命题逻辑(PROPOSITIONAL LOGIC)：
语法，语义和推理

判断布尔逻辑表达式的值

$$\exists y \forall x (x = y + 1) \Longrightarrow \forall w \forall z (w = z)$$

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

知识库	领域特定事实 (Domain-specific facts)
推理引擎	领域独立的通用算法和代码

逻辑（Logic）

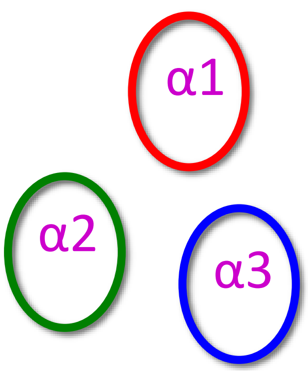
语法(Syntax): 定义句子(什么样的句子是允许的)

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

一阶谓词逻辑 (First-order logic)

- 语法: $\forall x\, \exists y\, P(x,y) \land \neg Q(Joe,f(x)) \Rightarrow f(x)=f(y)$
- 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

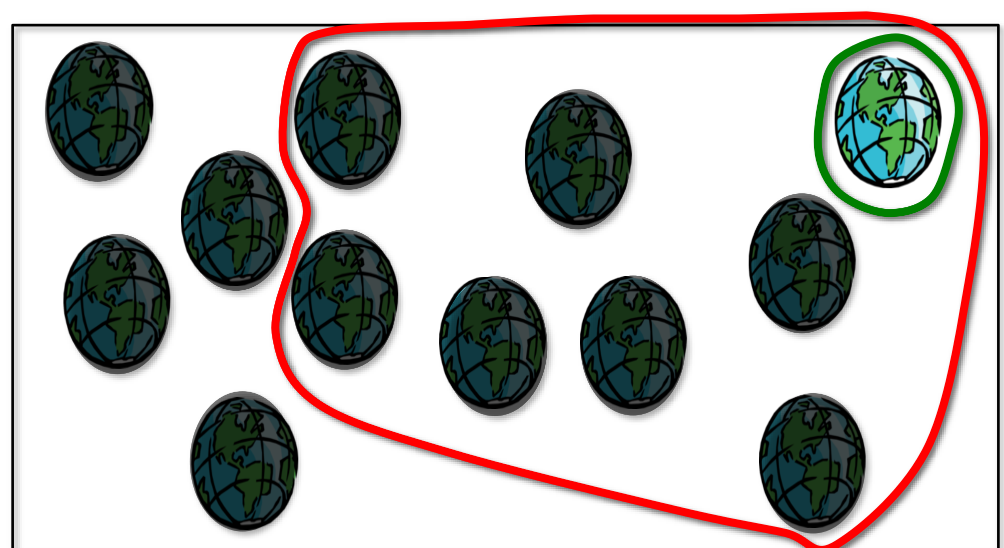
- 换句话说，α-的世界（为真的那些世界） 是 β-的世界的一个子集 [***models***(α) $\subseteq$ ***models***(β)]

例如, $\alpha_2 \models \alpha_1$

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

α_1

α_2



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：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

Propositions (命题)

- 一个命题是一个声明句子，用来声明一个事实，这个事实要么是真，要么是假，但不会既是真又是假。
- 命题举例
  - 北京是中国的首都。
  - 海南省位于中国最南部。
  - 1+1=2。
  - 2+2=6 。
- 非命题句子
  - 现在几点钟？
  - 仔细阅读这段文字。
  - x+1=2
  - x+y=z

命题逻辑的应用: 逻辑谜题推理

泥巴孩子谜题: (muddy children puzzle)

一个父亲告诉他的两个孩子，一个男孩和一个女孩，在后院玩的时候不要把自己弄脏。然而在玩耍时，两个孩子的额头都被泥巴弄脏了。当他们回来站在父亲面前时，父亲对他们俩说

“**你们俩中至少有一个脑门上有泥巴**”，然后父亲问这两个孩子一个问题“**你知道你的脑门上有泥巴吗？**”回答只能是：**是或否**。

父亲问这个问题**两次，两个孩子的回答会是什么？**

假设一个孩子只能看到另一个孩子的泥巴，但看不到自己脑门上的泥巴；两个孩子都是诚实的，并且都是同时回答每一个问题。

命题逻辑的连接符号

- 一个命题符号（英文字母）代表一个命题语句
- $\neg p$：　非p，真值是P的相反
- Conjunction（联合，合取）: $p \wedge q$ ["与"]
- Disjunction（分离，析取）: $p \vee q$ ["或"]
- Exclusive or (互斥或，异或)：$p \oplus q$
- 条件声明：　$p \Rightarrow q$ ["if p then q"]（有的书上也用单横箭头表示）

  p: *假设（前提）*, q: *结论*
- 双向条件声明：　$p \Leftrightarrow q$ ["p if and only if q "]

  也可以定义为 $(p \Rightarrow q) \wedge (q \Rightarrow p)$

命题逻辑（Propositional logic）：语法

给定: 一组 命题字符 $\{X_1, X_2, \ldots, X_n, P, Q, R, North, \ldots\}$（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, 或叫规则）
- 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) = $\neg$ **then return** not(PL-TRUE?(Arg1(α),model))

if Op(α) = $\wedge$ **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

命题逻辑的应用：系统规范说明

举例，检查以下系统规范说明的一致性

1. 一个讯息是被存储在缓冲区里，或已被重新发送。
2. 一个讯息没有被存储在缓冲区里。
3. 如果一个讯息存储在缓冲区里，那么这条讯息已被重新发送。

命题逻辑的应用：系统规范说明

让 p 代表“这条讯息被存储在缓冲区里”

让 q 代表“这条讯息已被重新发送”

