提醒

- 作业2 已发布
- http://qiqi789.github.io/teaching/AI/

更新通知

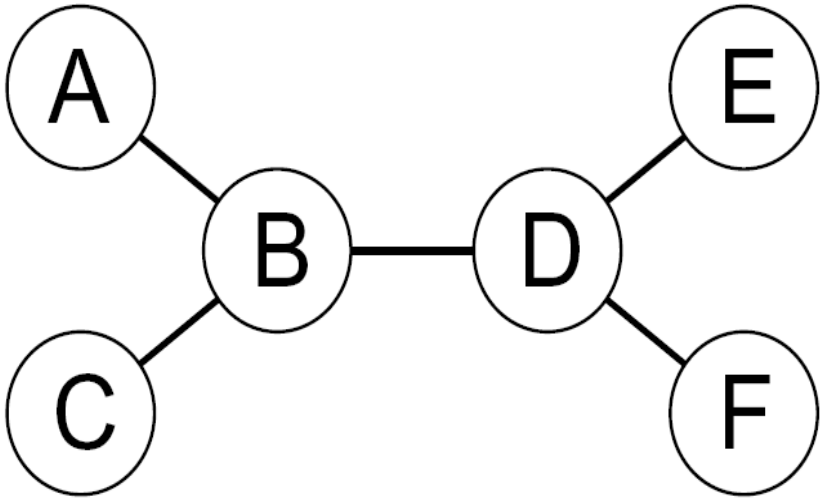
2015/10/24

> 作业2 已贴出；11月4日课上提交。

今天的内容

- 约束满足问题 剩余部分
- 命题逻辑
  - 语法，语义和推理
- 逻辑型的智能体

树结构的 CSPs

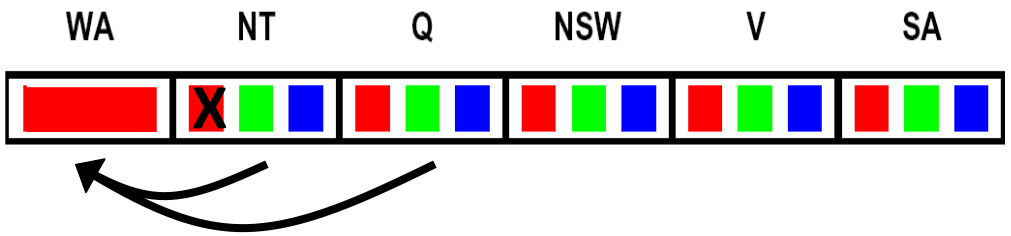


定理: 如果约束图无环(树结构的)，则其对应的约束满足问题的求解时间复杂度是 $O(n\ d^2)$

- 比较一般的 CSPs, 最差时间复杂度是 $O(d^n)$

弧的一致性（Arc consistency）

一个弧 X → Y 是 *一致的* 当且仅当 对于X中的 *每一个* x 值，Y中存在 *某个* y 值 不违背任何一个约束条件



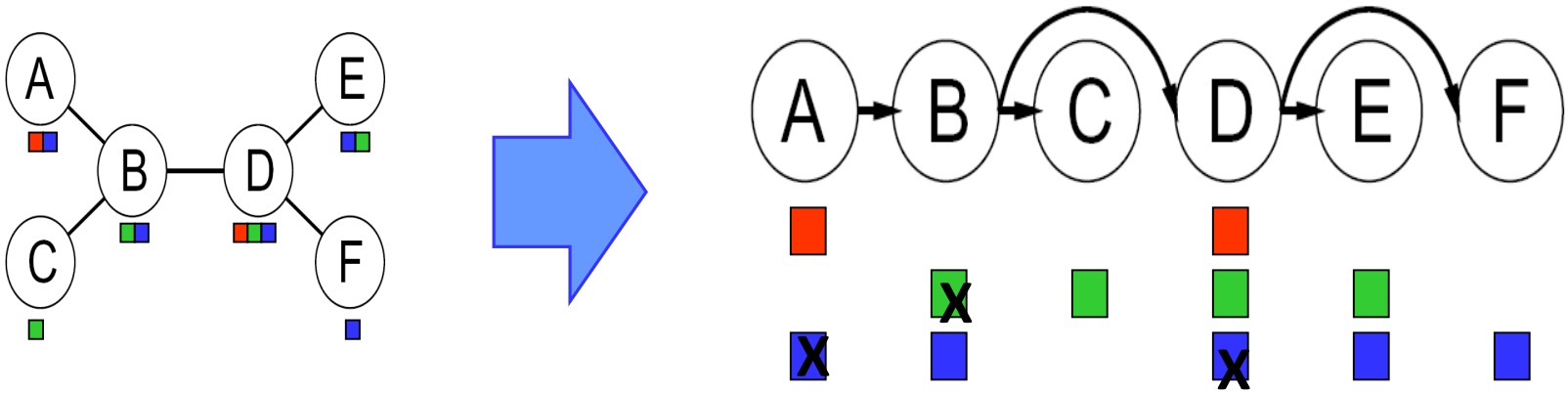
从弧的尾部删除

前向检查（Forward checking）：强制检查剩余未赋值变量指向新赋值变量的弧的一致性

树结构的 CSPs

树结构的CSPs的求解算法：

- 排序: 选一个根节点, 把变量线性排序，使得父节点排在子节点之前



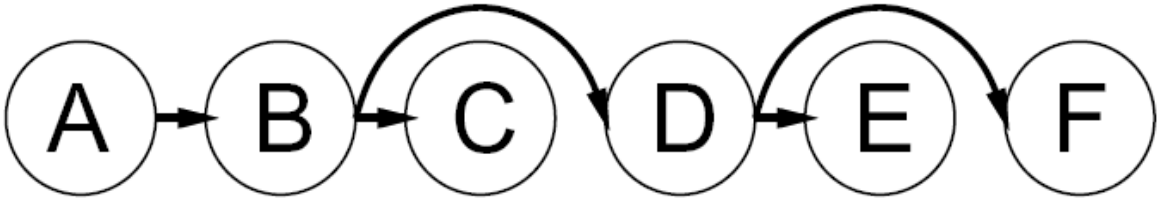
- **从后向前删除**: For i = n : 2, 执行： 删除不一致的值(父节点(X_i),X_i)
- 从前向后赋值: For i = 1 : n, 赋值 X_i 和父节点 Parent(X_i) 相一致的值

运行时间: $O(n\ d^2)$

树结构的 CSPs

声明 1: **在从后向前**的一致性检查后, 所有根到叶的弧都是一致性的

证明: 每个 X->Y 如果是一致性的，那么 Y的值域以后不会被减小 (因为 Y 的 子节点 在Y 之前先被处理过)



