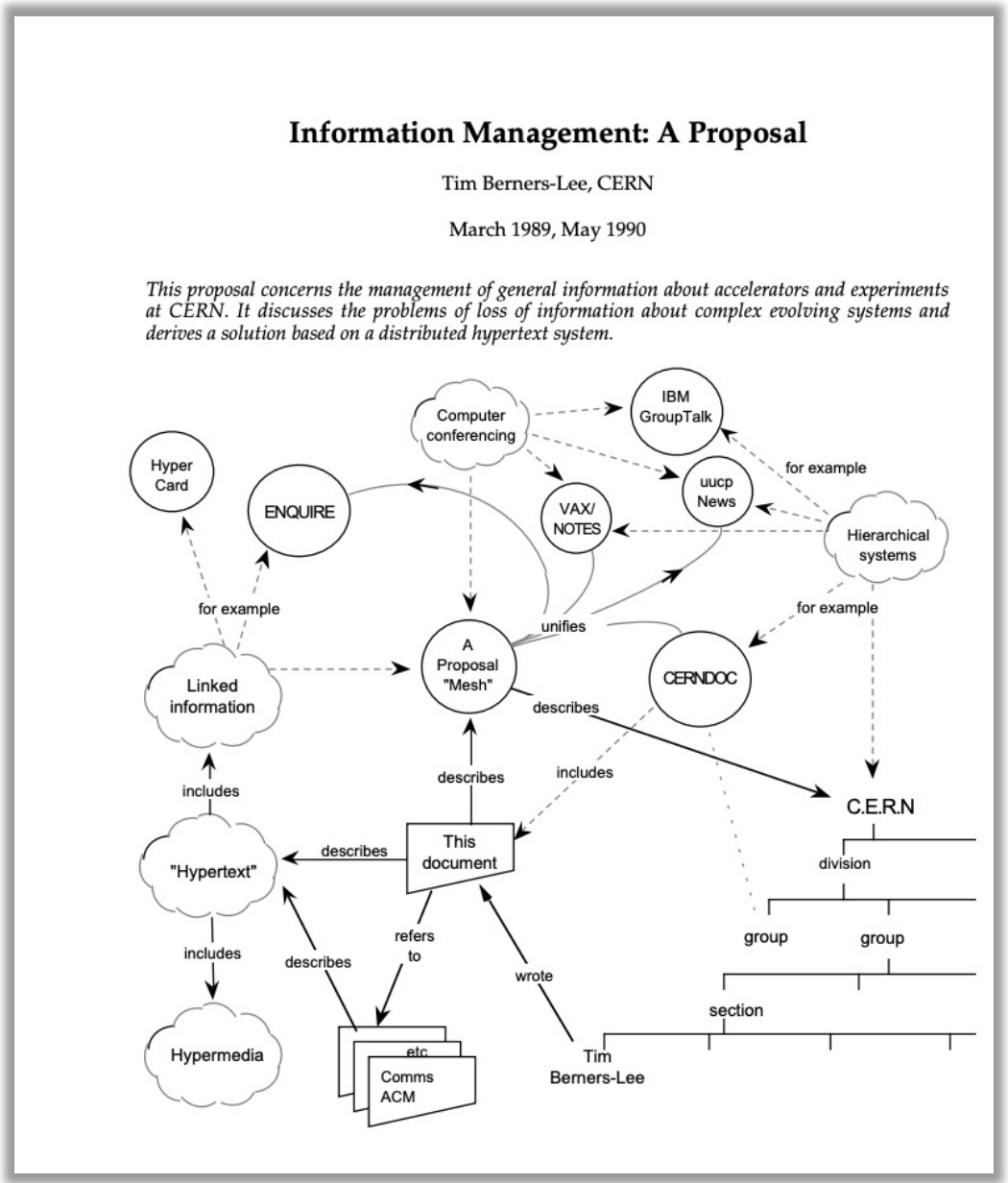

趙忠堯


李國鼎


張翔


Jingguang Chen

万维网



Information Management: A Proposal

Tim Berners-Lee, CERN

March 1989, May 1990

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext system.

万维网（world wide web) 是以链接为中心的信息系统

Linked information system

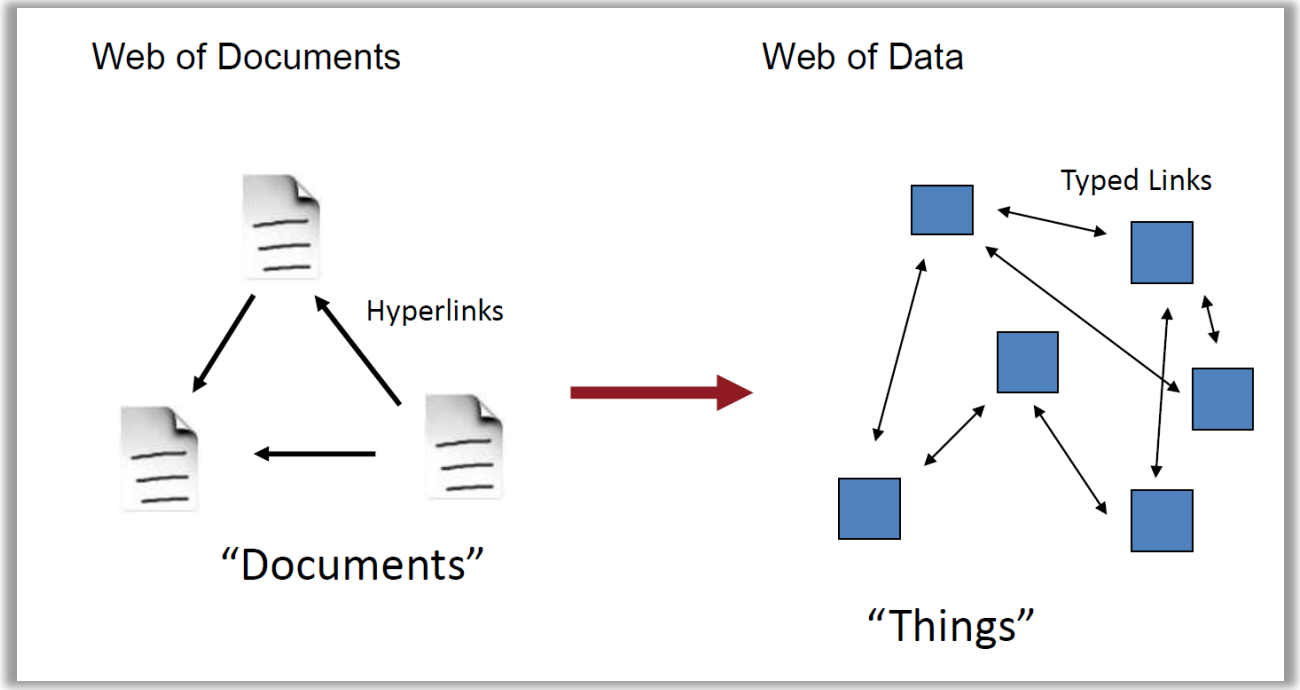
万维网以**文本**和**链接**描述信息，用户通过超链接浏览互联网上的各类资源，也可以通过互联网将自己的信息发布出去

Tim Berners-Lee
2016年图灵奖

语义万维网

语义万维网(semantic web)中，信息内容具有良好的语义定义，

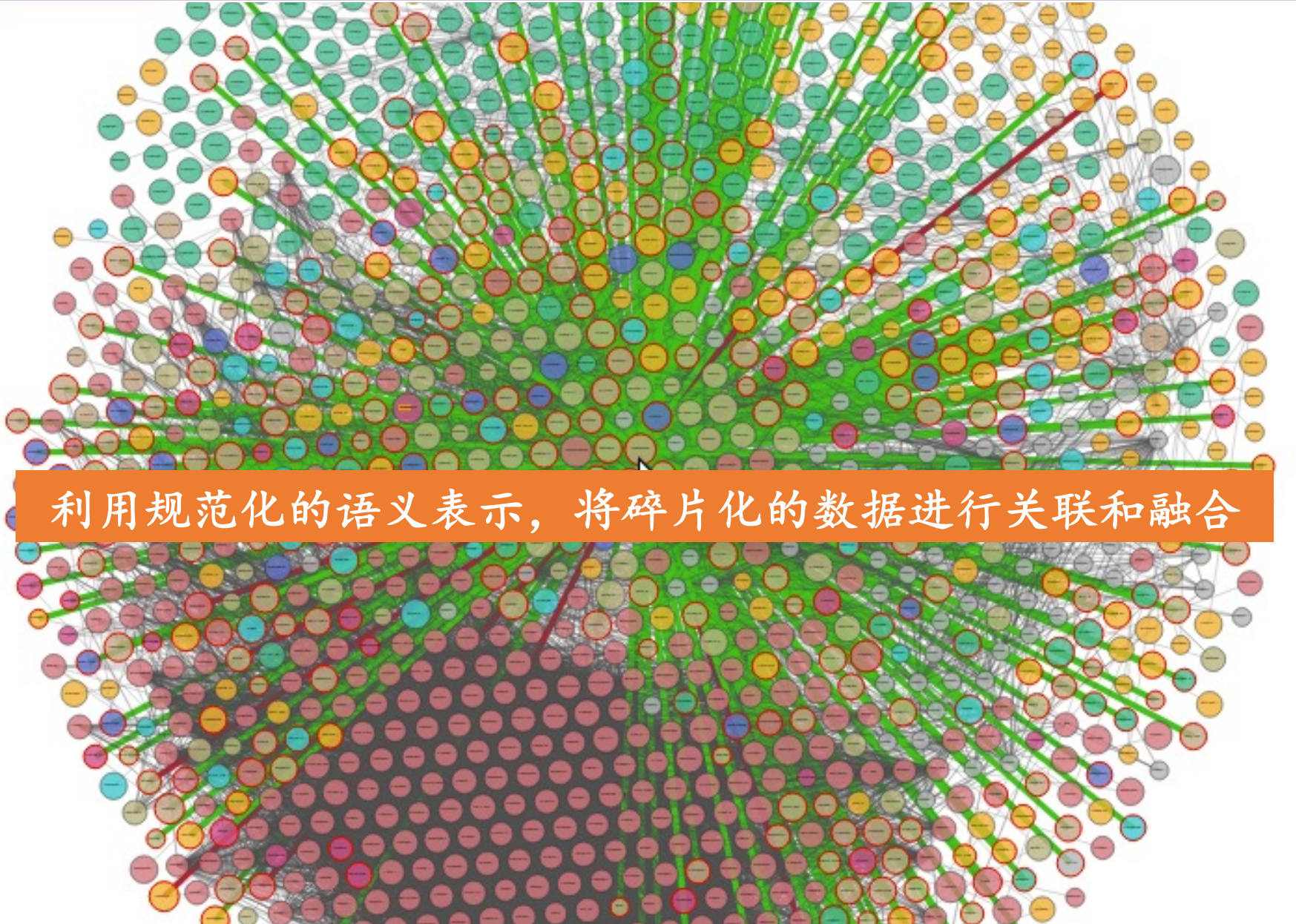
计算机可以理解并自动存取语义信息



语义网与万维网的主要区别

- ✓ 万维网是关于网页链接的结构，节点是网页，节点之间通过超链接连接
- ✓ 语义网是关于语义内容结构化表示的图结构，节点是语义信息

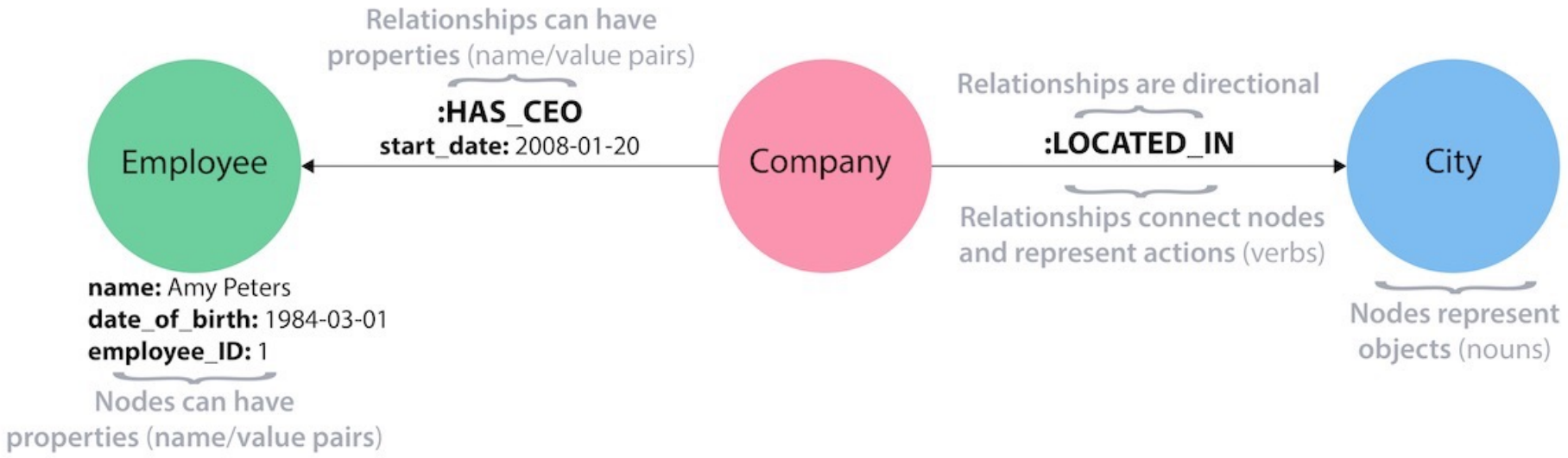
Linked Data



利用规范化的语义表示，将碎片化的数据进行关联和融合

属性图(Property Graph)