之前的三条规范可以表示为：

$$p \vee q, \qquad \neg p, \qquad p \to q$$

命题逻辑的应用：系统规范说明

如果再添加一条规范说明：“这条讯息还没有被重新发送”，那么新的逻辑句子集合变为：

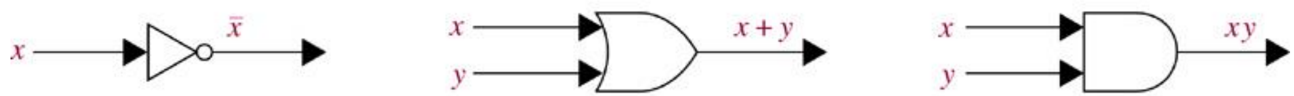
$$p \vee q, \qquad \neg p, \qquad p \rightarrow q \qquad \neg q$$

这个系统规范说明是不一致性的。

没有p 和 q 的真值可以使以上所有的声明句子都为真。

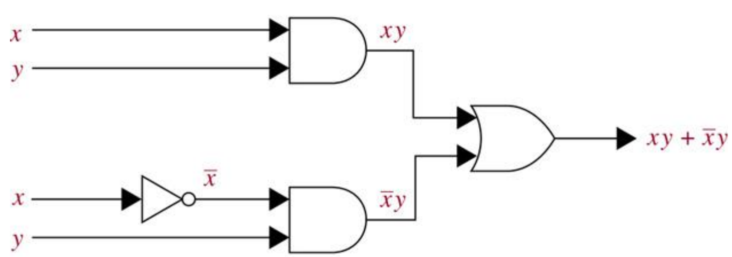
命题逻辑的应用：逻辑门电路

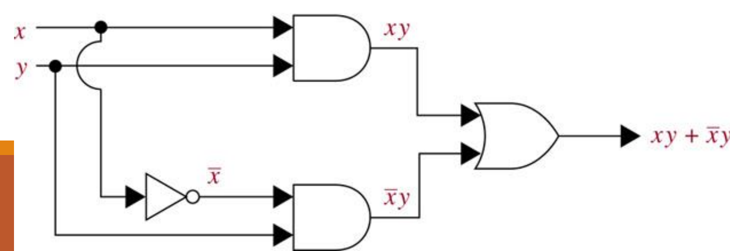
基本的逻辑门电路：



(a) Inverter (b) *OR* gate (c) *AND* gate

逻辑门电路的组合：





同义反复和矛盾式（Tautology and contradiction）

TABLE 1 Examples of a Tautology and a Contradiction.

p	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
T	F	T	F
F	T	T	F

- 一个复合命题句子:
  - Tautology（同义反复，重言式）： 总为真
  - Contradiction（矛盾，不一致）： 总为假
  - Contingency（偶然性）： neither a tautology nor a contradiction

Valid, satisfiable, or unsatisfiable?

$$\neg(x \vee \neg(x \wedge (z \vee T))) \implies \neg(y \wedge (\neg y \vee (T \implies F)))$$

逻辑等价（一致性）Logical equivalence

- p ≡ q : 复合句p 和 q 是逻辑等价的
  - 如果 p $\boxed{\leftrightarrow}$ q 是一个同义反复式
- 可以用真值表来决定两个命题句子是否逻辑等价

用真值表判断两个句子是否等价（同意义的）

- ┐(p v q) 和┐p ^┐ q 是等价的

TABLE 3 Truth Tables for $\neg(p \vee q)$ and $\neg p \wedge \neg q$.

p	q	$p \vee q$	$\neg(p \vee q)$	$\neg p$	$\neg q$	$\neg p \wedge \neg q$
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

$$p \to q \equiv \neg p \vee q$$

p	q	p→q	┐p˅q
T	T	T	T
T	F	F	F
F	T	T	T
F	F	T	T

De Morgan's laws

TABLE 2 De Morgan's Laws.
$\neg(p \wedge q) \equiv \neg p \vee \neg q$
$\neg(p \vee q) \equiv \neg p \wedge \neg q$

逻辑上的一致性

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

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

命题逻辑推理规则

- $((p \to q) \wedge p) \to q$ is the rule of inference called **modus ponens**（**肯定前件式推理**） (*mode that affirms*), or the **law of detachment**

$$\begin{array}{l} \quad p \\ \quad p \to q \\ \hline \therefore q \end{array}$$

TABLE 1 Rules of Inference.

Rule of Inference	*Tautology*	*Name*
p $p \to q$ $\therefore q$	$[p \wedge (p \to q)] \to q$	Modus ponens
$\neg q$ $p \to q$ $\therefore \neg p$	$[\neg q \wedge (p \to q)] \to \neg p$	Modus tollens
$p \to q$ $q \to r$ $\therefore p \to r$	$[(p \to q) \wedge (q \to r)] \to (p \to r)$	Hypothetical syllogism
$p \vee q$ $\neg p$ $\therefore q$	$[(p \vee q) \wedge \neg p] \to q$	Disjunctive syllogism
p $\therefore p \vee q$	$p \to (p \vee q)$	Addition
$p \wedge q$ $\therefore p$	$(p \wedge q) \to p$	Simplification
p q $\therefore p \wedge q$	$[(p) \wedge (q)] \to (p \wedge q)$	Conjunction
$p \vee q$ $\neg p \vee r$ $\therefore q \vee r$	$[(p \vee q) \wedge (\neg p \vee r)] \to (q \vee r)$	Resolution

Resolution（“归结”推理规则）

- 基于同义反复式： $((p \vee q) \wedge (\neg p \vee r)) \rightarrow (q \vee r)$
- 归结结果: $q \vee r$
- Let r=q, we have $(p \vee q) \wedge (\neg p \vee q) \rightarrow q$
- Let r=F, we have $(p \vee q) \wedge \neg p \rightarrow q$
- 在逻辑规划中重要的推理规则。