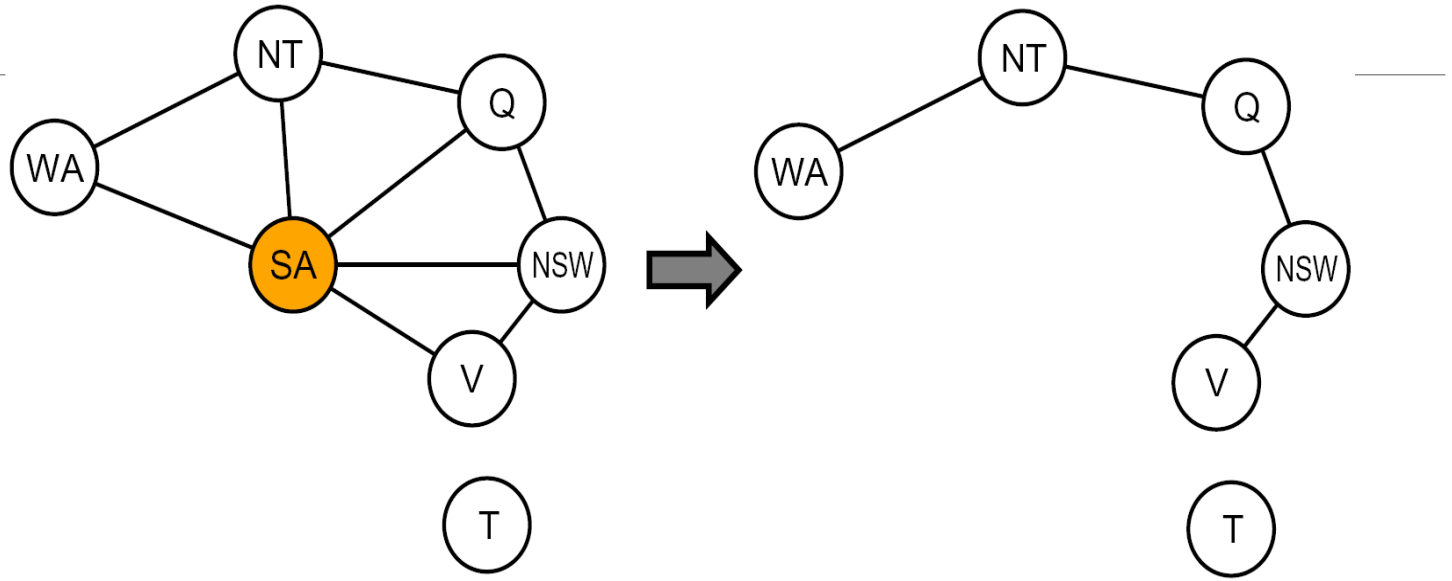
声明 2: **如果从根到叶的弧都是一致性的**, 那么从前向后的赋值过程将不会有回溯

证明: 归纳法

为什么这个算法不适用于约束图中有环的情况?

改进结构

几乎是树结构的 CSPs



条件制约: 赋值一个变量, 剪裁这个变量的邻居变量的值域

切集条件制约: 对切集变量赋值 (所有可能的组合)，使得 剩下的约束图变成一个树

切集大小是c，时间复杂度为 $O((d^c) (n-c) d^2)$，c 很小时算法很快

例如, 80 个变量, c=10, 40亿 years -> 0.029 秒

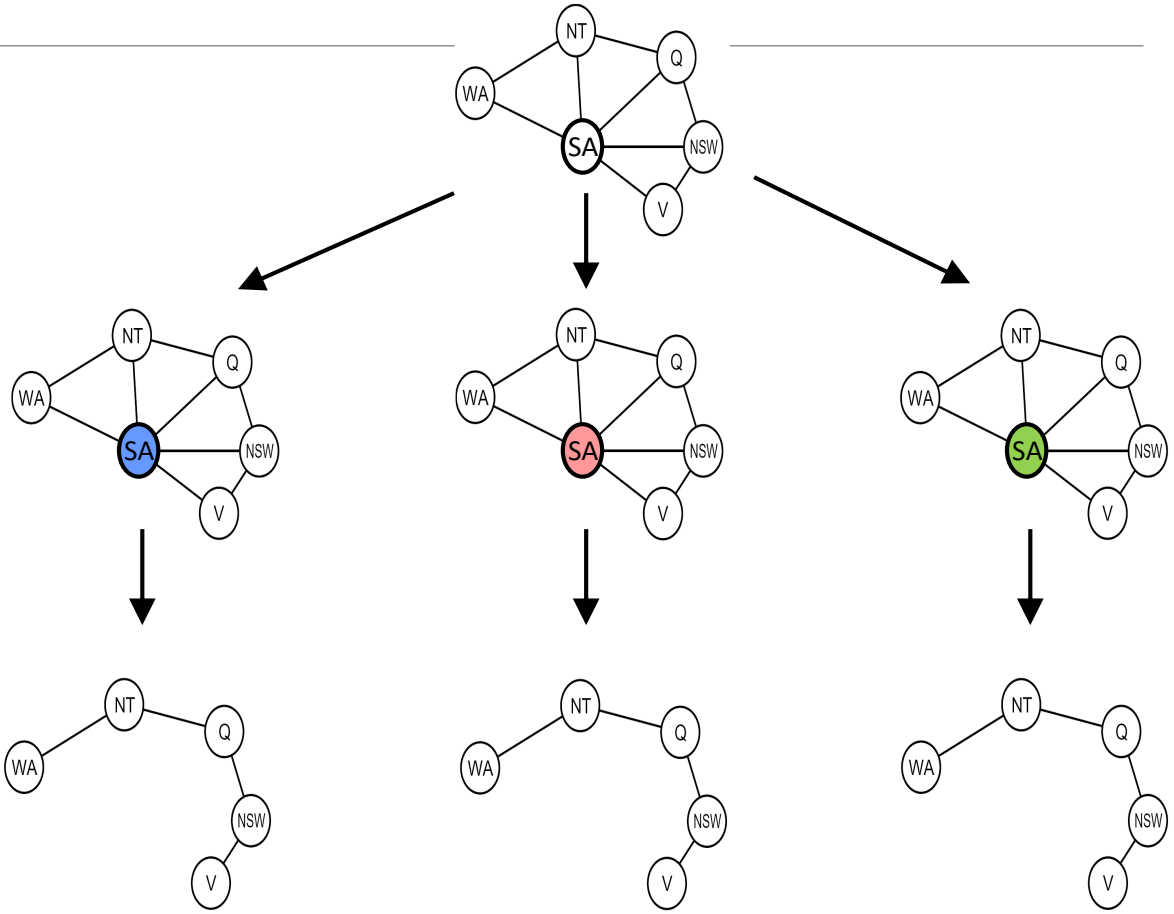
切集条件化制约

选择一个切集

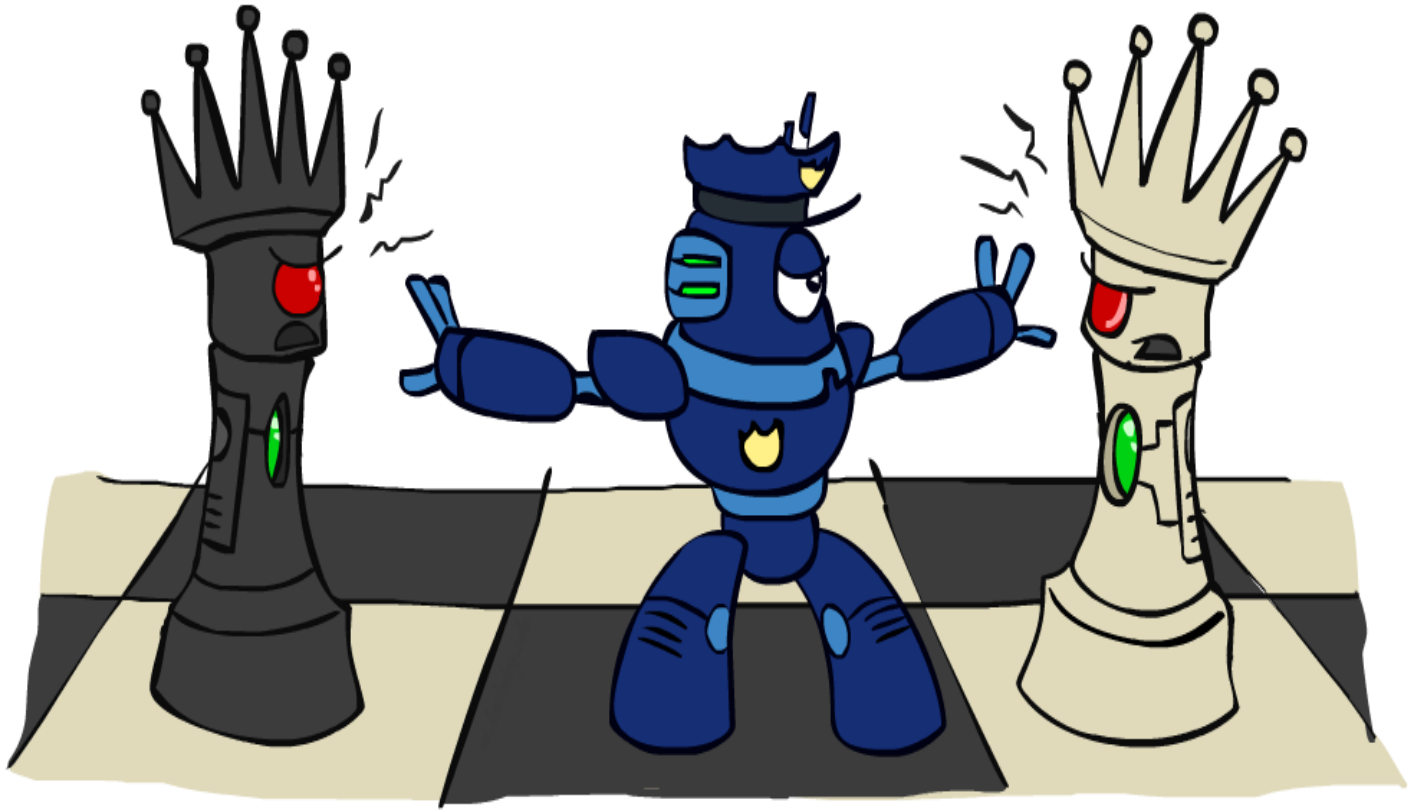
对切集变量赋值(所有可能组合)