- 属性图由节点(Nodes)、关系(Relationships)、属性(Property)和标签(Labels)组成
- 节点上包含属性，属性可以以任何键值形式存在
- 关系连接节点，每个关系都有拥有一个方向、一个标签、一个开始节点和结束节点
- 关系也可以有属性，即边属性，可以通过在关系上增加属性提供有关边的元信息，如创建时间

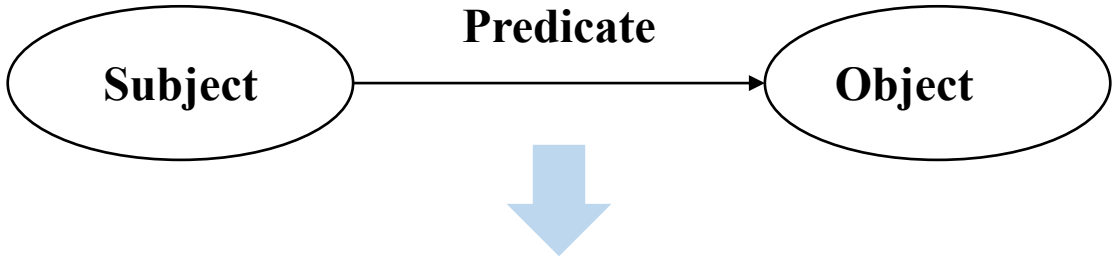


资源描述框架RDF

RDF表示 **R**esource **D**escription **F**ramework (资源描述框架)

国际万维网联盟W3C推动的面向Web的语义标准

RDF的基本组成单位是三元组 SPO

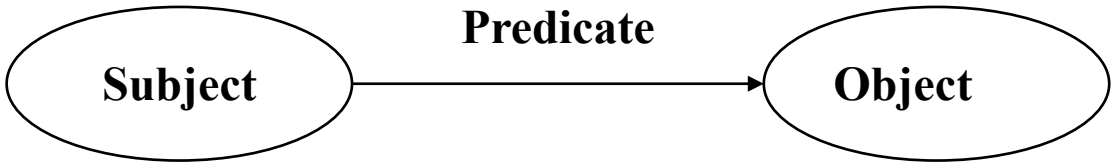


（subject（主），predicate（谓词），object（宾语））

（subject（南京大学），predicate（位于），object（江苏））

RDF图

基本数据模型：有向标记图



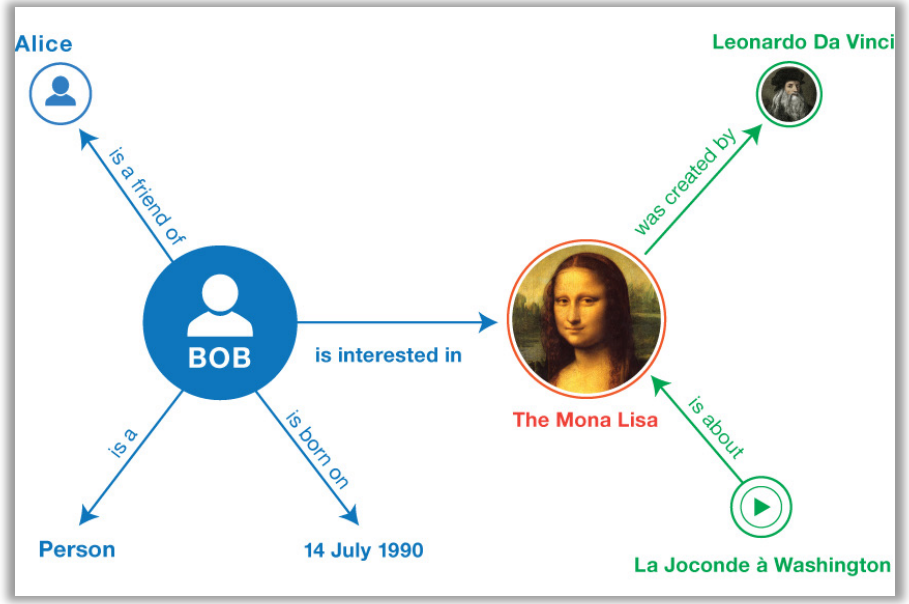
<Bob> <is a> <person>

<Bob> <is a friend of> <Alice>

<Bob> <is born on> <the 4th of July 1990>

<Bob> <is interested in> <the Mona Lisa>

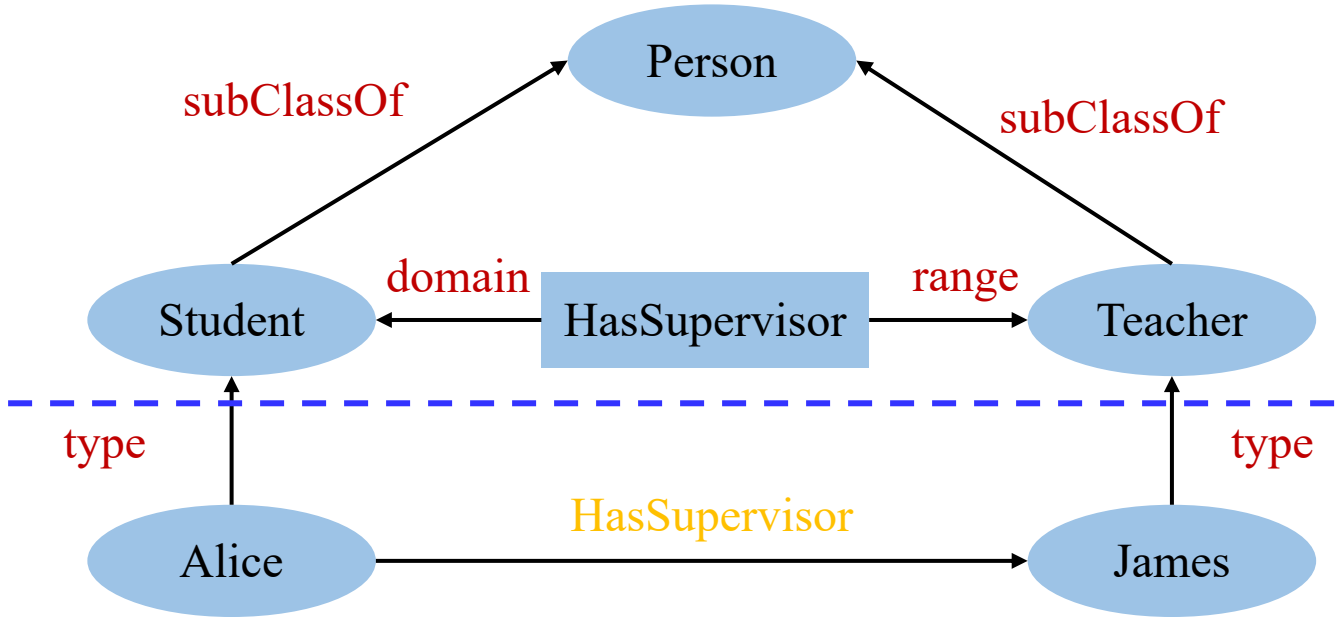
<the Mona Lisa> <was created by> <Leonardo da Vinci>



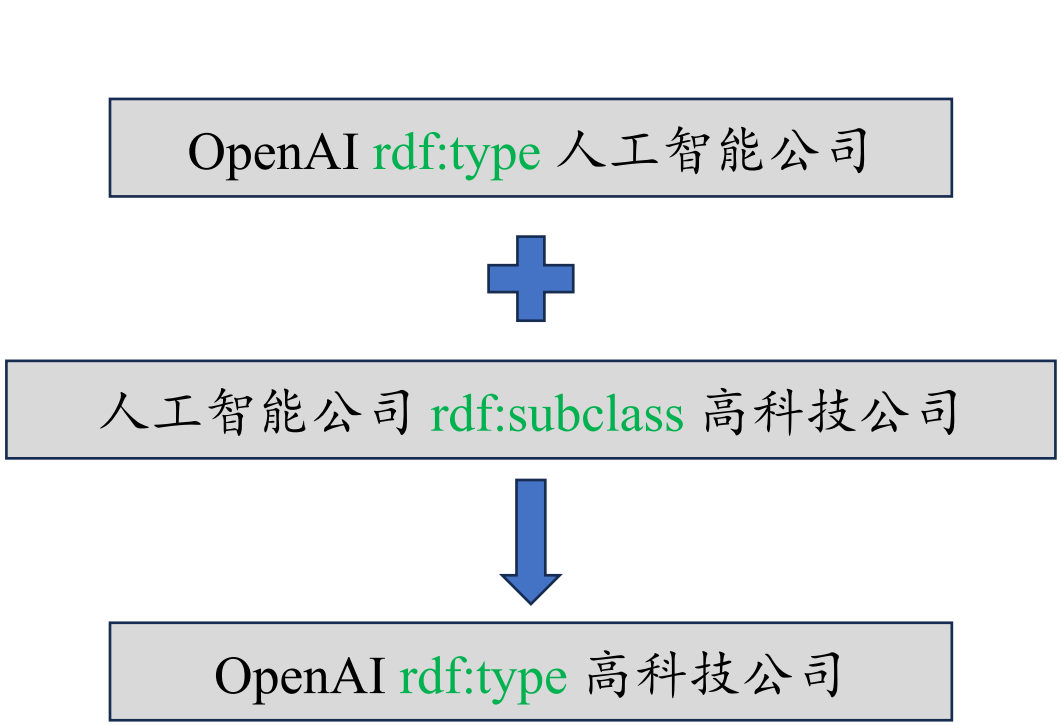
RDF Schema

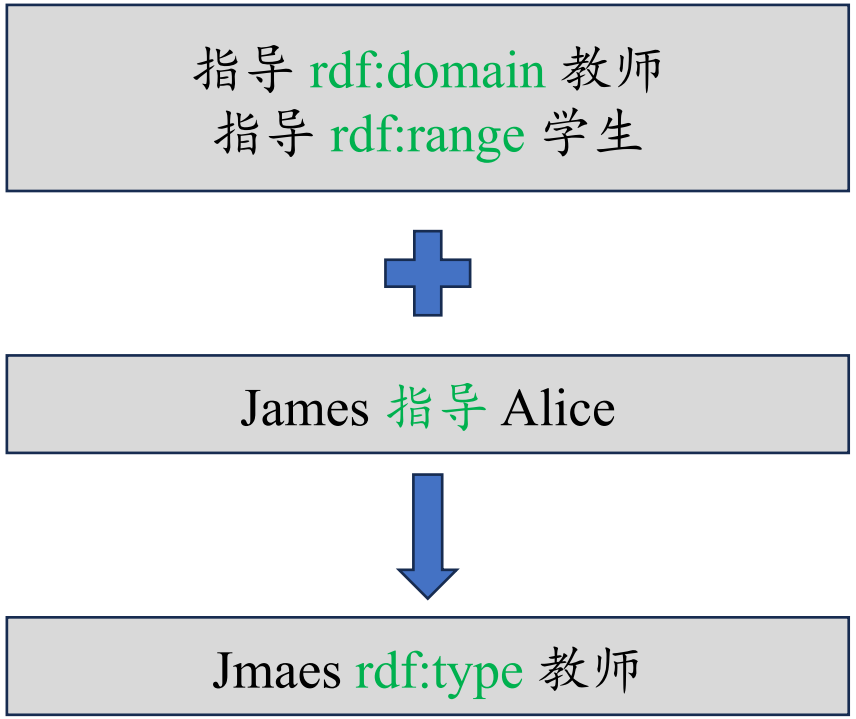
RDFS(Resource Description Framework Schema)是用来定义RDF中的类和属性语义的描述性语言，定义了资源间的继承关系及属性约束等

1. rdfs:Class. 用于定义类
2. rdfs:domain. 用于表示该属性属于哪个类别
3. rdfs:range. 用于描述该属性的取值类型
4. rdfs:subClassOf. 用于描述该类的父类
5. rdfs:subProperty. 用于描述该属性的父属性



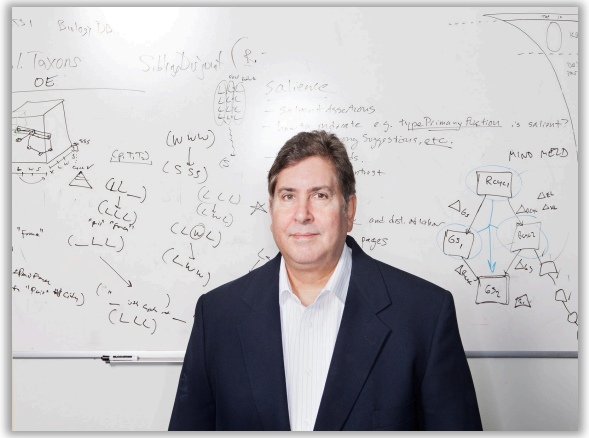
基于RDFS的简单推理





典型知识图谱项目：CYC

- 取自英文单词“百科全书”（encyclopedia），目标是把人类的常识编码，建成知识库
- 典型的常识知识如“Every tree is a Plant”，“People die eventually”等
- Cyc知识库主要由术语Terms和断言Assertions组成
- Cyc目前有两个版本，企业版和研究版，研究版对研究人员开放