归结规则举例

- 使用归结法作为唯一的推理规则进行证明时，要求 the hypotheses（假设前提） and the conclusion（结论） 必须表达为 clauses（子句）
- **Clause（子句）**: a disjunction of variables or negations of these variables （变量或者其非形式的”或”表达式）

$$\text{Show } (p \wedge q) \vee r \text{ and } r \rightarrow s \text{ imply } p \vee s$$

$$(p \wedge q) \vee r \equiv (p \vee r) \wedge (q \vee r)$$

$$r \rightarrow s \equiv \neg r \vee s$$

合取范式(CNF)

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 v 到 ∧

- 每个句子可以表 … 取
- 每个子句是一 … *文字* … *析取*
- 每个文字是 … 个正的 … 符或 …
- 通过一序列标准的变换 转成 …
  - $R \Rightarrow (D \Leftrightarrow U)$
  - $R \Rightarrow ((D \Rightarrow U) \wedge (U \Rightarrow D))$
  - $\neg R \vee ((\neg D \vee U) \wedge (\neg U \vee D))$
  - $(\neg R \vee \neg D \vee U) \wedge$
    $(\neg R \vee \neg U \vee D)$

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*)**:

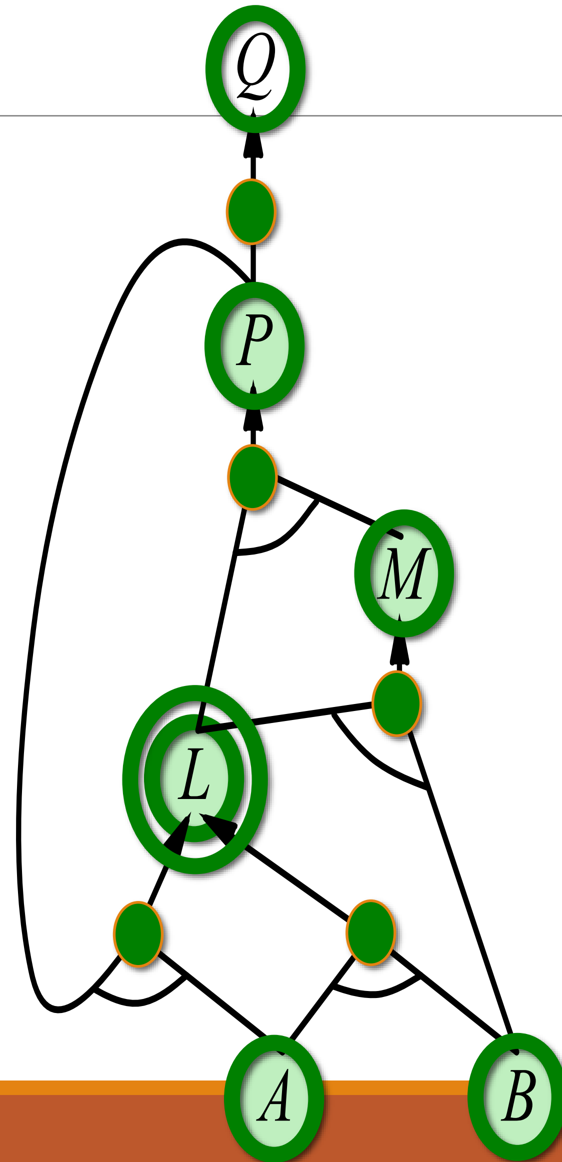
- 一组分离(disjunction)的文字(literals),只有一个是正的（其他都含否定符）; 可转成以下形式
- (字符的结合 '与' (conjunction)) $\Rightarrow$ 字符; 或
- 一个单一的字符 (注意 X 相当于 True $\Rightarrow$ X)

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~~ ~~L~~ ~~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 xxxx false true

B xxxx false true

L xxxx false true

M xxxx false true

P xxxx false true

Q xxxx false true

简单的模型检查(model checking)