计算剩余的CSP相对于每一组切集赋值

求解剩余的 CSPs (树结构的)



局部方法求解 CSPs



最小冲突算法

像爬山算法, 但不完全是!

算法: 开始时所有状态都已随机赋值，迭代改进，直至问题得到求解,

- 变量选择: ***随机选择*** 任何冲突的变量
- 值选择: 最小冲突启发信息:
  - 选择一个和约束条件冲突最少的值
  - 随机打破平局情况

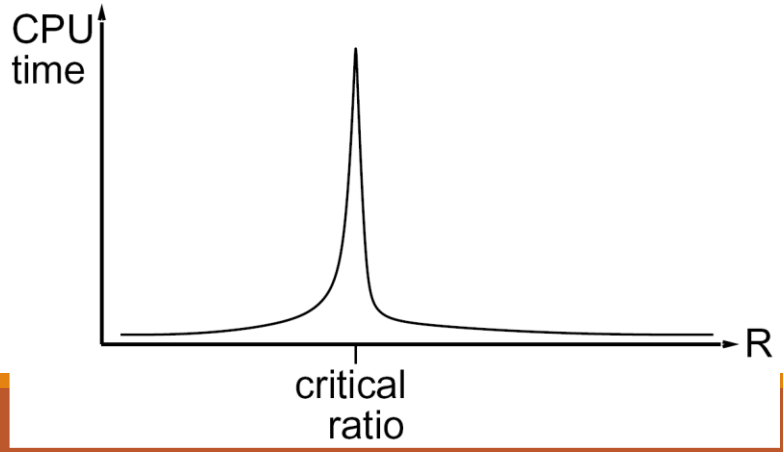
图着色演示: 最小冲突法

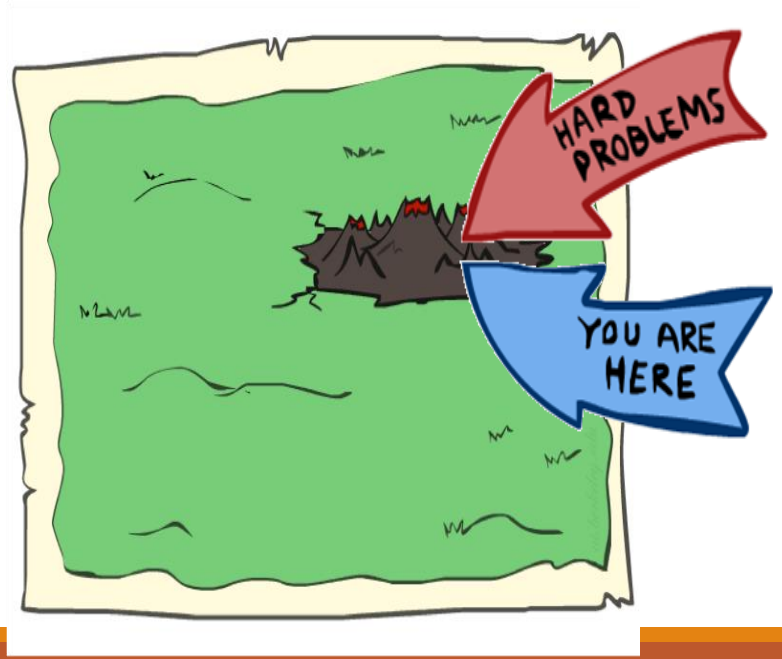
最小冲突算法的表现性能

给定随机初始化状态下, 几乎可以在常量时间里以高成功概率求解 n 为任意数的 n-皇后问题， (例如, n = 10,000,000)!

同样的表现性能对任意随机产生的 CSP都适用，除了在一个很窄的比例范围内

$$R = \frac{\text{约束条件数}}{\text{变量数}}$$



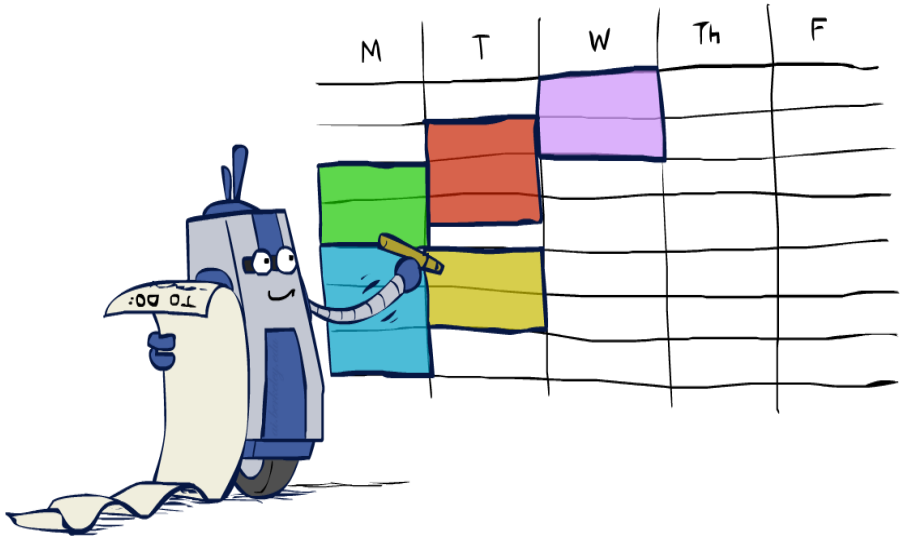


总结：约束满足问题

约束满足问题是一类特殊的搜索问题:

- 状态是变量的 (部分) 配值
- 目标检测通过检查约束条件
- 通用的算法和启发信息

基本算法: 回溯搜索



提速思路:

- 排序
- 过滤
- 结构

实践中，最小冲突法的局部算法经常很有效

命题逻辑(PROPOSITIONAL LOGIC)：语法，语义和推理

知识(knowledge)

知识库（knowledge base）= 在形式语言中定义的一个句子的集合

声明式（declarative）法构建智能体(或其他系统):

- 告诉 智能体它需要知道的(或让它自己***学习***这些知识)
- 然后它可以询问 自己在一个环境里如何行动———回答应遵循知识库里的知识

在***知识层(knowledge level)***描述智能体

- 具体说明智能体 ***知道 什么***, 它的目标是什么，和实现细节无关

一个推理算法可以回答任何可以回答的问题

- 比较而言，一个搜索算法只能回答 “如何从A 到 B” 的问题

领域特定事实 (Domain-specific facts)