雷纳特（1950-）

典型知识图谱项目：WordNet

- WordNet是著名的词典知识库，主要用于词义消歧，由普林斯顿大学认知科学实验室从1985年开始研发

https://wordnet.princeton.edu/

- WordNet主要定义了名词、动词、形容词和副词之间的语义关系

- 例如：
  - ✓ 名词之间的上下位关系（如：“犬科动物”是“狗”的上位名词）
  - ✓ 动词之间的蕴含关系，（如：“打鼾”蕴含着“睡眠”）

- WordNet3.0已经包含超过15万个词和20万个语义关系

典型知识图谱项目：ConceptNet

- ConceptNet是常识知识库。最早源于MIT媒体实验室的Open Mind Common Sense（OMCS）项目，OMCS项目是由著名人工智能专家Marvin Minsky于1999年提议创立
- ConceptNet主要依靠互联网众包、专家创建来构建，新版本也导入了大量开放的结构化数据，如Dbpedia、Wikinary、Wordnet等
- ConceptNet以三元组形式的关系型知识组成，ConcepNet5版本中已经包含由2800万关系描述
- 与谷歌知识图谱比，ConcepNET比较侧重词与词之间的关系，更加接近于WordNet，但又比WordNet包含的关系类型更多
- ConceptNet完全免费开放、支持多种语言

典型知识图谱项目：Freebase

截至2014年年底，Freebase共包含6800万个实体、10亿条关系、超过24亿事实三元组信息

2016年谷歌对freebase停止更新，把所有数据捐给WikiData

典型知识图谱项目：WikiData

目标是构建全世界最大的免费知识库，采用CC0完全自由许可协议

维基基金会

开放社区贡献

https://www.wikidata.org/wiki/Wikidata:Main_Page

典型知识图谱项目：BabelNet

- BabelNet是类似于WordNet的多语言词典知识库，目标是解决WordNet在非英语语种中数据缺乏的问题，采用的方法是将WordNet词典与Wikipedia百科集成
- 首先建立WordNet中的词与Wikipedia的页面标题的映射，然后利用Wikipedia中的多语言链接，再辅以机器翻译技术，来给WordNet增加多种语言的词汇
- BabelNet包含了271中语言，1400万同义词组，36.4万词语关系和3.8亿从Wikipedia中抽取的链接关系，集成了WordNet在词语关系上的优势和Wikipedia在多语言方面的优势，是目前最大的多语言词典知识库

典型知识图谱项目：NELL

Never-Ending Language Learner (NELL)

永不停歇的语言学习者

- NELL是卡内基梅隆大学开发的知识库，采用互联网挖掘的方法
  从Web自动抽取三元组知识

- 基本理念：给定一个初始的本体（少量类和关系的定义）和少量样本，让机器能够
  通过自学习的方式不断的从Web学习和抽取新的知识

- 目前NELL已经抽取了500多万条三元组知识

典型知识图谱项目：YAGO

- YAGO是德国马克斯-普朗克研究所 (Max Planck Institute, MPI) 构建的大型多语言知识库

- YAGO主要继承了Wikipedia、WordNet和GeoNames三个来源的数据。YAGO将WordNet的词汇定义与Wikipedia的分类体系进行了融合，使其具有更加丰富的实体分类体系

- YAGO考虑了时间和空间知识，为很多知识条目增加了时间和空间维度的属性描述

https://yago-knowledge.org/

中文领域开放知识图谱

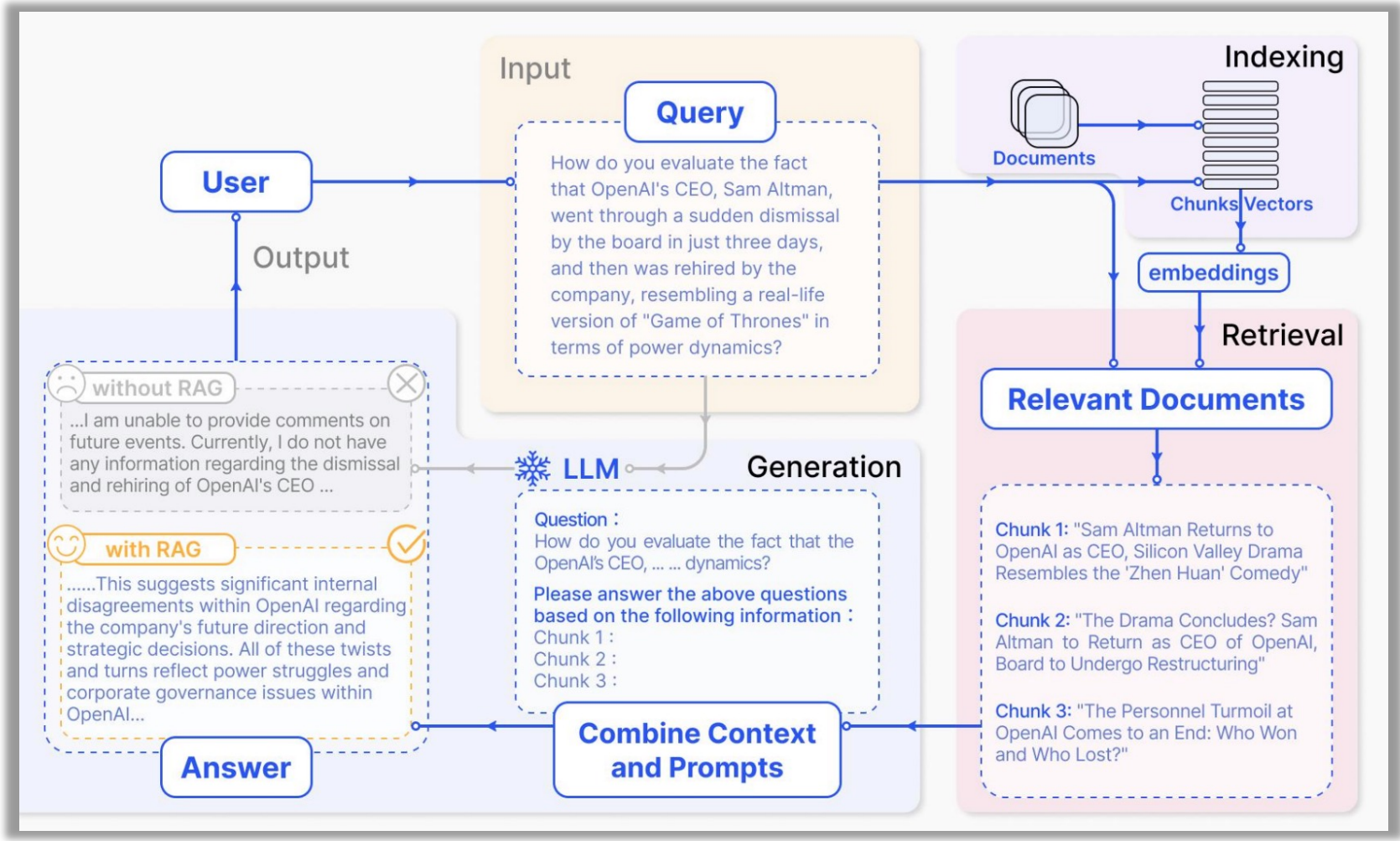
http://www.openkg.cn/

应用场景：大模型时代的知识图谱

Retrieval-Augmented Generation (RAG)

检索增强

- 在回答问题或生成文本时，LLM首先从大量文档中检索相关信息，然后基于这些信息生成答案
- 通过附加外部知识库，提升大模型回复的可靠性与准确性
- 尤其适用于知识密集型任务

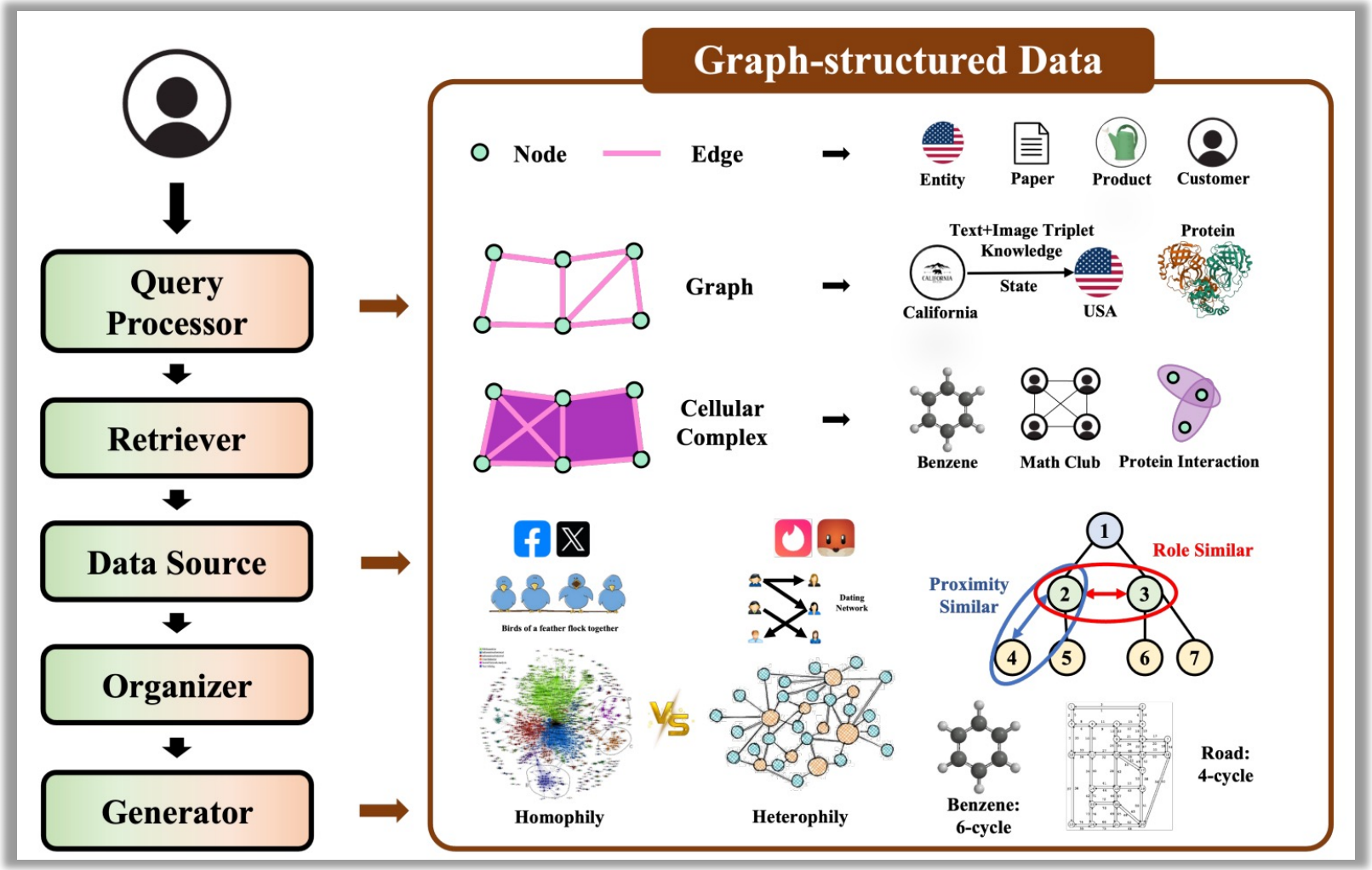


应用场景：大模型时代的知识图谱

Graph RAG

基于知识图谱进行检索增强

- 用户提问：“特朗普的女儿的母校在哪里?”
- KG 多跳推理：特朗普--daughter--> Ivanka Trump --AlumniOf--> University of Pennsylvania --Location--> 费城
- 将推理路径喂给 LLM，生成回答
- KG 提供**精准事实**和**可解释性**，LLM 提供**自然语言接口**和**泛化能力**