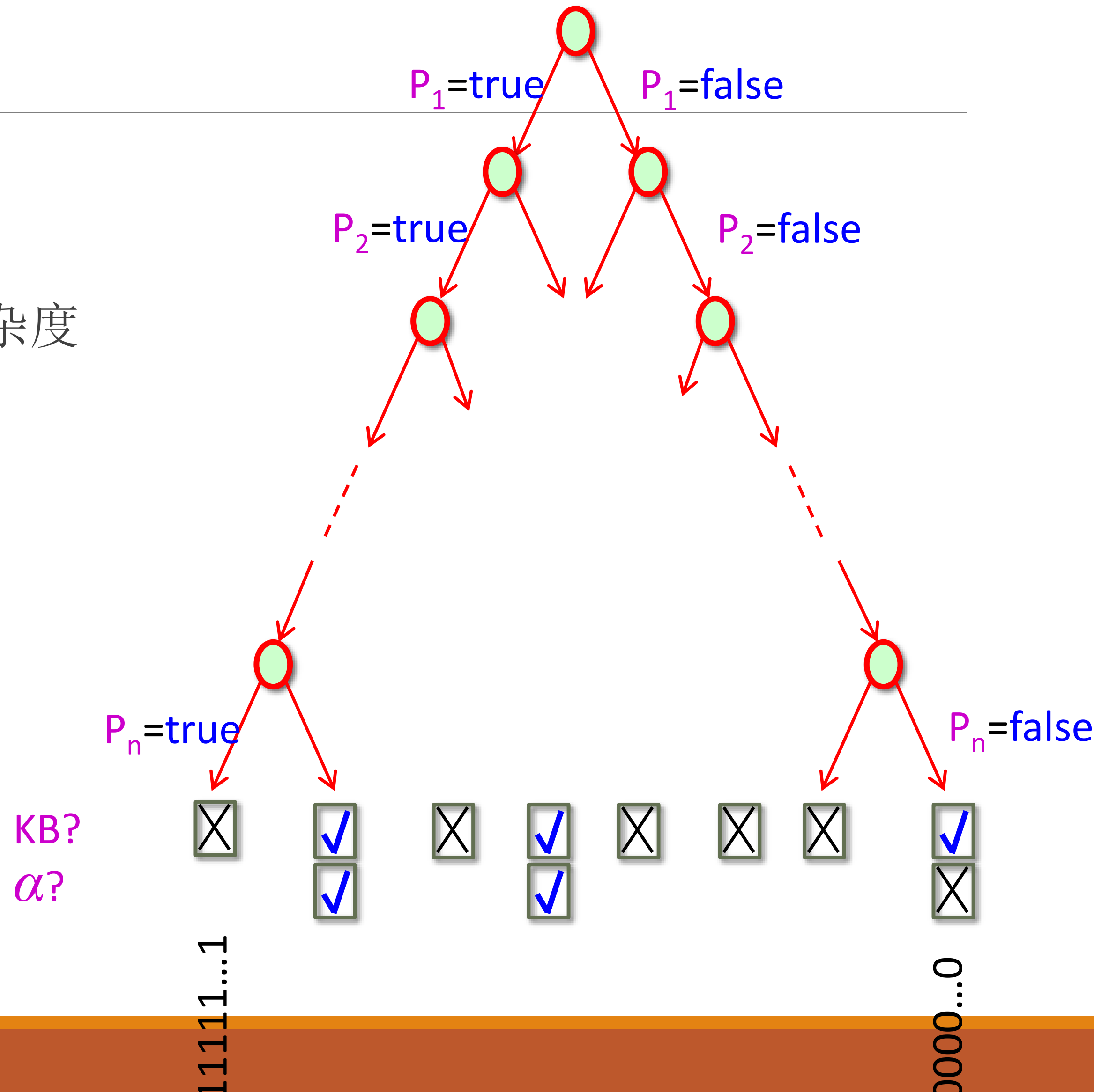
```
function TT-ENTAILS?(KB, α) returns true or false
  return TT-CHECK-ALL(KB,α,symbols(KB) U symbols(α),{})
function TT-CHECK-ALL(KB,α,symbols,model) returns true or false
  if empty?(symbols) then
      if PL-TRUE?(KB,model) then return PL-TRUE?(α,model)
      else return true
  else
      P ← first(symbols)
      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* ，如果它至少在一个模型（一组对逻辑变量的赋值）里为真。
- 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 $\alpha \Rightarrow \beta$ 在所有模型里为真 （演绎公理 deduction theorem）
  - 因此 $\neg(\alpha \Rightarrow \beta)$ 在所有模型里为假
  - 因此 $\alpha \wedge \neg\beta$ 在所有模型里为假, 即 *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;
  - 等价于 归谬法(reductio ad absurdum)，或反证法。

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

判读F是否是可满足的

$$F = (x_1 \vee \bar{x_2}) \wedge (x_2 \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_4) \wedge$$
$$(x_4 \vee \bar{x_1}) \wedge (\bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_2} \vee x_3)$$

高效的SAT问题求解器(SAT solvers)

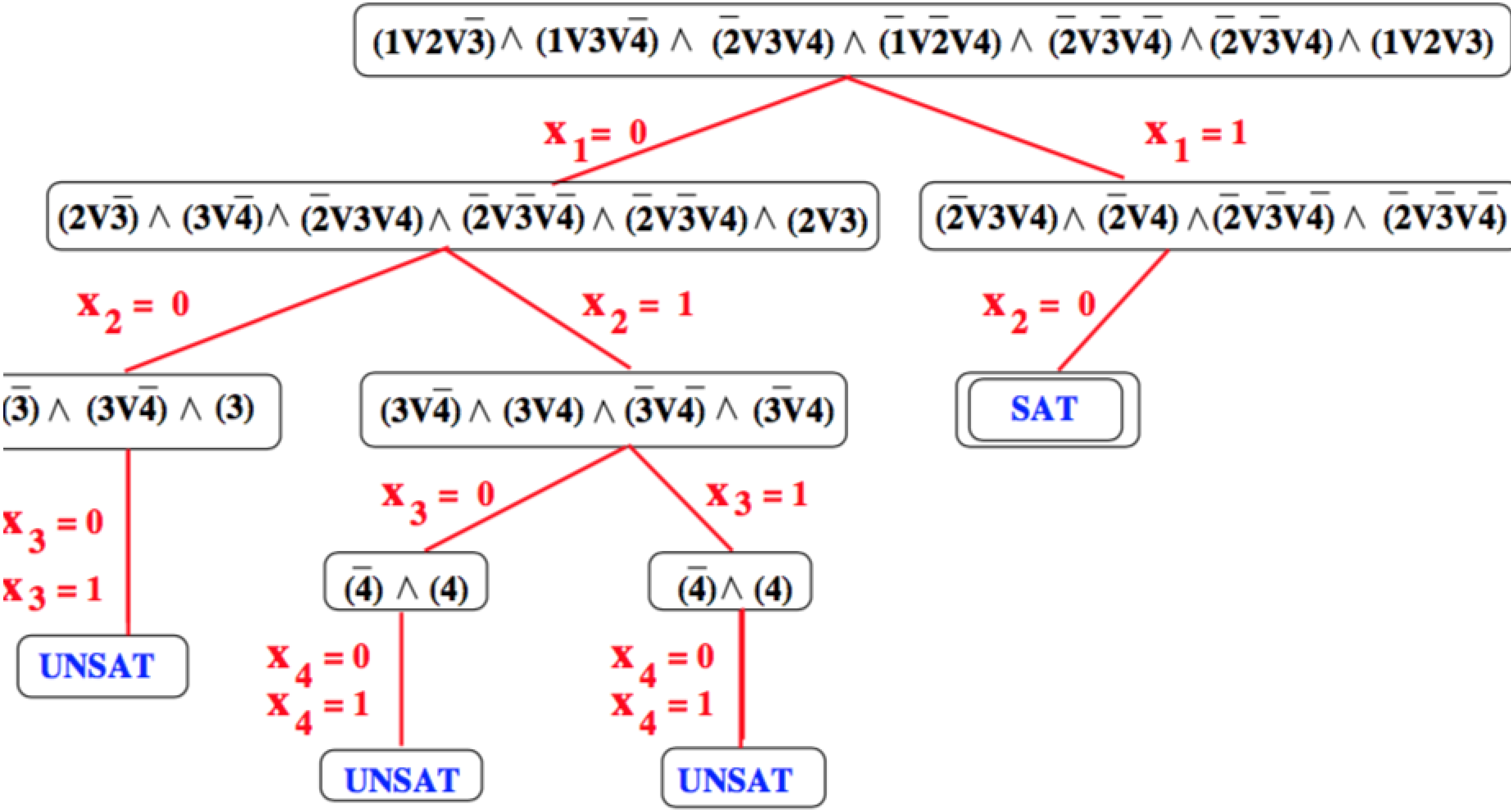
DPLL (Davis-Putnam-Logemann-Loveland) 是 现代SAT求解器的核心算法

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., $(A \lor B) \land (A \lor \neg C)$ 被满足，通过 {A=true}
  - 某一个子句为假; e.g., $(A \lor B) \land (A \lor \neg C)$ 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 $(A \lor B) \land (A \lor \neg C) \land (C \lor \neg B)$ 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, $(A \lor B) \land (A \lor \neg C)$ becomes $(false \lor B) \land (false \lor \neg C)$, i.e. $(B) \land (\neg C)$
  - 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}))
```