领域独立的通用算法和代码

逻辑（Logic）

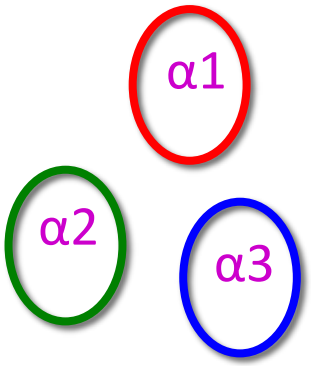
语法(Syntax): 定义句子(什么样的句子是允许的)

语义（Semantics）：

- *可能的世界 (possible worlds) 有哪些?*
- 这些句子在哪些世界里为 *真*？(句子真实性的*定义*)

语义空间

语法空间



举例

命题逻辑（Propositional logic）

- 语法: $P \vee (\neg Q \wedge R)$; $X_1 \Leftrightarrow (Raining \Rightarrow Sunny)$
- 可能的一个世界: {P=true,Q=true,R=false}, {X1=true, Raining = false, Sunny=true} , or {110}， {101}
- 语义: $\alpha \wedge \beta$ 是真 当且仅当 α 真 并且 β 真 (等)

一阶谓词逻辑 (First-order logic)

- 语法: $\forall x \exists y\ P(x,y) \wedge \neg Q(Joe,f(x)) \Rightarrow f(x)=f(y)$
- 可能的世界: 对象(Objects) o_1, o_2, o_3; P 成立对于 $<o_1,o_2>$; Q 成立对于 $<o_3>$; $f(o_1)=o_1$; $Joe=o_3$; 等
- 语法: $\phi(\sigma)$ 是真在这样一个世界里 当 $\sigma=o_j$ 并且 ϕ 成立对于 o_j; 等。

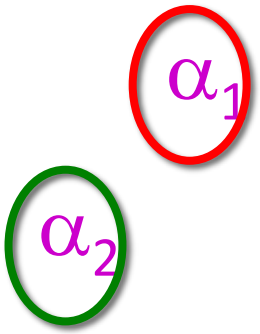
推理的内容: 蕴涵/导出(entailment)

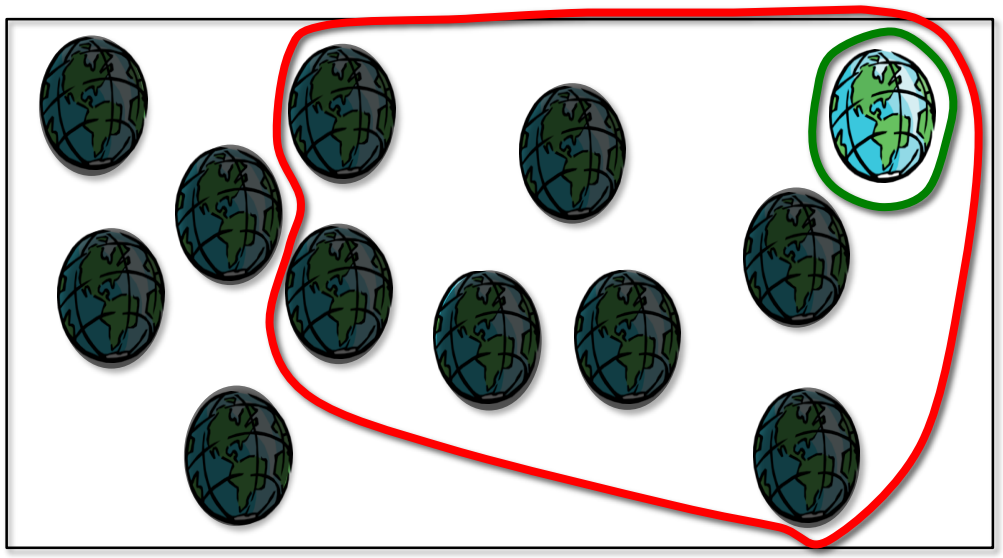
蕴涵: $\alpha \models \beta$ ("α 导出(entails) β" or "β 遵循于(follows from) α")当且仅当在 α 为真的每个世界里, β 也是真

- 换句话说, α-的世界（为真的那些世界） 是 β-的世界的一个子集 [***models***(α) $\subseteq$ ***models***(β)]

例如, $\alpha_2 \models \alpha_1$

(比如 α_2 是 $\neg Q \wedge R \wedge S \wedge W$
α_1 是 $\neg Q$)





推理的过程: 证明

证明--指 α 和 β 之间的导出（蕴涵）关系的***演示证明***

方法 1: ***模型检查 (model-checking)***

- 对于每一个可能的世界里, 如果 α 为真 ， 那么确认 β 也真
- 有限多的世界里可行（比如命题逻辑）； 但不容易对于一阶谓词逻辑

方法 2: ***定理证明 (theorem-proving)***

- 搜寻一系列的证明步骤 (应用 推理规则(***inference rules***)) 从 α 引导到 β
- 例如, 从 $P \wedge (P \Rightarrow Q)$, 推理出 Q 通过 肯定前件式推理(***Modus Ponens***)

合理性 (Sound) 算法: 所有被推理证明出来的，实际上也都是被蕴涵的

完全性 (Complete) 算法: 所有被蕴涵的（句子），都可以被推理证明出来

关于完全搜索算法和逻辑推理算法

完全的 逻辑推理算法和 完全的搜索算法 之间的关联是什么？

回答 1: 都含有 “完全的…算法”

回答 2: 都可用来求解任何相关的 可解决解答的 问题

回答 3: 推理可以建成一个搜索问题

- 初始状态: KB（知识库） 包含 α
- 行动: 应用任何可以和 KB 相匹配的推理规则, 并添加结论句子
- 目标检测: KB 包含 β

因此， 任何一个完全的 搜索算法 （广度优先，迭代加深）产生一个完全的推理算法

命题逻辑（Propositional logic）：语法

给定: 一组 命题字符 {X_1,X_2,…, X_n ,P, Q, R, North,…}（可为真或假）
- (True 和 False 也包含其中，真值固定)

X_i 是一个句子（原子语句）

复杂句
- 如果 α 是句子，那么 $\neg\alpha$ 是一个句子 （否定）
- 如果α 和 β是句子，那么$\alpha \wedge \beta$是一个句子 （结合/“与”）
- 如果α 和 β是句子，那么$\alpha \vee \beta$是一个句子 （分离/“或”）
- 如果α 和 β是句子，那么$\alpha \Rightarrow \beta$是一个句子 （隐含,或条件的if …then）
- 如果α 和 β是句子，那么$\alpha \Leftrightarrow \beta$是一个句子 （双向条件的 if and only if ）
- 逻辑连接符，和（）的组合

文字(literal): 原子语句和否定的原子语句

没有其他样式的句子!

命题逻辑: 语义

给定一个模型（model），决定一个句子的真值

真值表

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

命题逻辑(Propositional Logic): 语义

---

function PL-TRUE?(α,model) **returns** true or false

if α 是一个命题符号 **then return** Lookup(α, model)

if Op(α) = ¬ **then return** not(PL-TRUE?(Arg1(α),model))

if Op(α) = ∧ **then return** and(PL-TRUE?(Arg1(α),model),
PL-TRUE?(Arg2(α),model))

，等。

(也叫做 “语法上的递归”)

举例

- $R \Rightarrow (D \Leftrightarrow U)$ (如果天下雨, 当且仅当你有一把雨伞时，你才不会被淋湿)
- Model {D=true, R=false, U=true} ….
- false $\Rightarrow$ (true $\Leftrightarrow$ true)
- false $\Rightarrow$ true
- true

推理方法（回顾）

方法 1: 模型检查 ***model-checking***

- 对于每个可能的世界, 如果 α 为真 则 β 亦真

方法 2: 定理证明 ***theorem-proving***

- 搜索一系列的证明步骤 (应用推理规则 ***inference rules***) 从 α 导致到 β

合理的（Sound） 算法:所有被推理证明出来的，实际上也都是被蕴涵的

完全的（Complete） 算法: 所有被蕴涵的都是能够被推导证明出来的

逻辑上的一致性

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

简单的定理证明过程: 前向推理 (Forward chaining)