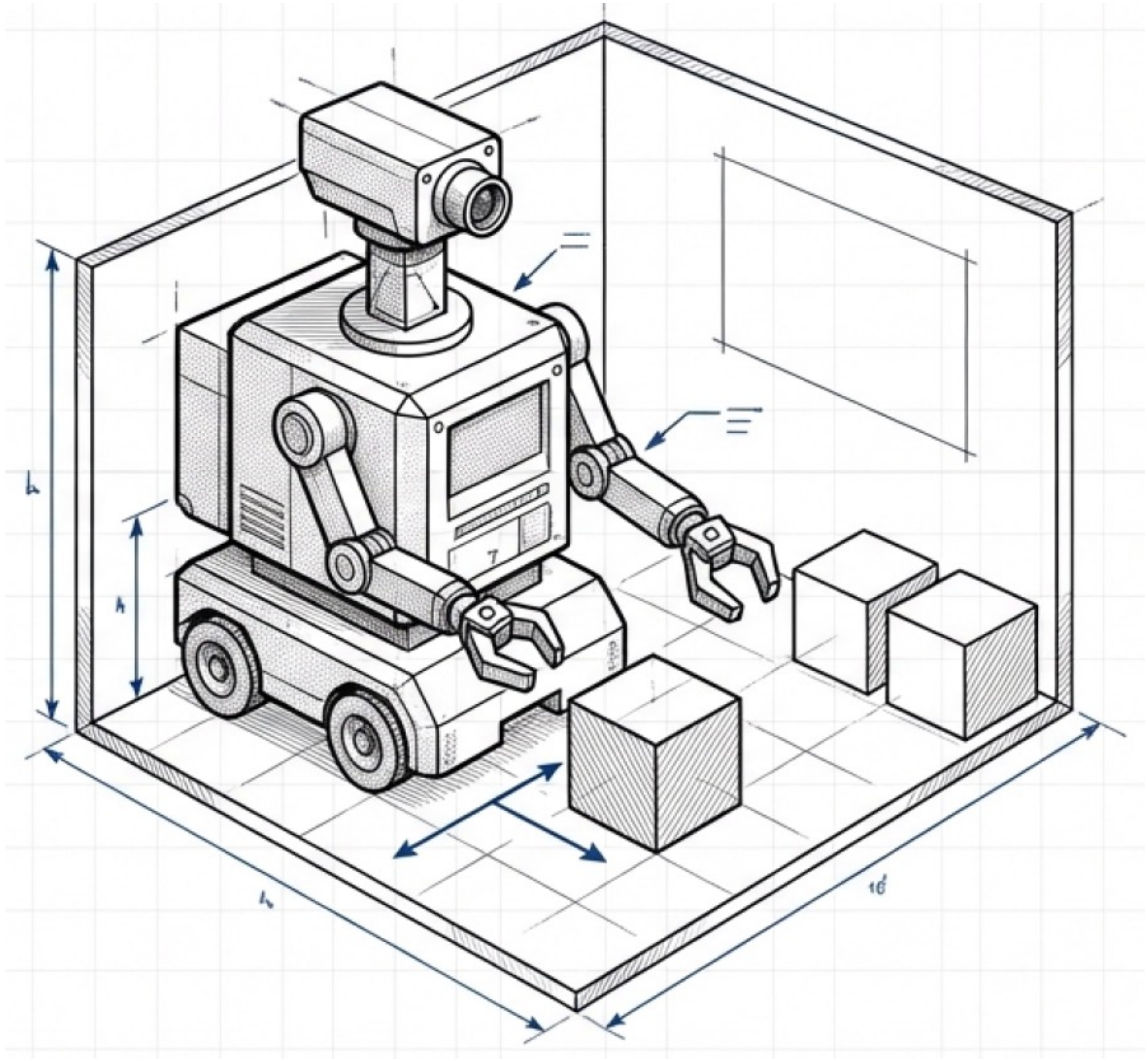
自动规划(Automated Planning)

自动规划(Automated Planning)

自动规划：智能体如何决定下一步做什么

一个智能体在特定环境中，为了达到一个明确的目标状态(Goal State)，从一个初始状态(Initial State)开始，自动生成一系列有效行动(Actions)序列的过程

人工智能领域的一个根本性问题



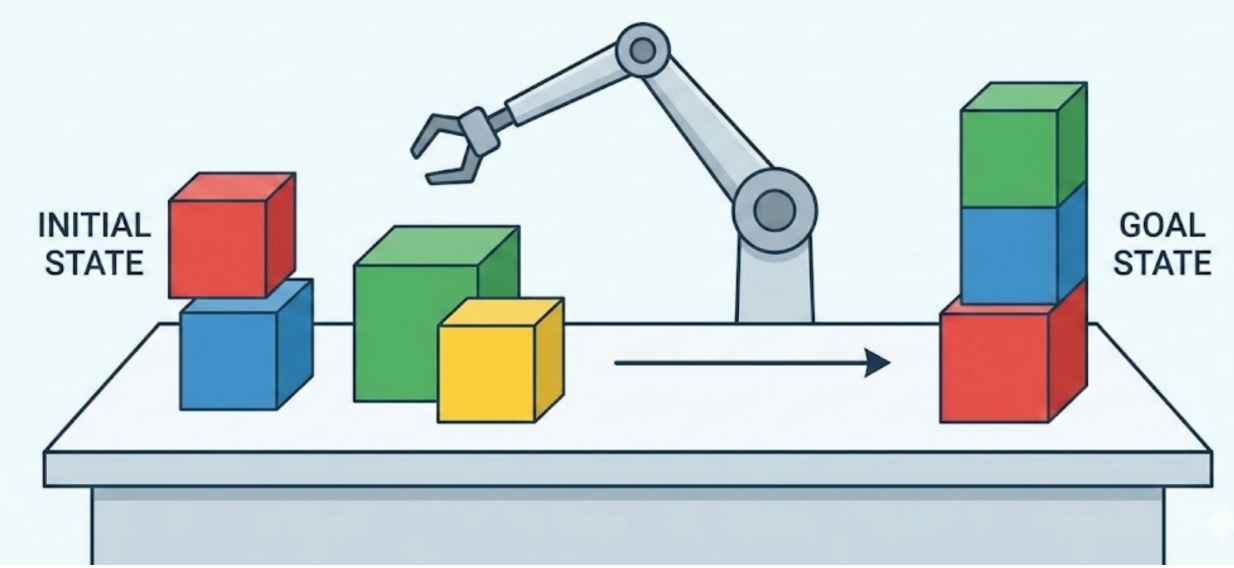
关键案例

先驱：Shakey the Robot (1969)

第一个能够推理自身行为，并与物理世界交互的通用移动机器人，能够自主规划路径和动作完成“推箱子”任务

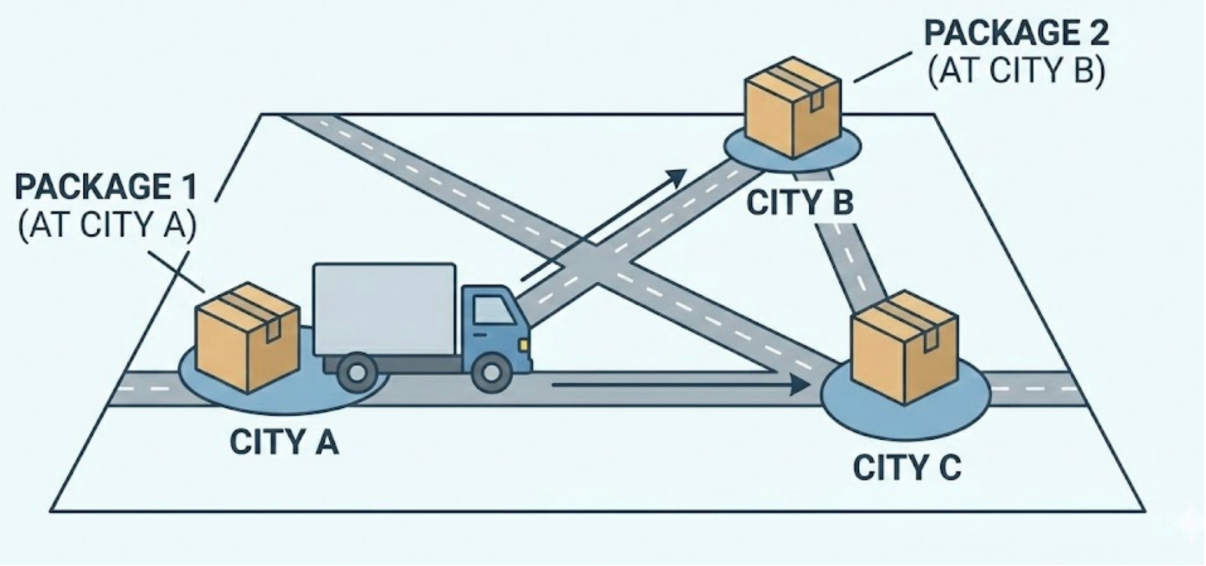
自动规划(Automated Planning)

积木世界(Blocks World)



规划机械臂的一系列动作，将木块从初始状态移动到目标状态

物流运输(Logistics)



规划卡车的行驶路线和包裹的装卸操作，将所有包裹从初始位置运输到指定城市

描述世界：建立形式化语言

为了让计算机理解世界并推理，必须将现实世界抽象成精确的、机器可读的规则

- STRIPS语言 (Stanford Research Institute Problem Solver)
- 核心思想：利用一阶逻辑来描述世界状态与动作

- 状态(State)

一组逻辑文字，是当前世界中所有为“真”的事实

例如，在积木世界中：

State: { On(A, Table), On(B, Table), On(D, C), ……}

描述世界：建立形式化语言

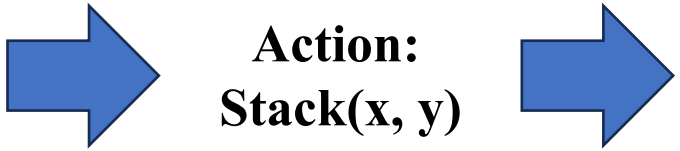
- 动作(Action)：定义了状态如何改变，每个动作都有明确的前提(Preconditions)和效果(Effects)

例：Stack(x, y)，将积木x堆叠到积木y上

前提(Preconditions)

执行动作前必须为真的条件

- Clear(y)：积木y上方必须是空的
- Holding(x)：机械臂必须正抓着积木x



Action:
Stack(x, y)

效果(Effects)

动作执行后世界发生的变化

添加表(Add List)：新增为真的事实

- **On(x, y)**
- **Clear(x)**
- **ArmEmpty()**

删除表(Add List)：不再为真的事实

- **Clear(y)**
- **Holding(x)**

PDDL：规划领域的通用语言

PDDL(Planning Domain Definition Language) 是定义规划问题的标准语言，是STRIPS的扩展和标准化

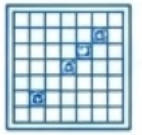
- 核心结构：分离“物理法则”和“具体任务”

Domain文件(.pddl)

定义世界通用的规则，包含谓词(Predicates)，如“On”，“Clear”和动作模版(Action Schemas)，如Stack(x, y)

```
(:action stack
  :parameters (?x ?y)
  :precondition (and (clear ?y) (holding ?x))
  :effect (and (on ?x ?y)
               (clear ?x)
               (arm-empty)
               (not (holding ?x))))
```

Problem文件(.pddl)

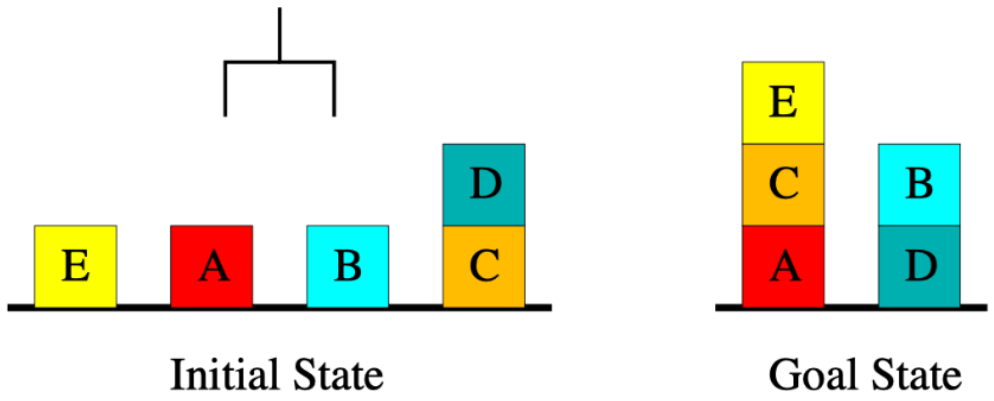


定义一个具体的、待解决的场景，包含对象(Objects)，如Block A, Block B；初始状态(Initial State)和目标状态(Goal State)

```
(:objects blockA blockB)
(:init (on-table blockA)
       (on-table blockB)
       (clear blockA)
       (clear blockB)
       (arm-empty))
(:goal (on blockA blockB))
```

PDDL: Blocks World

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:types block)
  (:predicates (on ?x - block ?y - block)
               (ontable ?x - block)
               (clear ?x - block)
               (handempty)
               (holding ?x - block))
  (:action pick-up
     :parameters (?x - block)
     :precondition (and (clear ?x) (ontable ?x) (handempty))
     :effect (and (not (ontable ?x)) (not (clear ?x))
                  (not (handempty)) (holding ?x)))
  (:action stack
     :parameters (?x - block ?y - block)
     :precondition (and (holding ?x) (clear ?y))
     :effect (and (not (holding ?x)) (not (clear ?y))
                  (clear ?x) (handempty) (on ?x ?y)))
  (:action unstack
     :parameters (?x - block ?y - block)
     :precondition (and (on ?x ?y) (clear ?x) (handempty))
     :effect (and (holding ?x) (clear ?y) (not (clear ?x))
                  (not (handempty)) (not (on ?x ?y))))
  (:action put-down
     :parameters (?x - block)
     :precondition (holding ?x)
     :effect (and (not (holding ?x)) (handempty) (clear ?x) (ontable ?x)))
)
```