ig. **10.3** A sketch of the DPLL algorithm, acting on the formula $_1 \vee x_2 \vee \overline{x}_3)\wedge(x_1 \vee x_3 \vee \overline{x}_4)\wedge(\overline{x}_2 \vee x_3 \vee x_4)\wedge(\overline{x}_1 \vee x_2 \vee x_4)\wedge(\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)\wedge$ $_2 \vee \overline{x}_3 \vee x_4)\wedge(x_1 \vee x_2 \vee x_3)\wedge(\overline{x}_1 \vee x_2 \vee \overline{x}_4)$. In order to get a more readable figure, ıe notation has been simplified: a clause such as $(\overline{x}_1 \vee x_2 \vee x_4)$ is denoted here by $(\overline{1}\ 2\ 4)$.

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

Definition 2 *We say that an algorithm runs in* **Polynomial Time** *if, for some constant c, its running time is $O(n^c)$, where n is the size of the input.*

Definition 3 *A Problem A is* **poly-time reducible** *to problem B (written as $A \leq_p B$) if we can solve problem A in polynomial time given a polynomial time black-box algorithm for problem B. Problem A is* **poly-time equivalent** *to problem B ($A =_p B$) if $A \leq_p B$ and $B \leq_p A$.*

P, NP, and NP-Completeness

Definition 5 **P** *is the set of decision problems solvable in polynomial time.*

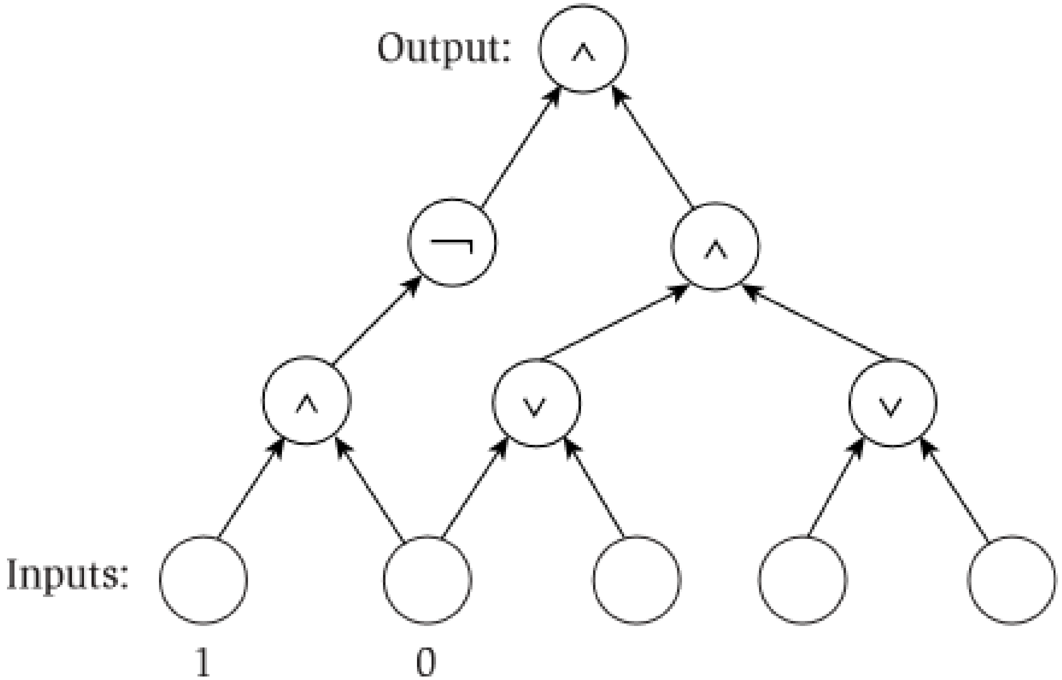
Definition 6 **NP** *is the set of decision problems that have polynomial-time* verifiers. *Specificially, problem Q is in* **NP** *if there is a polynomial-time algorithm $V(I, X)$ such that:*

- *If I is a YES-instance, then there exists X such that $V(I, X) =$ YES.*
- *If I is a NO-instance, then for all X, $V(I, X) =$ NO.*

Definition 7 *Problem Q is* **NP***-complete if:*

1. *Q is in* **NP**, *and*
2. *For any other problem Q' in* **NP**, *$Q' \leq_p Q$.*

Circuit Satisfiability is NP-complete



ι circuit with three inputs, two additional sources that have assigned truth
ɔne output.