应用肯定前件式推理(Modus Ponens) 产生新的事实:

- 给定 $X_1 \wedge X_2 \wedge \ldots X_n \Rightarrow Y$ 和 $X_1, X_2, \ldots, X_n$
- 推理出 Y

前向推理持续应用这个规则, 不断添加新的事实, 直到没有可添加的为止

要求 KB（知识库） 只包含 ***确定子句(definite clauses)***:

- 一组分离(disjunction)的文字(literals),只有一个是正的（其他都含否定符）; 可转成以下形式
- (字符的结合(conjunction)) $\Rightarrow$ 字符; 或
- 一个单一的字符 (注意 X 相当于 True $\Rightarrow$ X)

前向链接算法(Forward chaining)

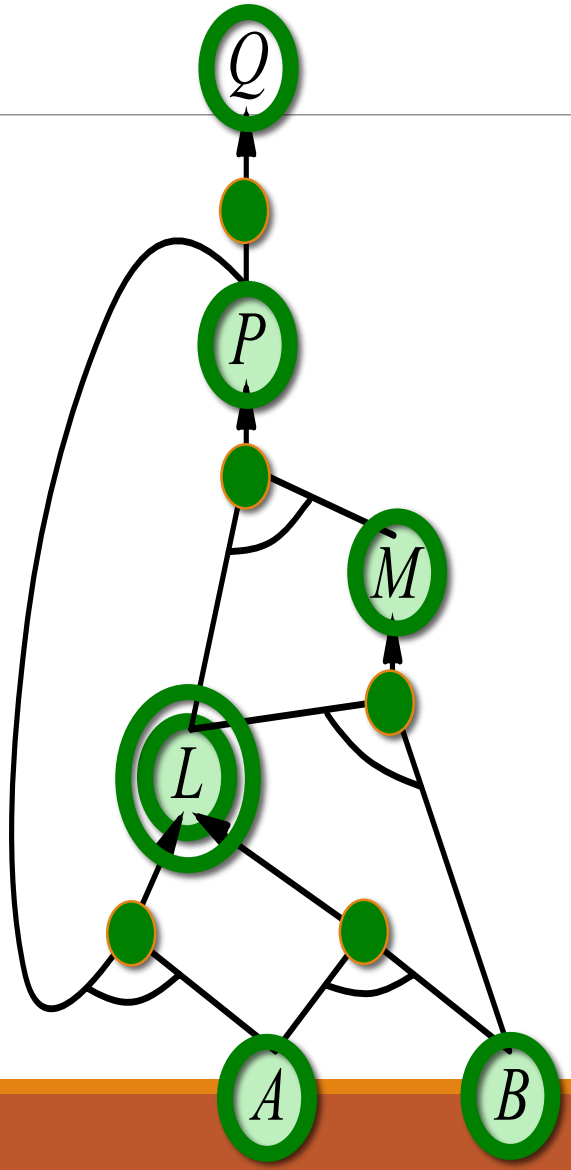
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  count ← 一个表，count[c] 是子句 c 的前提中的还未知的字符数量
  inferred ← 一个表，inferred[s] 初始化为 false 对于所有字符 s
  agenda ← 一个字符队列, 初始化为 KB 里 (为真的)所有字符
  while agenda is not empty do
    p ← Pop(agenda)
    if p = q then return true
    if inferred[p] = false then
        inferred[p]←true
        for each 子句 c in KB where p 在 c的前提里 do
            减一 count[c]
            if count[c] = 0 then add c的结论 to agenda
  return false
```

前向链接推理举例: 证明 Q（被蕴涵）

子句	COUNT	INFERRED
$P \Rightarrow Q$	1 / 0	A false true
$L \wedge M \Rightarrow P$	2 / 1 / 0	B false true
$B \wedge L \Rightarrow M$	2 / 1 / 0	L false true
$A \wedge P \Rightarrow L$	2 / 1 / 0	M false true
$A \wedge B \Rightarrow L$	2 / 1 / 0	P false true
A	0	Q false true
B	0	

AGENDA

A B L M P Q



前向链接(FC)推理的性质

定理: FC 是合理的(sound) 和 完全的(complete)， 对于确定子句(definite-clause)组成的KBs

合理性: 遵循于肯定前件式（ Modus Ponens）的合理性

完全性证明:

1. 假设FC 达到了一个固点，即 没有新的原子语句被推导出
2. 最终的 *inferred* 表可以被考虑成一个模型 ***m***, 即字符被赋给了 true/false 值
3. 在原始的 KB 中的每一个子句在 ***m*** 里为真

   证明: 假定一个子句 $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ 在 ***m*** 中为假

   那么 $a_1 \wedge \ldots \wedge a_k$ 必为真在 ***m*** 并且 b 为假 在 ***m***

   如此说明算法还未达到一个固点! （与假设矛盾）
4. 因此 ***m*** 是 KB 的一个模型 （KB 在 m 里为真）
5. 如果 KB |= q, q 则在KB中的每一个模型里为真, 包括 ***m；*** 即q可以被算法推导出来

A false true

B false true

L false true

M false true

P false true

Q false true

简单的模型检查(model checking)

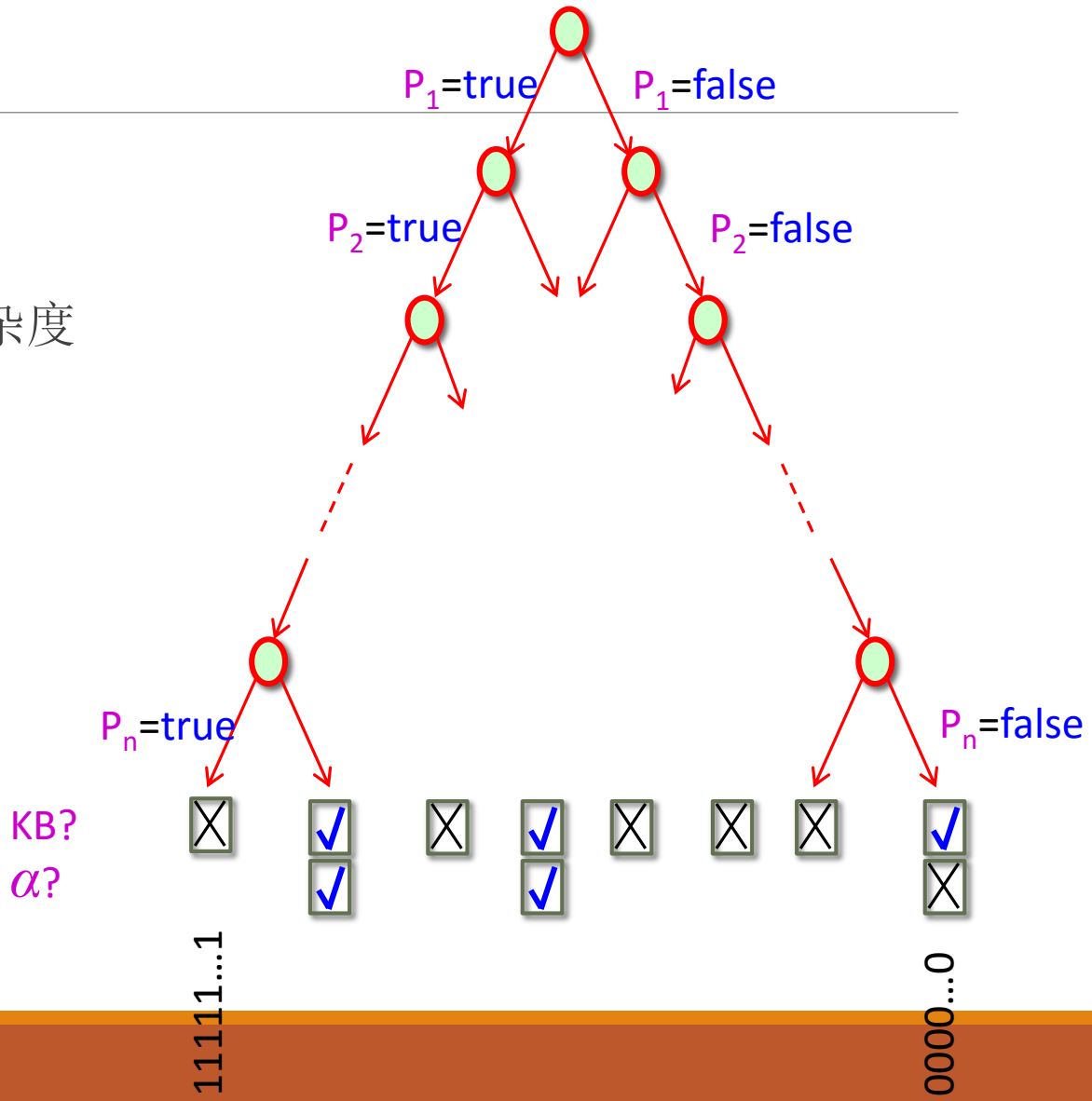
```
function TT-ENTAILS?(KB, α) returns true or false
  return TT-CHECK-ALL(KB,α,symbols(KB) ∪ symbols(α),{})
function TT-CHECK-ALL(KB,α,symbols,model) returns true or false
  if empty?(symbols) then
      if PL-TRUE?(KB,model) then return PL-TRUE?(α,model)
      else return true
  else
      P ← first(symbols)
      rest ← rest(symbols)
      return and (TT-CHECK-ALL(KB,α,rest,model ∪ {P = true})
                  TT-CHECK-ALL(KB,α,rest,model ∪ {P = false }))
```

简单的模型检查, 继续

深度优先，类似于回溯算法(backtracking)的递归

$O(2^n)$ 时间复杂度, 线性空间复杂度

可以有更高效的算法!



可满足性 和 导出(蕴涵)

一个语句是 ***可满足的*** ， 如果它至少在一个世界里为真 (参见 CSPs!)

假设我们有一个超高效的 SAT solver; 我们如何能用它来测试蕴涵关系?

- 假定 $\alpha \models \beta$
- 那么 $\alpha \Rightarrow \beta$ 为真 在所有世界（演绎公理）
- 因此 $\neg(\alpha \Rightarrow \beta)$ 为假 在所有世界
- 因此 $\alpha \wedge \neg\beta$为假 在所有世界, i.e., 不可满足的(unsatisfiable)

所以, 把否定的结论添加到 所知道的语句里, 测试其不可满足性 (un)satisfiability; 也叫 归谬法(reductio ad absurdum)

高效的 SAT solvers 需要 合取范式(***conjunctive normal form)***

合取范式(CNF)

每个句子可以被表达为

每个子句是一个 *文字* 的 *析取*

每个文字是一个正[illegible]字符或一个[illegible]定的

通过一序列标准的变换 转成 CNF:

- $R \Rightarrow (D \Leftrightarrow U)$
- $R \Rightarrow ((D \Rightarrow U) \wedge (U \Rightarrow D))$
- $\neg R \vee ((\neg D \vee U) \wedge (\neg U \vee D))$
- $(\neg R \vee \neg D \vee U) \wedge$
  $(\neg R \vee \neg U \vee D)$

替换双向条件，用两个暗示条件

替换 $\alpha \Rightarrow \beta$ 用 $\neg\alpha \vee \beta$

分配 $\vee$ 到 $\wedge$

高效的SAT问题求解器(SAT solvers)

DPLL (Davis-Putnam-Logemann-Loveland) 是 现代SAT求解器的核心算法

本质上是一个对模型的回溯搜索，和一些额外的技术:

- ***提早终止***: 如果
  - 所有子句都被满足; e.g., $(A \vee B) \wedge (A \vee \neg C)$ 被满足，通过 {A=true}
  - 任何一个子句为假; e.g., $(A \vee B) \wedge (A \vee \neg C)$ 为假，当 {A=false,B=false}

- ***纯文字（字符）***: 如果一个字符在剩下所有未满足的子句里的符号都是统一的，那么赋给这个字符那个值
  - 例如, A 是纯的，并且正号的 $(A \vee B) \wedge (A \vee \neg C) \wedge (C \vee \neg B)$ 所以赋给 true

- ***单元子句***: 如果一个子句只剩下一个单一的文字，那么给这个字符赋值使之满足该子句
  - 例如, 如果 A=false, $(A \vee B) \wedge (A \vee \neg C)$ becomes $(\text{false} \vee B) \wedge (\text{false} \vee \neg C)$, i.e. $(B) \wedge (\neg C)$
  - 满足单元子句的过程中经常会导致进一步的传递，产生新的单元子句。

DPLL 算法