```
(define (problem bw-abcde)
(:domain blocksworld)
(:objects a b c d e)
(:init (on-table a) (clear a)
       (on-table b) (clear b)
       (on-table e) (clear e)
       (on-table c) (on d c) (clear d)
       (arm-empty))
(:goal (and (on e c) (on c a) (on b d))))
```

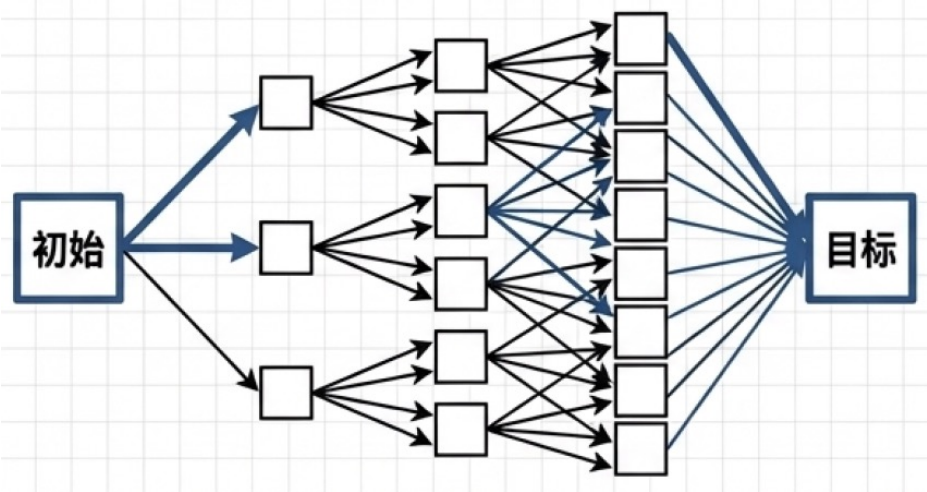
规划算法的两种基本思路

有了世界的描述，如何找到从“初始”到“目标”的动作序列?

前向规划(Forward Search)

思路：从初始状态出发，不断应用所有可行的动作，探索未来的状态，直到满足目标状态

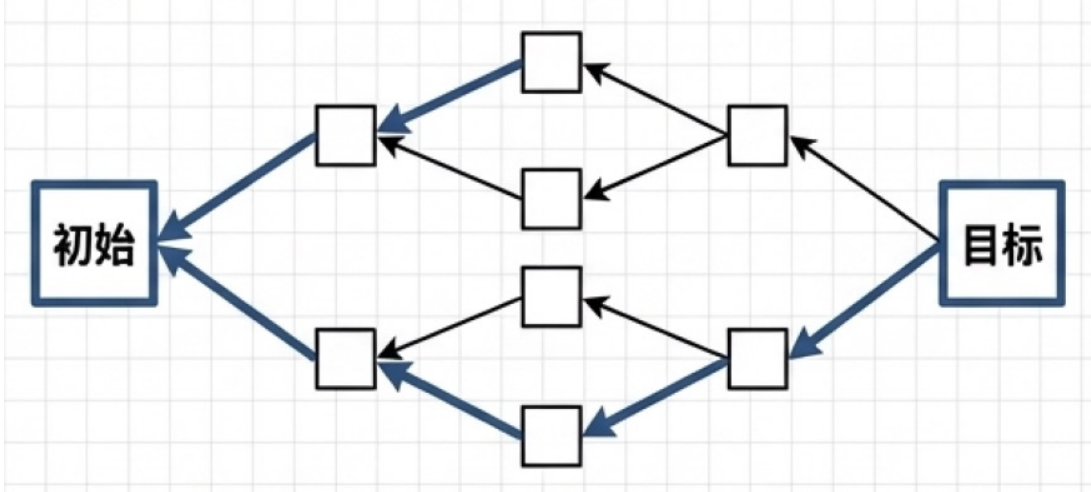
- 优点：直观，易于实现
- 缺点：分支因子巨大，会产生大量与目标无关的动作和状态



后向规划(Backward Search)

思路：从目标状态出发，反向应用动作，寻找能达成该目标的前置状态，直到达到初始状态

- 优点：目标驱动，只考虑目标相关动作，分支因子通常更小
- 缺点：处理复杂的目标组合时逻辑更复杂

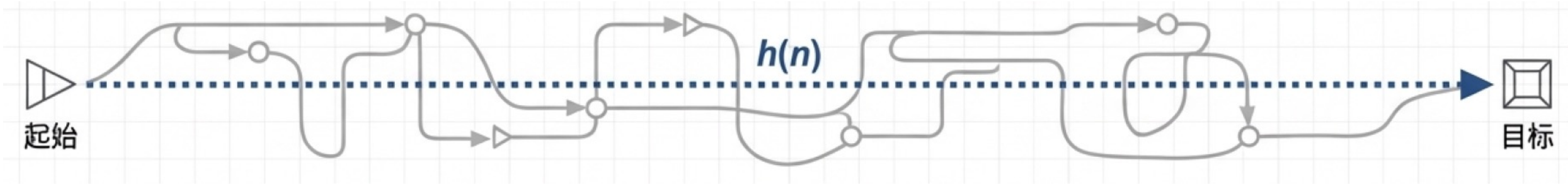


启发式搜索让规划变得更高效

将规划问题转化为启发式搜索问题，成功的关键在于如何设计一个好的启发式函数，用于估算当前状态到目标的距离

设计松弛问题(relaxed problem)

如果我们作弊，让问题变得更简单，那么解决这个简单问题的成本就可以作为原问题的成本下界估计



1. 忽略前提启发式

假设所有动作都没有前提条件，可以随时执行，计算达成目标所需要的最少动作数量

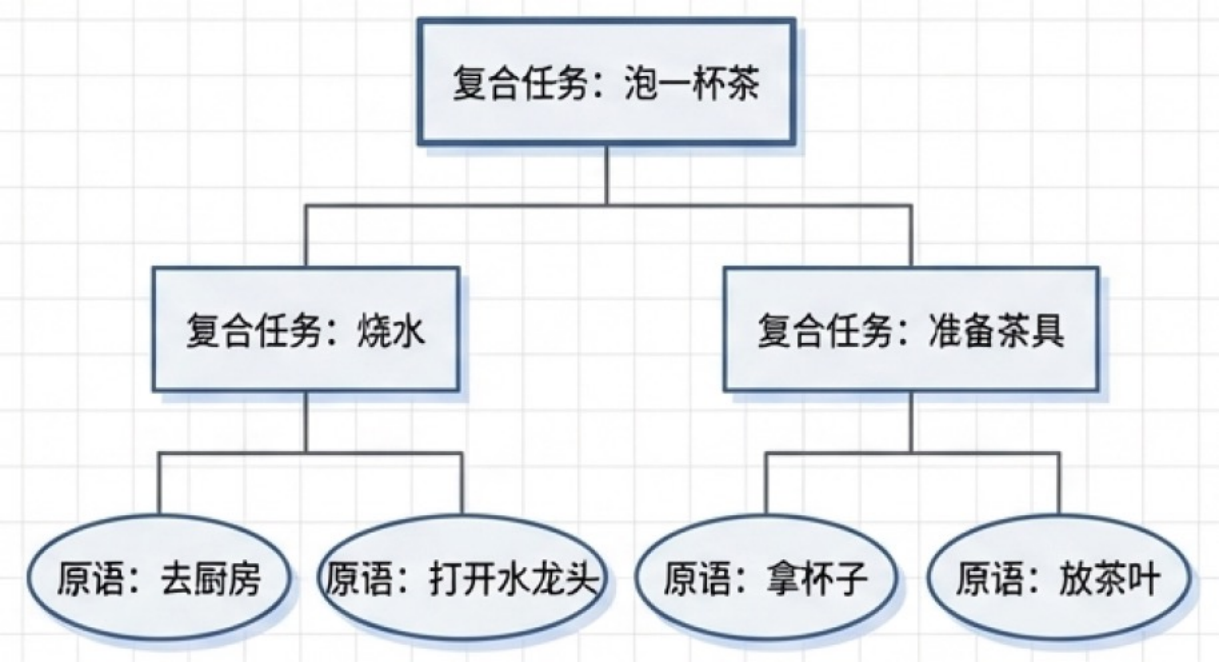
2. 忽略删除效果启发式

假设动作只有添加效果，没有删除效果，被称为单调松弛(Monotonic Relaxation)，一旦某个事实为真，将永远为真

超越原子动作：像人类一样分层思考

分层任务网络(Hierarchical Task Network, HTN)

传统规划器思考的是一系列原子动作，人类则不同，我们先思考高层任务(如泡一杯茶)，在将其分解为子任务(如烧水、放茶叶)

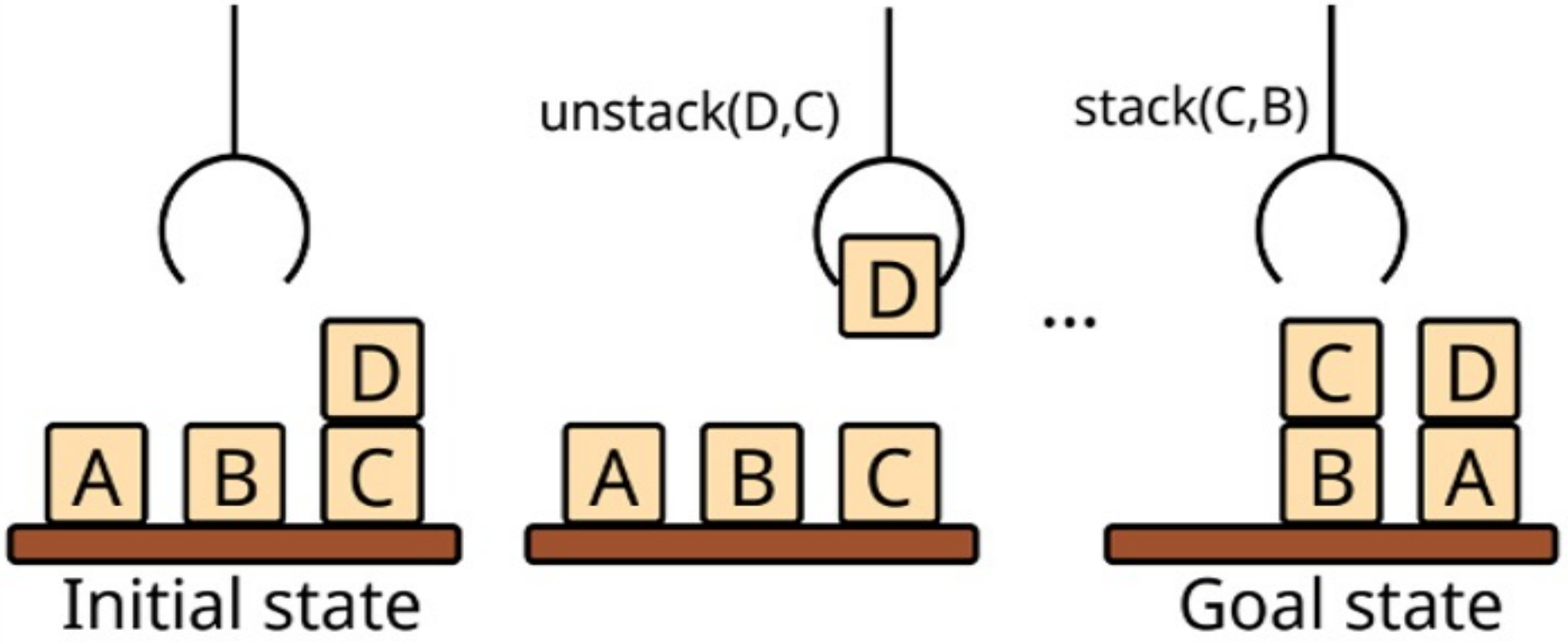


HTN的关键结构

- 复合任务：高层次的、可分解的目标
- 方法：如何将一个复合任务分解为一系列更小的子任务
- 原语任务(Primitive Tasks)：不可再分解的原子动作，对应PDDL中的动作

基于SMT求解Blocks World

SMT Solver 可以求解逻辑约束满足性问题，如何把Blocks World转换成类似问题？

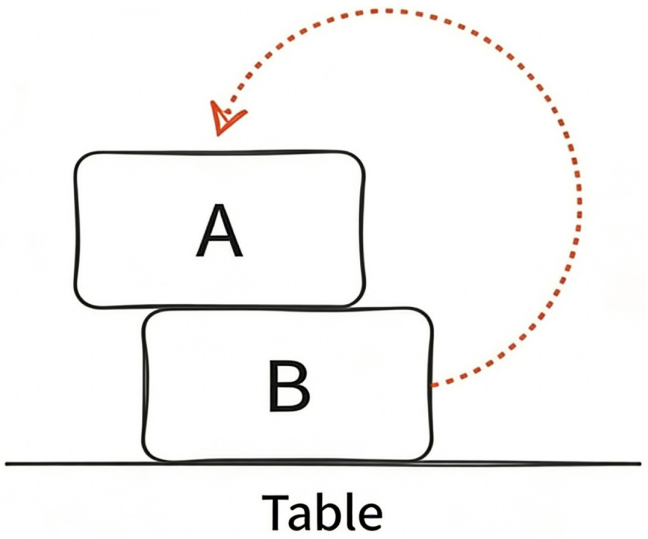


挑战

如何把一个动态的物理谜题，转化为一个静态的逻辑问题，交于机器求解

动态的物理世界

这是一个关于移动、时间和改变的故事，规则是直观的、物理的



静态的逻辑世界

我们的“读者”是逻辑求解器Z3，以为只懂真与假的逻辑大师。它无法理解移动，但是能判断一个庞大逻辑公式的对错

Z3 SMT Solver

$$(P \wedge Q) \Rightarrow R$$

我们的任务：将“规划问题”转化为“逻辑可满足性问题”

积木世界

对 象 (The Objects)

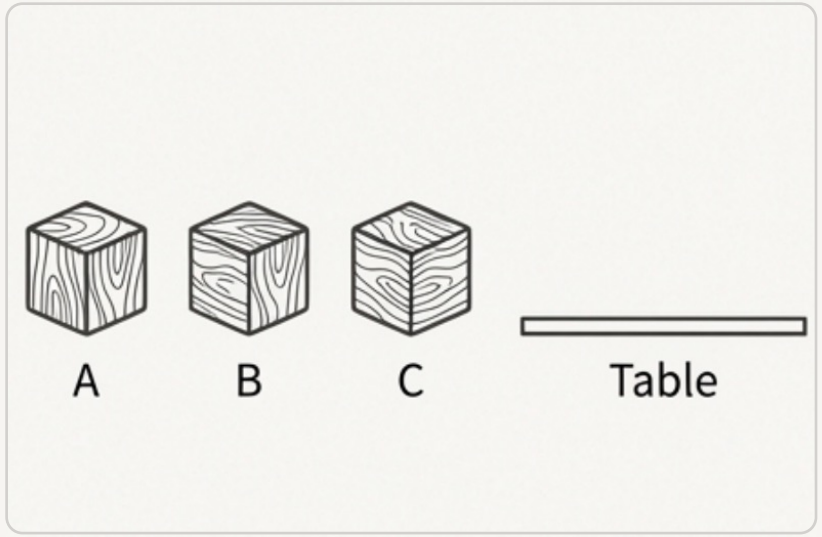
我们的世界由两种基本元素构成

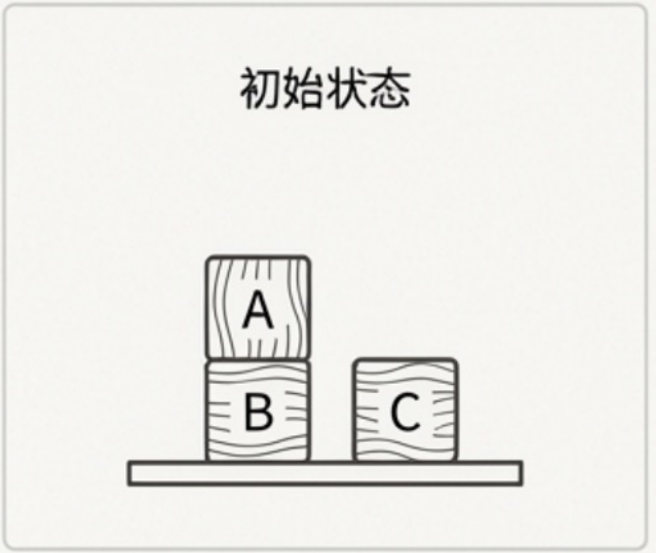
状 态 (The State)

一个状态就是积木在某一时刻的堆叠关系

目 标 (The Goal)

找到一系列合法的移动操作，从初始状态到达目标状态





核心思想：如何用逻辑来描述从初始状态到目标状态的每个瞬间

用逻辑谓词定义世界

为了让逻辑求解器理解一个随时间变化的世界，我们必须将时间作为一个显式维度加入我们的描述中。假设整个过程在T个时间步内完成

状态谓词：On(b, o, t)

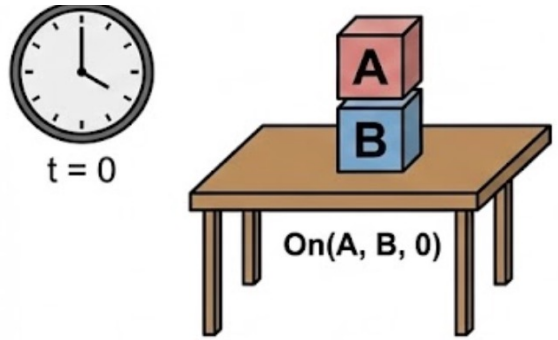