To express independent set problem

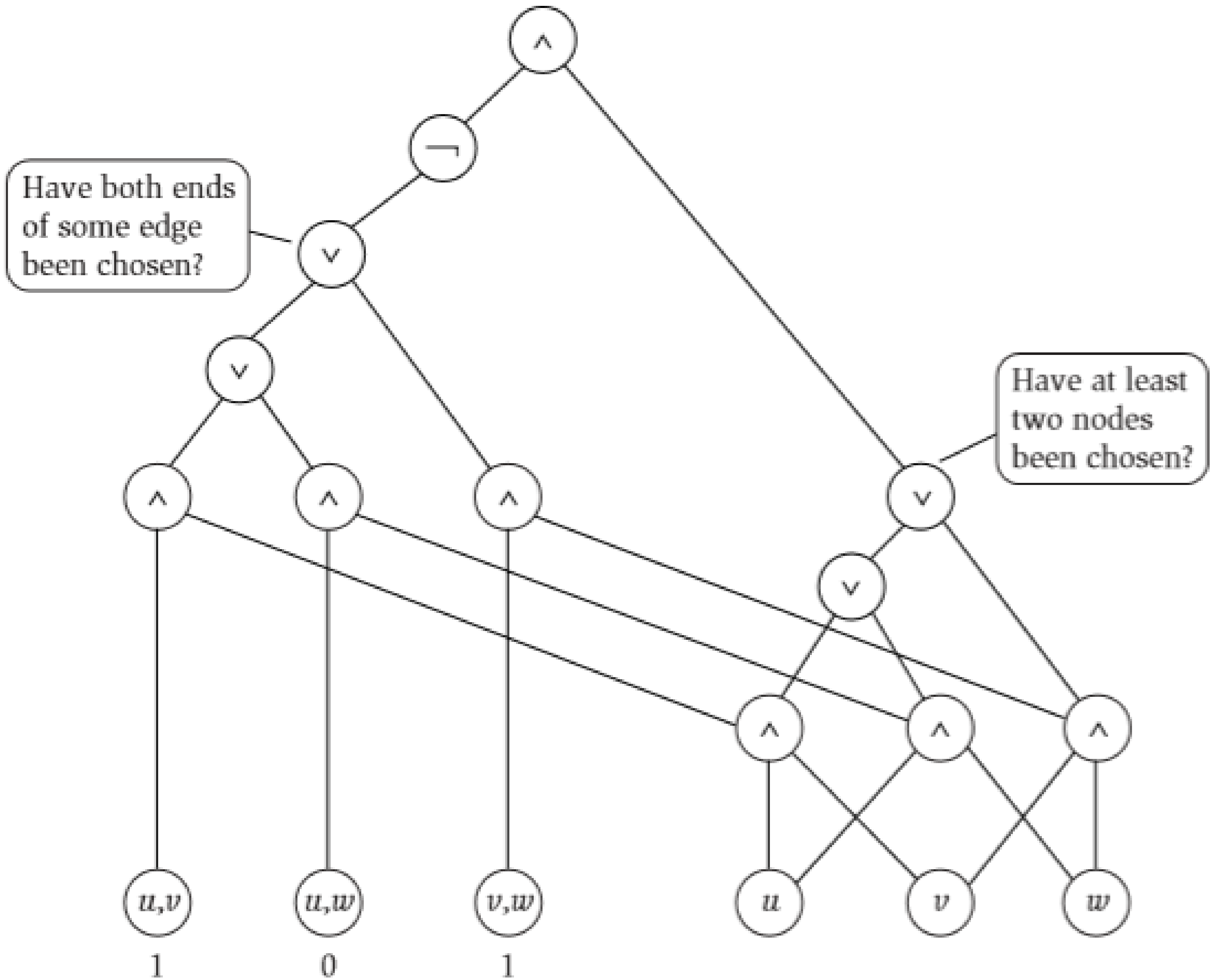


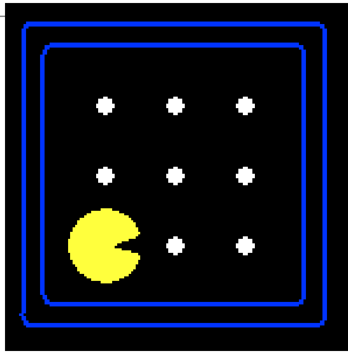
Figure 8.5 A circuit to verify whether a 3-node graph contains a 2-node independent set.