```
function DPLL(子句集,字符集,模型) returns true or false
    if every 子句 in 子句集 is true in 模型 then return true
    if 某个 子句 in 子句集 is false in 模型 then return false
    P,value ←FIND-PURE-SYMBOL(字符集,子句集,模型)
    if P is non-null then return DPLL(子句集, 字符集–P, 模型∪{P=value})
    P,value ←FIND-UNIT-CLAUSE(子句集,模型)
    if P is non-null then return DPLL(子句集, 字符集–P, 模型∪{P=value})
    P ← First(字符集); rest ← Rest(字符集)
    return or(DPLL(子句集,rest,模型∪{P=true}),
              DPLL(子句集,rest,模型∪{P=false}))
```

效率

DPLL的简单实现: 求解 ~100 变量

额外技巧:

- 变量和值的选取排序 (参见 CSPs)
- 分治法（divide and conquer）
- 记录下 无法求解的情况，作为额外的子句，用来避免重蹈覆辙
- 索引和 增量计算技巧，使得DPLL 算法的每一步都是高效的 (通常 O(1))
  - 索引子句中每个变量（字符）的符号（正或是否定的）
  - 变量赋值过程中持续记录已满足的子句数量
  - 持续记录每个子句中剩余的文字符号的数量

DPLL的真正实现: 可以求解 ~10,000,000 变量

SAT 求解器在现实中的应用

电路验证: 超大规模集成电路 是否给出正确的计算?

软件验证: 程序是否计算正确的结果?

软件综合: 哪些程序计算正确结果?

协议验证: 这个安全协议能否被攻破?

协议合成: 哪些协议对于这个任务是安全的?

规划: 智能体的行为规划?

总结

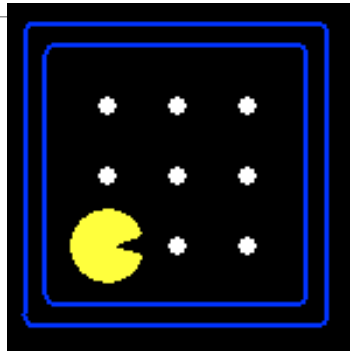
- 一种可能的智能体框架: 知识 + 推理
- 逻辑 提供了一种对知识进行编码的正规方法
  - 一个逻辑的定义: 语法, 可能世界的集合, 真值条件
- 逻辑推理计算句子间的导出（蕴涵）关系

人工智能导论：逻辑型的智能体

一个基于知识的智能体