“在时刻t，积木b是否在对象o上?”

On(b, o, t)

b ∈ Blocks (e.g., A, B, C)

O ∈ Blocks ∪ {Table}

t ∈ {0, …， T}

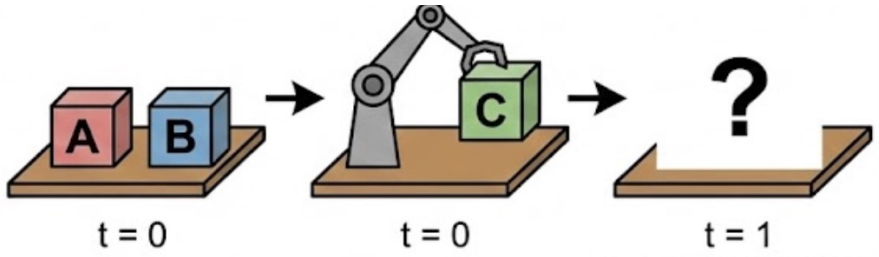


动作谓词：Move(b, src, dest, t)

“从时刻t到t+1，是否将积木b从src移动到dest”

Move(b, src, dest, t)

t ∈ {0, …， T-1}

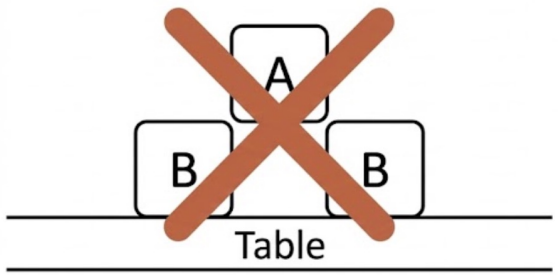


构建语法：状态一致性约束

物理世界的常识

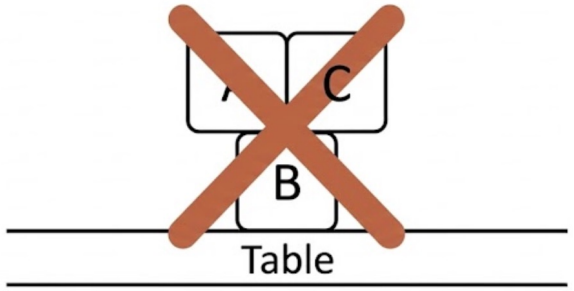
Rule1:位置唯一性

一个积木不能分身，任何时刻，它都必须且只能在一个地方



Rule2：独占性

一个积木的上端最多只能有一个积木



对应的逻辑约束

Rule1:位置唯一性

对应的任意积木b和时刻t，On(b, o, t)为真的o只有一个

$$\forall b,t:\sum_{o} On(b,o,t)=1$$

Rule2：独占性

对于任意积木b和时刻t，最多只有一个积木在它上面

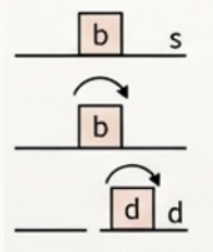
$$\forall b,t:\sum_{b\prime} On(b',b,t)\leq 1$$

构建语法：动作的因果链

一个动作Move(b, s, d, t)不是凭空发生的，它的执行必须有前提，也必然会产生后果。

前置条件(Preconditions)

我们要执行Move(b, s, d, t)，那么在时刻t必须满足：



On(b, s, t)：b必须真的在s上

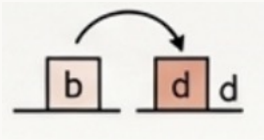
Clear(b, t)：b的顶部必须是空的

Clear(d, t)：目标位置的顶部也必须时空的

Move(b, s, d, t) -> On(b, s, t) ∧ Clear(b, t) ∧ Clear(d, t)

效果(Effects)

如果Move(b, s, d, t)成功执行，那么在下一个时刻t+1:



On(b, d, t+1)：b将会出现在d上

Move(b, s, d, t) -> On(b, d, t+1)

每个时刻只能执行一个动作：$\forall t: \sum Move(b, s, d, t) = 1$

还缺什么？

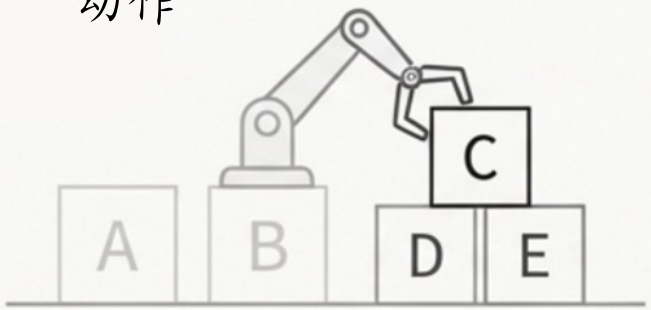
我们告诉了求解器，一个动作会导致什么

但是还没有告诉它，一个动作不会导致什么

时刻t=0

On(A, Table, 0)
On(B, Table, 0)

动作



Move(C, D, E, 0)
执行一个完全无关的动作

时刻t=1

在t=1时，A和B在哪里？根据我们目前的规则，它们可以在任何地方，甚至凭空消失！因为没有任何规则禁止它们这么做

缺少了物理世界最重要的定律之一：惯性定律，如果一个物体没有被外力改变，它的状态应保持不变

框架公理

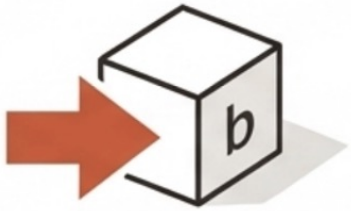
明确告知求解器：状态保持不变，除非被一个动作明确地改变

思考一个问题：对于一个积木b和位置loc，On(b, loc, t+1)为真的充要条件是什么？

只有两种可能性：

保持(Staying Put)

积木b在t时刻已经在loc上，
并且在t到t+1之间没有被移动



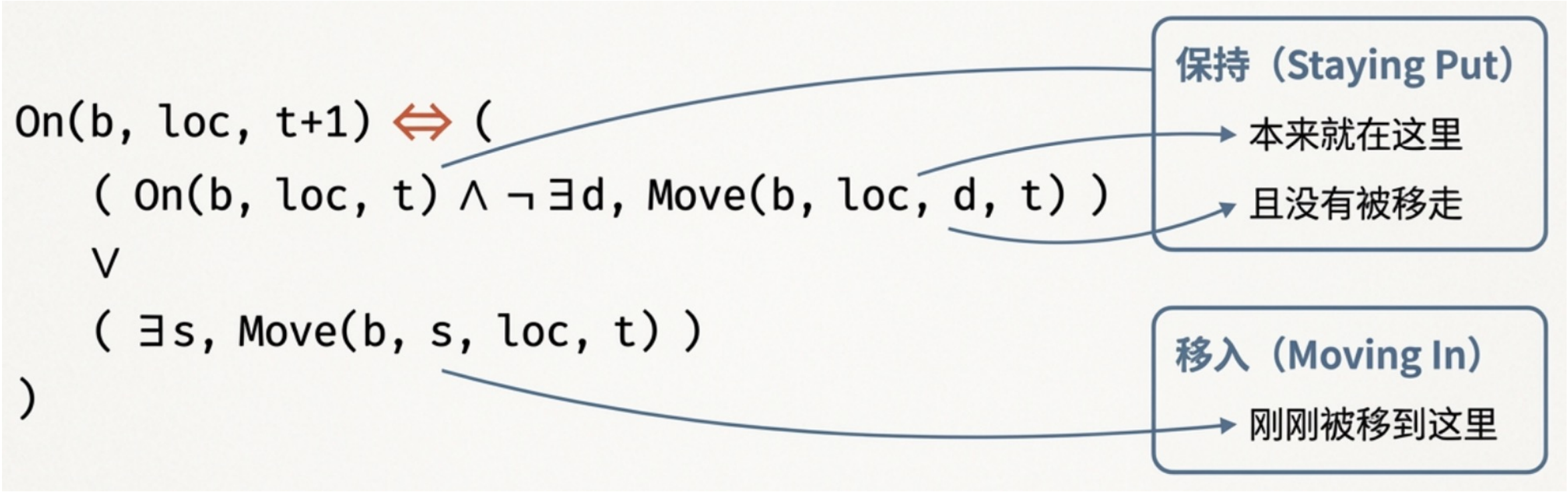
移入(Moving In)

积木b在t+1时刻被移动到loc上

通过定义状态改变的所有可能，我们就隐式的排除了其他不可能的原因(比如凭空消失)

解释性框架公理

最终的语法规则：解释性框架公理



双向蕴含像一把逻辑锁，把t+1时刻的状态与t时刻的状态完全绑定，不允许任何“魔法”发生

从逻辑到代码：Z3的实现

我们的逻辑手稿如何变成可执行的代码？

约束1：位置唯一性

$$\forall b, t: \sum_{o} On(b, o, t) = 1$$

约束2：独占性

$$\forall b, t: \sum_{b'} On(b', b, t) \leq 1$$

约束3：动作前置条件

Move(…)->On(…) ∧ Clear(…)

Python Z3 代码片段