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

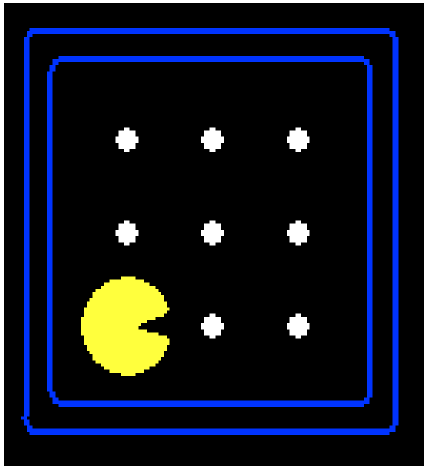


Pacman 行动只能根据局部的感知信息
- 四个布尔感知变量 代表在每个方向是否有墙

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在时刻 0位于[1,1])   At_3,3_1 等
- 墙的位置
  - Wall_0,0   Wall_0,1 …
- 感知到的
  - Blocked_W_0 (向西走被墙阻挡，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

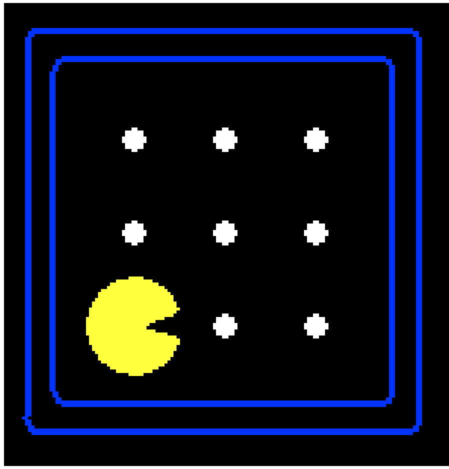
2^{N^2T} 可能的模型（世界）！， N=10, T=100 => 10^{3010}

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 ....)

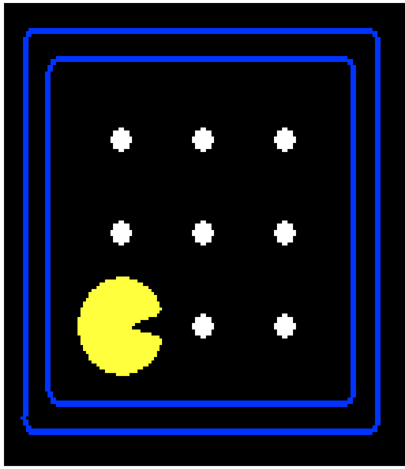


- 这样的句子有多少？

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 ∧ Wall_0,1 ⇒ Blocked_W_0
- At_1,1_1 ∧ Wall_0,1 ⇒ Blocked_W_1
- ....
- At_3,3_9 ∧ Wall_3,4 ⇒ 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)]
- 举例，对于 Pacman 的位置:
  - At_3,3_17 ⇔ [At_3,3_16 ∧ ¬((¬Wall_3,4 ∧ N_16) v (¬Wall_4,3 ∧ E_16) v ...)]
    v [¬At_3,3_16 ∧ ((At_3,2_16 ∧ ¬Wall_3,3 ∧ N_16) v
    (At_2,3_16 ∧ ¬Wall_3,3 ∧ N_16) v ...)]

初始状态

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

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

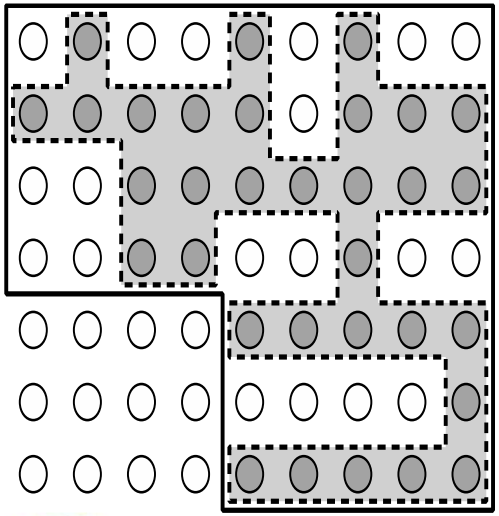
一个更 “积极主动的” 状态估计形式:

- 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?

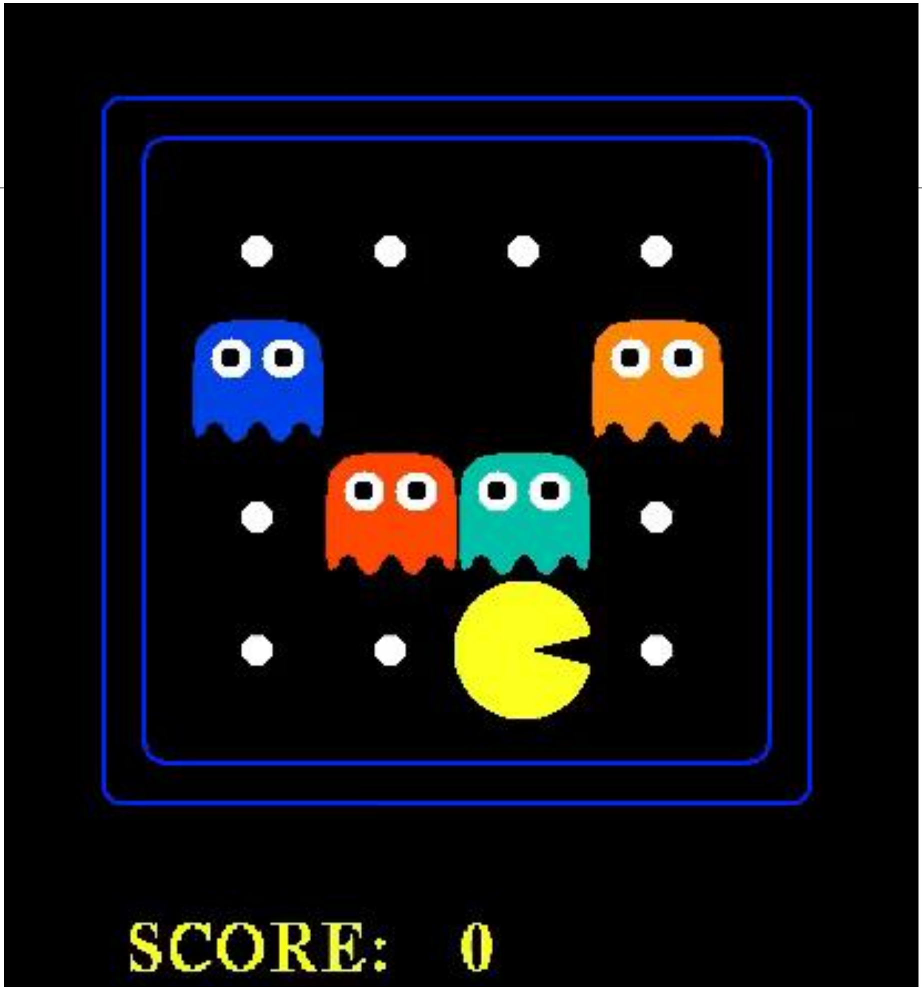
- 不是! 可能的情况是 X_t 或 $\neg X_t$ 都不被蕴涵, 并且 Y_t 或 $\neg Y_t$ 也都不被蕴涵, 但是某个约束, 例如, $X_t \vee Y_t$, *是* 被蕴涵的
  - 例如: 初始不确定性的智能体的位置