```
function KB-AGENT(percept) returns 一个行动
  内部记录：KB, 知识库
            t, 整数, 初始为 0
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t←t+1
  return action
```

举例：部分可观察的 Pacman



Pacman 行动只能根据局部的感知信息

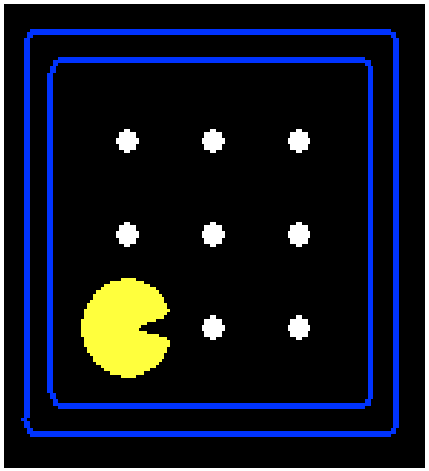
- 四个布尔感知变量 代表在每个方向是否有墙

它需要什么知识能开始行动?

- ***传感模型***: 句子 说明 当前感知变量是如何被当前的状态变量所决定的
- ***转换模型***: 句子 说明 下一个状态变量是如何被当前状态变量和Pacman的行动所决定的
- ***初始条件***: Pacman对于初始状态的知识
- ***领域约束***: 普通的事实, 例如，Pacman智能在一个时间做一件事，并且在一个时间只能出现在一个地方

所涉及到的Pacman的变量

- Pacman的位置
  - At_1,1_0 (Pacman 在[1,1] 在时刻 0) At_3,3_1 等
- 墙的位置
  - Wall_0,0 Wall_0,1 …
- 感知到的
  - Blocked_W_0 (向西被墙阻挡，在时刻 0) 等.
- 行动
  - W_0 (Pacman 向西移动，在时刻 0), E_0 等.



NxN 个世界， T 个时刻 => $N^2T + N^2 + 4T + 4T = O(N^2T)$ 变量数

2^{N^2T} 可能的世界!， N=10, T=100 => 10^{3010}

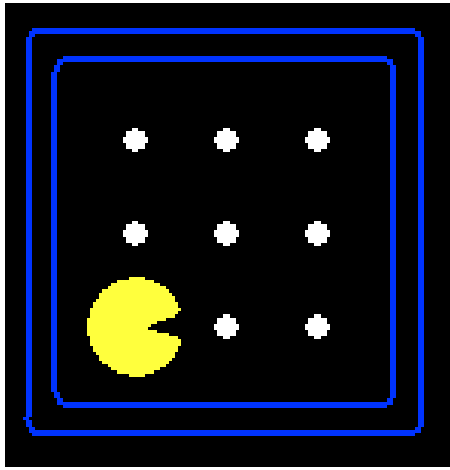
传感模型

描述如何产生 Pacman的感知

Pacman 感觉到向西有一面墙在时刻 t， *当且仅当* 他在位置 x,y 并且 有一面墙在位置 x-1,y

- Blocked_W_0 ⇔
- ((At_1,1_0 ∧ Wall_0,1) v
  (At_1,2_0 ∧ Wall_0,2) v
  (At_1,3_0 ∧ Wall_0,3) v ….)

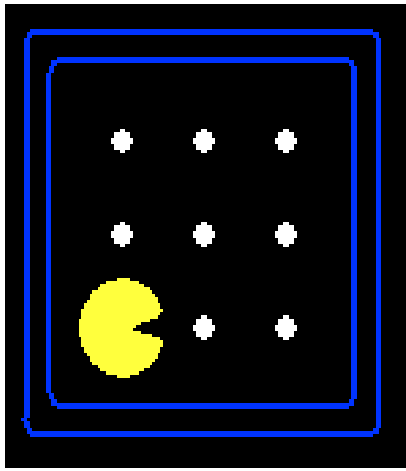
- 这样的句子有多少？



另一种传感器模型的问题

如果在时刻 t 他在位置 x,y 并且 有一堵墙在位置 x-1,y, 则 Pacman 在时刻 t 感知到一堵墙在他的西面

- $At\_1,1\_0 \wedge Wall\_0,1 \Rightarrow Blocked\_W\_0$
- $At\_1,1\_1 \wedge Wall\_0,1 \Rightarrow Blocked\_W\_1$
- ....
- $At\_3,3\_9 \wedge Wall\_3,4 \Rightarrow Blocked\_N\_9$



- 这种表达的问题
  - 不完整
  - 只是说 在这些条件下感知变量为真
  - 但没说 感知变量何时为假

如果左边为假，右边是真还是假？

转换模型 (transition model)

- 每个 ***状态变量*** 在每个时刻如何获得它的值?
- 部分可感知的Pacman里的状态变量 是 At_x,y_t , 例如, At_3,3_17
- 一个状态变量获得它的值，根据 ***后继状态公理 (successor-state axiom)***
  - $X_t \Leftrightarrow [X_{t-1} \wedge \neg$(某个 action$_{t-1}$ 使之为 false)] v
    [$\neg X_{t-1} \wedge$ (某个 action$_{t-1}$ 使之为 true)]
- 对于 Pacman 的位置:
  - At_3,3_17 $\Leftrightarrow$ [At_3,3_16 $\wedge \neg$((¬Wall_3,4 $\wedge$ N_16) v (¬Wall_4,3 $\wedge$ E_16) v …)]