```python
# --- 2. 状态一致性约束 (State Consistency) ---
for t in range(steps + 1):
    for b in blocks:
        # 物理约束1: 每个积木必须且只能在一个地方
        self.solver.add(PbEq([(self.on[(b, o, t)], 1) for o in objects if b != o], 1))

    for b in blocks:
        # 物理约束2: 一个积木上面最多只能有一个积木 (除非它是桌子)
        # 即: 对于任意积木 b, sum(on(other, b)) <= 1
        others_on_b = [self.on[(other, b, t)] for other in blocks if other != b]
        self.solver.add(AtMost(*others_on_b, 1))
```

```python
# --- 3. 动作约束 (Action Constraints) ---
for t in range(steps):
    # 互斥约束: 每个时间步只能执行一个移动操作
    all_moves = []
    for b in blocks:
        for src in objects:
            for dest in objects:
                if (b, src, dest, t) in self.move:
                    all_moves.append(self.move[(b, src, dest, t)])

    # 必须恰好有一个动作发生 (或者我们可以定义 No-Op, 这里简化为每步必动)
    self.solver.add(PbEq([(m, 1) for m in all_moves], 1))
```

从逻辑到代码：Z3的实现

我们的逻辑手稿如何变成可执行的代码？

Python Z3 代码片段

约束4：动作的因果链

- **前置条件(Preconditions)**
- **效果(Effects)**

Move(…)->On(…) ∧ Clear(…)

```python
# 遍历所有可能的移动操作
for b in blocks:
    for src in objects:
        for dest in objects:
            if (b, src, dest, t) not in self.move: continue

            action = self.move[(b, src, dest, t)]

            # --- Preconditions (前置条件) ---
            # 1. b 必须在 src 上
            # 2. b 必须是 clear 的 (没有东西在 b 上)
            # 3. dest 必须是 clear 的 (除非 dest 是 Table)

            pre_on_src = self.on[(b, src, t)]

            pre_b_clear = And([Not(self.on[(other, b, t)]) for other in blocks if other != b])

            if dest == 'Table':
                pre_dest_clear = True # 桌子永远是空的
            else:
                pre_dest_clear = And([Not(self.on[(other, dest, t)]) for other in blocks if other != dest])

            self.solver.add(Implies(action, And(pre_on_src, pre_b_clear, pre_dest_clear)))

            # --- Effects (效果) ---
            # 在 t+1 时刻, b 必须在 dest 上
            self.solver.add(Implies(action, self.on[(b, dest, t+1)]))
```

从逻辑到代码：Z3的实现

我们的逻辑手稿如何变成可执行的代码？

约束5：框架公理

On(b, loc, t+1)<−>Or(Staying, Moving)

Python Z3 代码片段

```python
for t in range(steps):
    for b in blocks:
        for loc in objects:
            if b == loc: continue

            # 情况A: t时刻就在这里，并且 t时刻没有执行任何把 b 从 loc 移走的操作
            was_here = self.on[(b, loc, t)]
            moved_away = False
            # 查找所有以 b 为操作对象，以 loc 为源头的动作
            moves_from_loc = []
            for dest in objects:
                 if (b, loc, dest, t) in self.move:
                     moves_from_loc.append(self.move[(b, loc, dest, t)])
            if moves_from_loc:
                moved_away = Or(moves_from_loc)
            else:
                moved_away = False # 不可能移走

            keep_staying = And(was_here, Not(moved_away))

            # 情况B: 刚刚执行了把 b 移到 loc 的操作
            moved_here = False
            moves_to_loc = []
            for src in objects:
                if (b, src, loc, t) in self.move:
                    moves_to_loc.append(self.move[(b, src, loc, t)])
            if moves_to_loc:
                moved_here = Or(moves_to_loc)
            else:
                moved_here = False

            # 综合: t+1 在 loc 的充要条件
            self.solver.add(self.on[(b, loc, t+1)] == Or(keep_staying, moved_here))
```

从逻辑到代码：Z3的实现

```python
# --- 测试案例 ---
if __name__ == "__main__":
    # 定义积木
    blocks = ['A', 'B', 'C']

    # 假设我们需要 3 步（如果步数给少了会显示无解，这是 SAT 规划的特点）
    # 初始: A在B上, B在C上, C在Table (一摞)
    # 目标: C在B上, B在A上, A在Table (倒过来的一摞)
    # 最优解通常需要 4 步: A->Tab, B->Tab, B->A, C->B (如果不允许直接移动堆叠好的，具体看约束)
    # 这里的解法可能是: A->Table, B->Table, B->A (Wait, B needs to go to A?), C->B

    planner = BlocksWorldSMT(blocks, steps=4)
    planner.add_constraints()

    # 初始状态: A on B, B on C, C on Table
    planner.set_initial({'A': 'B', 'B': 'C', 'C': 'Table'})

    # 目标状态: C on B, B on A, A on Table
    planner.set_goal({'C': 'B', 'B': 'A', 'A': 'Table'})

    planner.solve()
```

```
求解耗时: 0.0197s
找到 4 步的解决方案!
Step 1: Move A from B to Table
Step 2: Move B from C to Table
Step 3: Move B from Table to A
Step 4: Move C from Table to B
```

现代规划器

- 在线规划器： https://editor.planning.domains/#
- FastDownward：https://www.fast-downward.org/

阅读材料

Everything You Always Wanted to Know About *Planning* (But Were Afraid to Ask)

Jörg Hoffmann

Saarland University
Saarbrücken, Germany
hoffmann@cs.uni-saarland.de

Abstract. Domain-independent planning is one of the long-standing sub-areas of Artificial Intelligence (AI), aiming at approaching human problem-solving flexibility. The area has long had an affinity towards playful illustrative examples, imprinting it on the mind of many a student as an area concerned with the rearrangement of blocks, and with the order in which to put on socks and shoes (not to mention the disposal of bombs in toilets). Working on the assumption that this "student" is you – the readers in earlier stages of their careers – I herein aim to answer three questions that you surely desired to ask back then already: *What is it good for? Does it work? Is it interesting to do research in?* Answering the latter two questions in the affirmative (of course!), I outline some of the major developments of the last decade, revolutionizing the ability of planning to scale up, and the understanding of the enabling technology. Answering the first question, I point out that modern planning proves to be quite useful for solving practical problems - including, perhaps, yours.

https://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf

国际规划竞赛IPC(International Planning Competition)

- International Planning Competition，隶属ICAPS (International Conference on Automated Planning and Scheduling)会议
- 始于1998年，通常每2-3年举办一次
- 统一采用PDDL语言 作为领域标准
- 里程碑规划器：
  - **FF (Fast Forward) (IPC-2000)**：引入了忽略删除列表的启发式规划，现代规划器的鼻祖
  - **Fast Downward (IPC-2004)**：引入了 SAS+ 翻译和多启发式搜索架构，至今仍是学术界的基础框架
- 新趋势：
  - **HTN 复兴：分层规划重新引起重视**
  - **RL + Planning：传统搜索仍然强势，但RL正在冲击榜单**
  - **LLM+Planning：开始尝试基于大模型做规划任务**

https://ipc2023.github.io/

The International Planning Competition 2023

- Classical Tracks
  - Daniel Fišer, Saarland University
  - Florian Pommerening, University of Basel
- Learning Tracks
  - Jendrik Seipp, Linköping University
  - Javier Segovia-Aguas, Universitat Pompeu Fabra
- Probabilistic Tracks
  - Ayal Taitler, University of Toronto
  - Scott Sanner, University of Toronto
- Numeric Tracks
  - Joan Espasa Arxer, University of St Andrews
  - Enrico Scala, University of Brescia
- HTN Tracks
  - Ron Alford, MITRE
  - Dominik Schreiber, Karlsruhe Institute of Technology
  - Gregor Behnke, University of Amsterdam

LLM for Planning

Published in Transactions on Machine Learning Research (April/2025)

A Systematic Evaluation of the Planning and Scheduling Abilities of the Reasoning Model o1

Domain	Shots	Claude Models		OpenAI GPT-4 Models				LLaMA Models		Gemini Models	
		3.5 Sonnet	**3 Opus**	**4o**	**4o mini**	**4**	**4 Turbo**	**3.1 405B**	**3 70B**	**1.5 Pro**	**1 Pro**
Blocks world	One Shot	**346/600 (57.6%)**	289/600 (48.1%)	170/600 (28.3%)	49/600 (8.1%)	206/600 (34.3%)	138/600 (23%)	284/600 (47.3%)	76/600 (12.6%)	101/600 (16.8%)	68/600 (11.3%)
	Zero Shot	329/600 (54.8%)	356/600 (59.3%)	213/600 (35.5%)	53/600 (8.8%)	210/600 (34.6%)	241/600 (40.1%)	**376/600 (62.6%)**	205/600 (34.16%)	143/600 (23.8%)	3/600 (0.5%)
Mystery Blocks world	One Shot	19/600 (3.1%)	8/600 (1.3%)	5/600 (0.83%)	0/600 (0%)	**26/600 (4.3%)**	5/600 (0.83%)	21/600 (3.5%)	15/600 (2.5%)	-	2/500 (0.4%)
	Zero Shot	0/600 (0%)	0/600 (0%)	0/600 (0%)	0/600 (0%)	1/600 (0.16%)	1/600 (0.16%)	**5/600 (0.8%)**	0/600 (0%)	-	0/500 (0%)

https://openreview.net/pdf?id=FkKBxp0FhR

LLM for Planning

PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change

Karthik Valmeekam
School of Computing & AI
Arizona State University, Tempe.
kvalmeek@asu.edu

Matthew Marquez
School of Computing & AI
Arizona State University, Tempe.
mmarqu22@asu.edu

Alberto Olmo
School of Computing & AI
Arizona State University, Tempe.
aolmoher@asu.edu

Sarath Sreedharan*
Department of Computer Science,
Colorado State University, Fort Collins.
sarath.sreedharan@colostate.edu

Subbarao Kambhampati
School of Computing & AI
Arizona State University, Tempe.
rao@asu.edu

https://arxiv.org/pdf/2206.10498

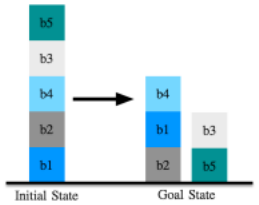
Task	Instances correct	
	GPT-4	I-GPT3
Plan Generation We showcase an instance and the respective plan as an example and prompt the machine with a new instance.	206/600 (34.3%)	41/600 (6.8%)
Cost-Optimal Planning We showcase an instance, the respective optimal plan and the associated cost as an example and prompt the machine with a new instance.	198/600 (33%)	35/600 (5.8%)
Plan Verification We showcase three instances and three distinct plans (goal reaching, non goal-reaching and inexecutable) and present the respective validation and explanations. We then present a new instance and a plan and ask the machine for to verify and provide an explanation, if needed.	352/600 (58.6%)	72/600 (12%)
Reasoning About Plan Execution We showcase an instance, an action sequence and the corresponding resulting state after executing the action sequence as an example. We then provide an instance and an executable action sequence and ask the machine to provide the resulting state.	191/600 (31.8%)	4/600 (0.6%)
Replanning We showcase an instance, the respective plan and present an unexpected change of the state. We then also present a new plan from the changed state. Finally, for a new instance we repeat the same except we ask the machine for the new plan.	289/600 (48.1%)	40/600 (6.6%)
Plan Generalization We showcase an instance and the respective plan as an example and prompt the machine with a new instance. The plans for both the instances can be generated by a fixed program containing loops and conditionals.	141/500 (28.2%)	49/500 (9.8%)
Plan Reuse We showcase an instance and the respective plan as an example and prompt the machine with an instance which requires only a certain prefix of the plan provided in the example.	392/600 (65.3%)	102/600 (17%)
Robustness to Goal Reformulation (Shuffling goal predicates) We showcase an instance and the respective plan as an example and prompt the machine with the same instance but shuffle the ordering of the goals.	461/600 (76.8%)	467/600 (77.8%)
Robustness to Goal Reformulation (Full → Partial) We showcase an instance with a fully specified goal state and the respective plan as an example and prompt the machine with the same instance but provide a partially specified goal state.	522/600 (87%)	467/600 (77.8%)
Robustness to Goal Reformulation (Partial → Full) We showcase an instance with a partially specified goal state and the respective plan as an example and prompt the machine with the same instance but provide a fully specified goal state.	348/600 (58%)	363/600 (60.5%)

前沿趋势：LLM + PDDL

LLM充当自然语言到形式化语言的翻译器，调用符号求解工具完成规划任务

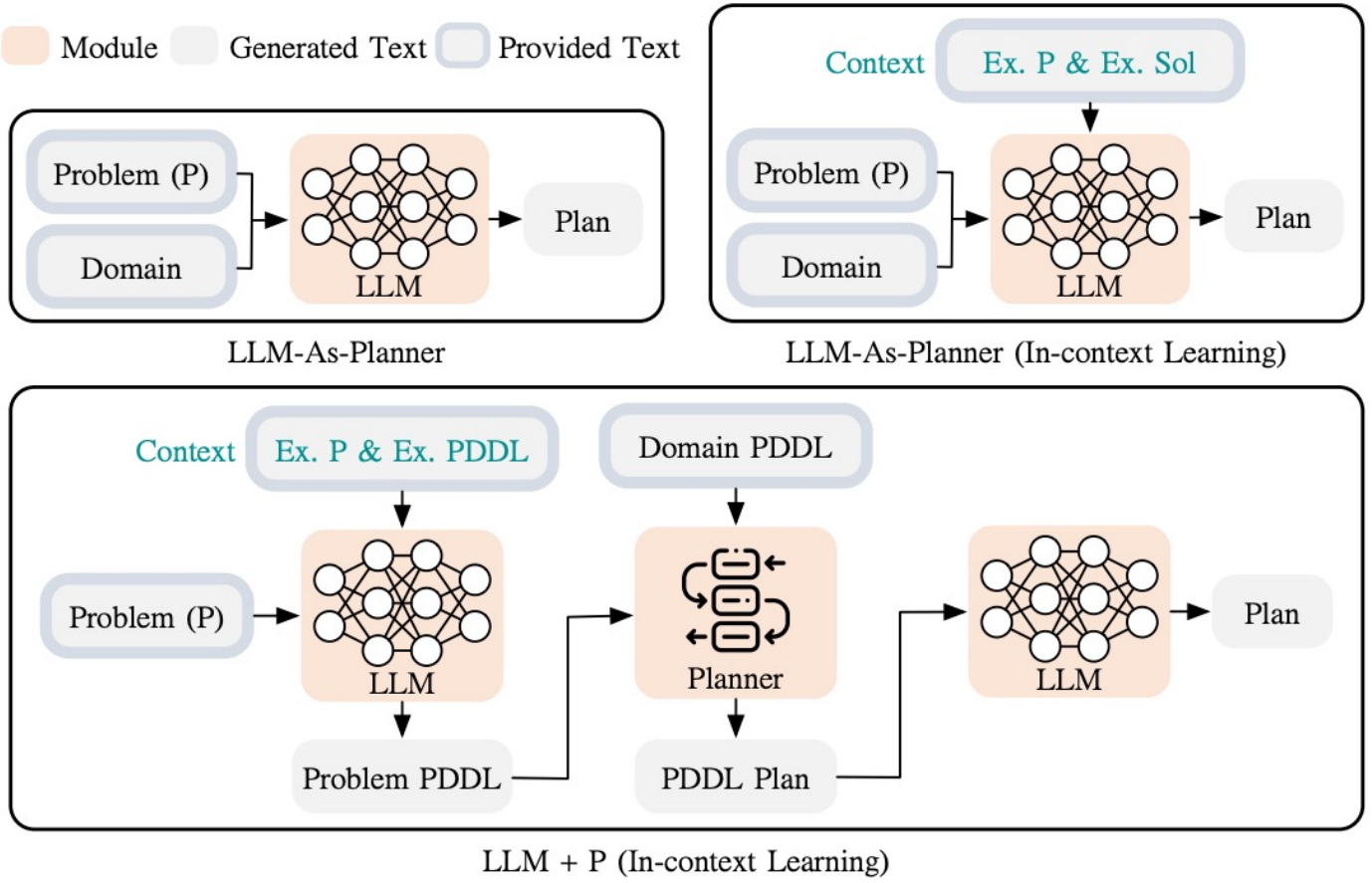
A Failure Example of GPT-4 in Planning

Problem (P1): You have 5 blocks. One cannot place more than one block on another block. b5 is on top of b3. b4 is on top of b2. b2 is on top of b1. b3 is on top of b4. b1 is on the table. b5 is clear. Your arm is empty.
Your goal is to move the blocks.
b1 should be on top of b2.
b3 should be on top of b5.
b4 should be on top of b1.



GPT-4:

```
1. Move b5 from b3 to the table.
2. Move b3 from b4 to b5.
3. Move b4 from b2 to the table.
4. Move b2 from b1 to b5.
5. Move b1 from the table to b2.
6. Move b4 from the table to b1.
```



LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. https://arxiv.org/pdf/2304.11477

前沿趋势：LLM + PDDL

LLM充当自然语言到形式化语言的翻译器，调用符号求解工具完成规划任务

(a) grasp bottle (b) free gripper (c) grasp soup can (d) place soup can (e) re-grasp bottle (f) place bottle

Fig. 2: Demonstration of the optimal tidy-up plan. The robot starts at the coffee table and 1) picks up the bottle, 2) navigates to a room with the side table and the recycle bin, 3) puts down the bottle, 4) grasps the soup can, 5) puts the soup can in the recycle bin, 6) re-grasps the bottle, 7) navigates to the kitchen, 8) places the bottle in the pantry.

LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. https://arxiv.org/pdf/2304.11477

符号主义人工智能的痛点：符号接地

Symbolic AI 的核心假设是：**智能是对符号的操作**

- **离散性与确定性**：On(Block A, Block B) 是一个非常明确、没有歧义的状态
- **组合性**：符号可以无限组合。学会了“拿”，学会了“苹果”，就能理解“拿苹果”，不需要重新训练
- **长程推理能力**：在数学证明、规划、逻辑谜题中，符号系统可以进行几十甚至上百步的推理
- **经典AI的困境**：对于一个纯符号系统，它知道 Apple 是一种 fruit，fruit 可以 eat。但如果你给它看一张苹果的照片，或者让机器人去抓一个苹果，它完全不知道这个符号对应现实中的哪一个物体
- **鸿沟**：**现实世界是连续的、嘈杂的（像素、声波、电压），而符号世界是离散的、抽象的，Symbolic AI缺乏一个将“连续信号”映射为“离散符号”的转换器**

THE SYMBOL GROUNDING PROBLEM

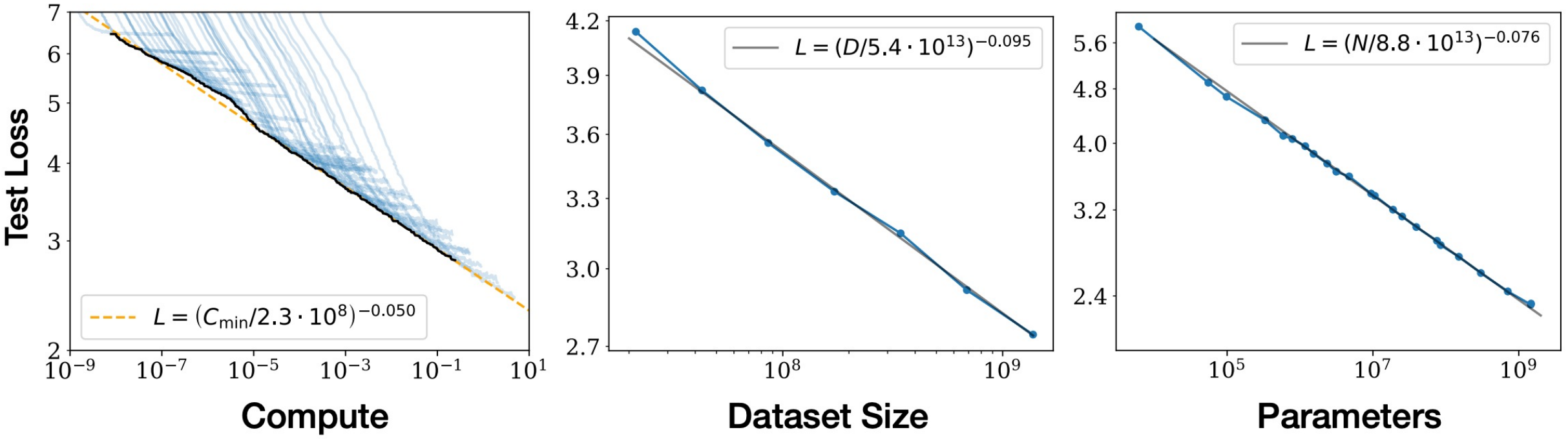
Stevan Harnad
Department of Psychology
Princeton University
Princeton NJ 08544
harnad@cogsci.soton.ac.uk

ABSTRACT: There has been much discussion recently about the scope and limits of purely symbolic models of the mind and about the proper role of connectionism in cognitive modeling. This paper describes the "symbol grounding problem": How can the semantic interpretation of a formal symbol system be made intrinsic to the system, rather than just parasitic on the meanings in our heads? How can the meanings of the meaningless symbol tokens, manipulated solely on the basis of their (arbitrary) shapes, be grounded in anything but other meaningless symbols? The problem is analogous to trying to learn Chinese from a Chinese/Chinese dictionary alone. A candidate solution is sketched: Symbolic representations must be grounded bottom-up in nonsymbolic representations of two kinds: (1) "iconic representations" , which are analogs of the proximal sensory projections of distal objects and events, and (2) "categorical representations" , which are learned and innate feature-detectors that pick out the invariant features of object and event categories from their sensory projections. Elementary symbols are the names of these object and event categories, assigned on the basis of their (nonsymbolic) categorical representations. Higher-order (3) "symbolic representations" , grounded in these elementary symbols, consist of symbol strings describing category membership relations (e.g., "An X is a Y that is Z"). Connectionism is one natural candidate for the mechanism that learns the invariant features underlying categorical representations, thereby connecting names to the proximal projections of the distal objects they stand for. In this way connectionism can be seen as a complementary component in a hybrid nonsymbolic/symbolic model of the mind, rather than a rival to purely symbolic modeling. Such a hybrid model would not have an autonomous symbolic "module," however; the symbolic functions would emerge as an intrinsically "dedicated" symbol system as a consequence of the bottom-up grounding of categories' names in their sensory representations. Symbol manipulation would be governed not just by the arbitrary shapes of the symbol tokens, but by the nonarbitrary shapes of the icons and category invariants in which they are grounded.

https://arxiv.org/html/cs/9906002

观点1：仅依赖大模型

大语言模型、多模态大模型已经展现初步的推理能力，尽管还存在幻觉、不可靠等问题，但只要能够持续Scaling，就能实现性能提升

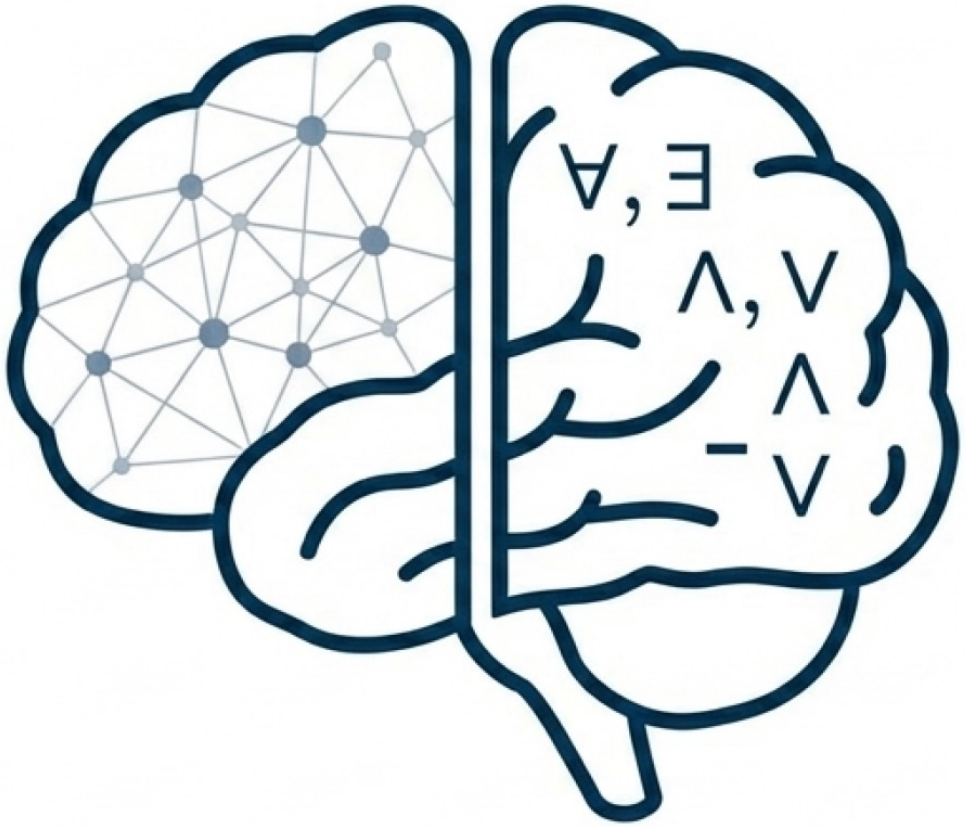


More data, More compute, Larger models

观点2：Neuro-Symbolic AI

Neural Networks

神经网络的感知、
识别能力
(System 1)



Symbolic

符号推理的严谨
逻辑、可靠性、
可解释性

Neuro-Symbolic AI: 结合两种AI范式的优势，创造即聪明又可靠的下一代人工智能系统

迈向下一站