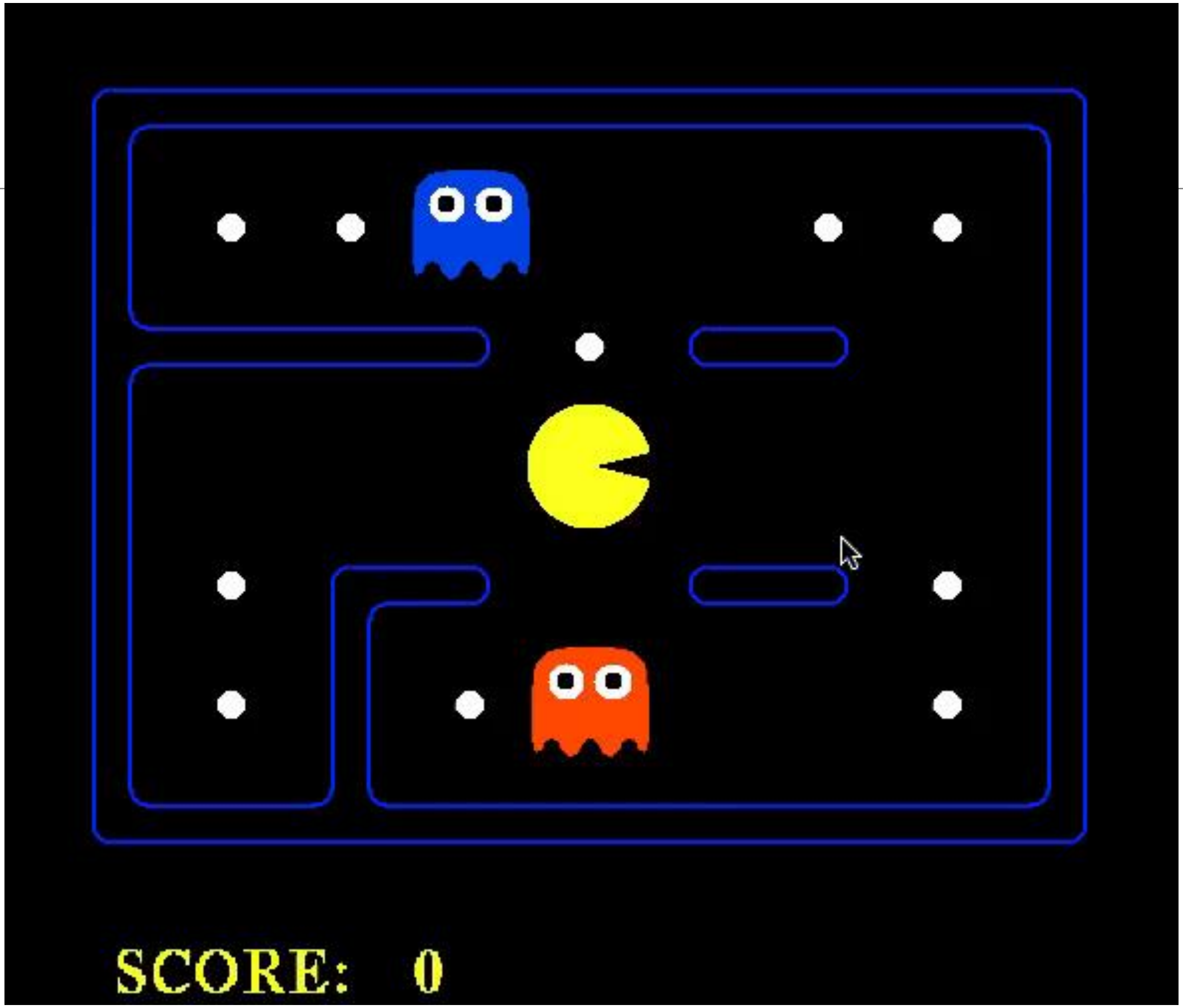
普遍来讲，完美的状态估计是很难达到的



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

SAT-Plan: 找到行动规划

找到最短行动规划路径

function SATPLAN(*init*, *transition*, *goal*, $T_{\max}$) **returns** solution or failure
  inputs: *init*, *transition*, *goal*, constitute a description of the problem
    $T_{\max}$, an upper limit for plan length

  for $t = 0$ **to** $T_{\max}$ **do**
    cnf ← TRANSLATE-TO-SAT(*init*, *transition*, *goal*, *t*)
    model ← SAT-SOLVER(*cnf*)
    if *model* is not null **then**
      return EXTRACT-SOLUTION(*model*)
  return *failure*

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