    v [¬At_3,3_16 $\wedge$ ((At_3,2_16 $\wedge$ ¬Wall_3,3 $\wedge$ N_16) v
    (At_2,3_16 $\wedge$ ¬Wall_3,3 $\wedge$ N_16) v …)]

初始状态

- 智能体可能知道它的初始位置:
  - $At\_1,1\_0 \wedge \neg At\_1,2\_0 \wedge \neg At\_1,3\_0 \ldots$
- 或者, 它可能不知道:
  - $At\_1,1\_0 \vee At\_1,2\_0 \vee At\_1,3\_0 \vee \ldots \vee At\_3,3\_0$
- 我们也需要一个 ***值域约束*** – 一个时间只能做一件事!
  - $\neg(W\_0 \wedge E\_0) \wedge \neg(W\_0 \wedge S\_0) \wedge \ldots$
  - $\neg(W\_1 \wedge E\_1) \wedge \neg(W\_1 \wedge S\_1) \wedge \ldots$
  - $\ldots \wedge (W\_0 \vee E\_0 \vee N\_0 \vee S\_0) \wedge \ldots$

状态估计

回忆智能体在部分可观察情况下的 ***信念状态(belief state)*** 定义:

- 给定行动和当前的感知，与之相符合的世界状态的集合
- ***状态估计*** 是指保持(预测/更新)当前的信念状态

对于一个逻辑型的智能体, 计算在当前状态下哪些变量为真，只不过是一个逻辑推理的问题

- 例如, 询问是否 KB ∧ <actions> ∧ <percepts> ⊨ Wall_2,2
- 简单但效率低: 每一步的推理涉及到一个智能体整个行动感知的历史

状态估计，继续

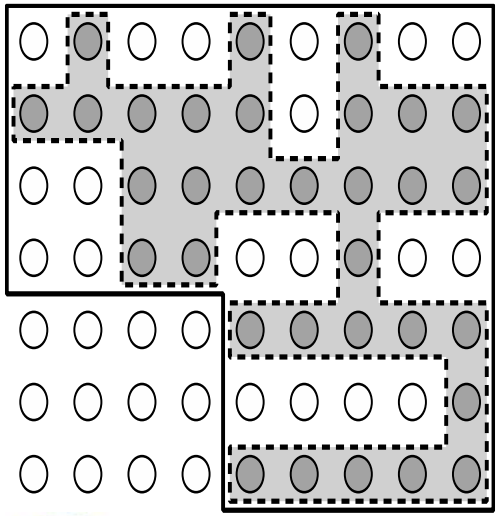
一个更“积极主动的”状态估计形式:

- 在每个行动和感知以后
  - 对每个状态变量 X_t
    - 如果 X_t 是被蕴涵的, 则加到 KB
    - 如果 $\neg X_t$ 是被蕴涵的, 则加到 KB

对于准确的状态评估是否这就足够?

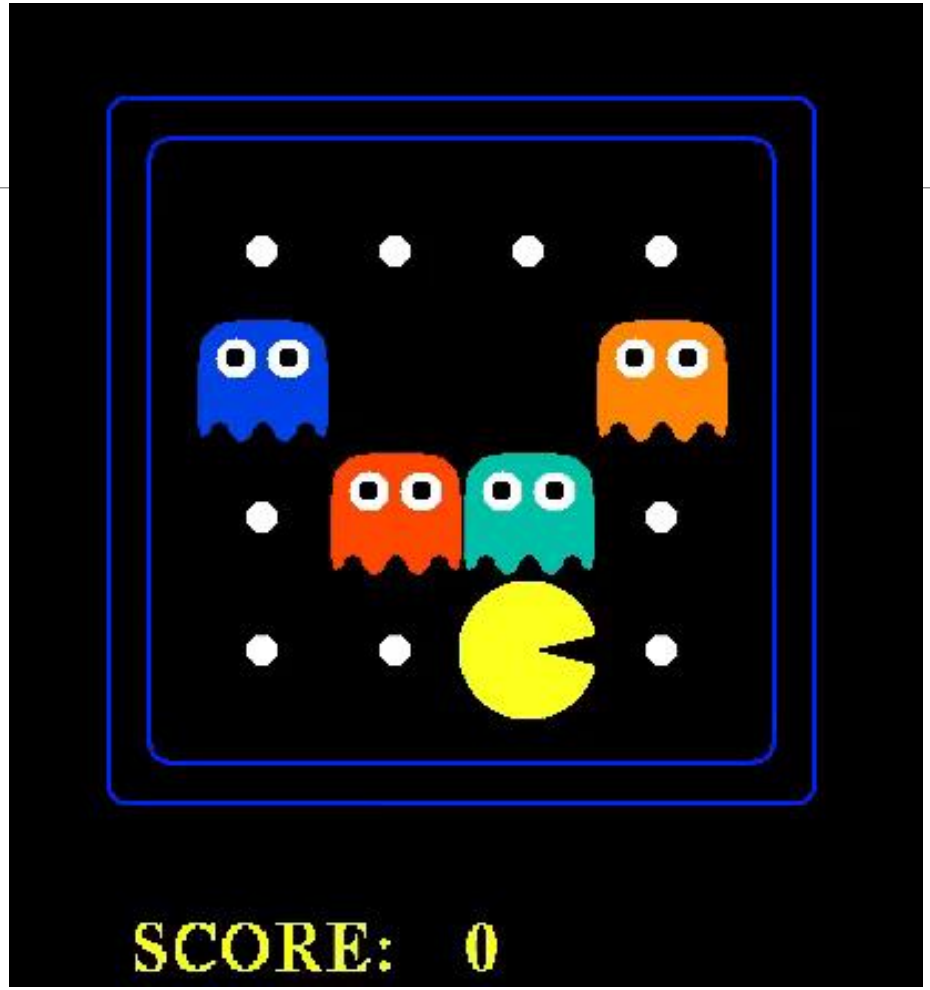
- 不是! 可能的情况是 X_t 或 $\neg X_t$ 都不被蕴涵, 并且 Y_t 或 $\neg Y_t$ 也都不被蕴涵, 但是某个约束, 例如, $X_t \vee Y_t$, *是* 被蕴涵的
  - 例如: 初始不确定性的智能体的位置

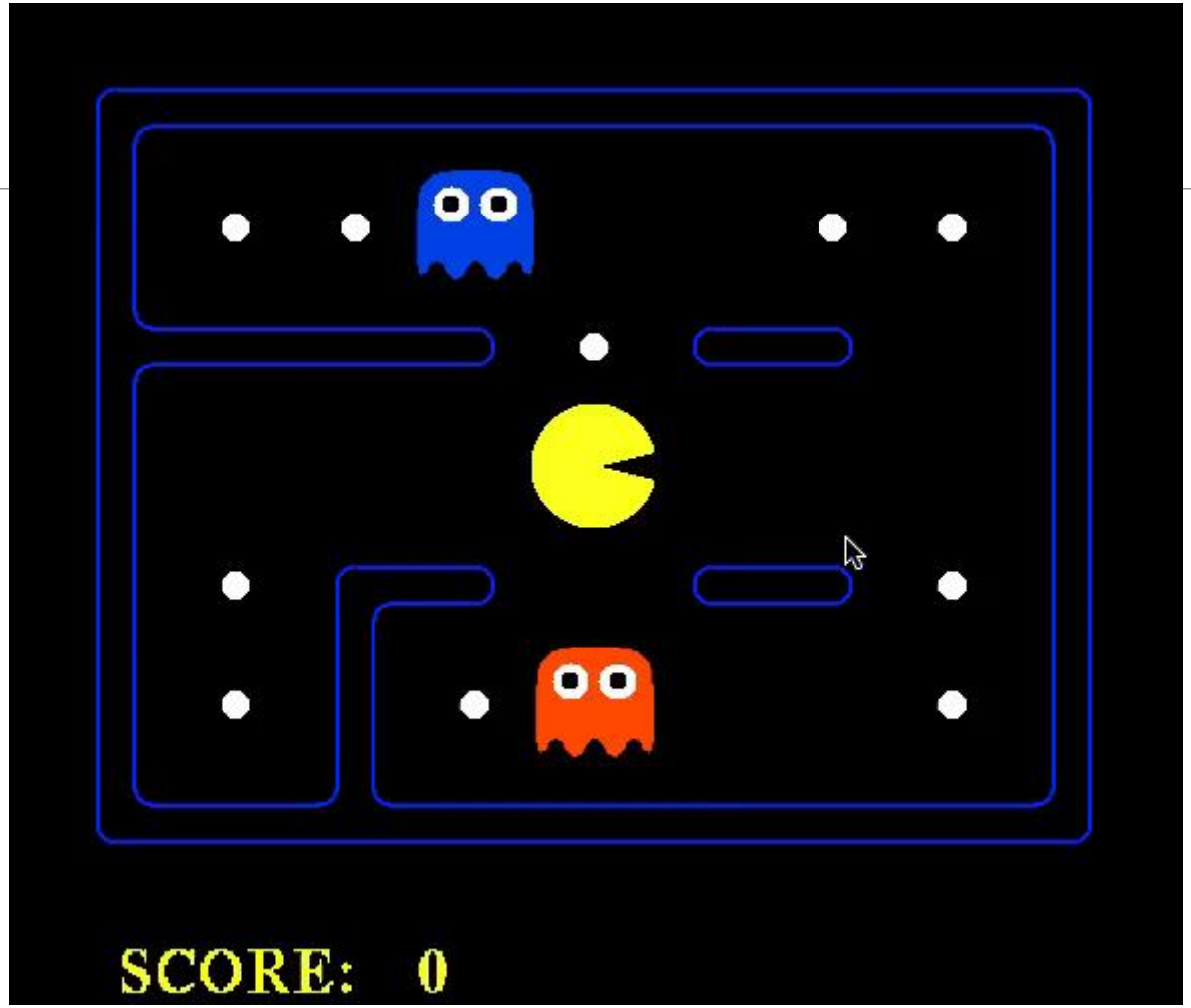
普遍来讲，完美的状态估计是很难达到的



可满足(satisfiability)来解规划(Planning)问题

- 给定一个超高效的 SAT 求解器，我们能用它来规划智能体的行动吗?
- 是的, 对于完全可观察的, 决定性的环境:
  - 规划问题是可解的 当且仅当 存在某个可满足的赋值对于所有变量
  - 相应的行动变量的真值构成了解
- 对于时间 T = 1 到无穷, 按以下内容构建知识库 (KB)， 并运行SAT solver:
  - 初始状态, 值域约束
  - 截至到时刻 T 的转换模型语句(包括后继状态转换公理，对于所有可能的行动)
  - 目标(Goal) 在时刻 T 为真


SCORE: 0


SCORE: 0


SCORE: 0

总结

- 声明法/陈述法(declarative approach) 构建智能体的框架
  - 知识库里是陈述语句（包括公理，感知到的事实）
  - 逻辑推理– 事实被蕴涵的推理证明，规划智能体行动
- 现代超高效的SAT 求解器，使得此法可在实践中可行
- 弱点：语句表达
  - 例如“对于每个时刻 t”，“对于每个方块位置[x,y]”
  - 一阶逻辑(first order logic) 提高了语句的表达性；其逻辑推理方法与命题逻辑的方法一致