1 命题逻辑

1.1 判断语句

¬：p 的否定(negation)，记作¬ p，表示“上周我没有中彩”，或者用等价语句“上周我中彩不是真的。”

∨：表示 p 和 r 中至少有一个为真的叙述：“上周我中了彩，或者，上周我中了赛马大奖。”我们将这个判断句记为 $p \vee r$，并称其为 p 和 r 的析取(disjunction)[⊖]。

∧：$p \wedge r$ 表示 p 和 r 的相当幸运的合取(conjunction)：“上周我中了彩和赛马大奖。”

→：最后一个规则，但肯定不是最不重要的规则，语句“如果上周我中了彩，那么我买了一张彩票”表达了 p 和 q 之间的一种蕴涵(implication)，指出 q 是 p 的一个逻辑结果。我们用 $p \to q$ 表示这一点[⊜]，并称 p 是 $p \to q$ 的假设(assumption)，而 q 是其结论(conclusion)。

4个基本的逻辑连接词： ¬ ∨ ∧ ->

约定：

优先级：

1. ¬ 大于 ∨ ∧ 大于 ->
2. 蕴含 -> 是右结合的。

逻辑等价（6个逻辑等价式）【存疑?】

定义了同样的概念。一些逻辑等价公式的例子如下：

$\neg(p \wedge q) \dashv\vdash \neg q \vee \neg p$　　$\neg(p \vee q) \dashv\vdash \neg q \wedge \neg p$

$p \to q \dashv\vdash \neg q \to \neg p$　　$p \to q \dashv\vdash \neg p \vee q$

$p \wedge q \to p \dashv\vdash r \vee \neg r$　　$p \wedge q \to r \dashv\vdash p \to (q \to r)$

读者应该能够用自然演绎逻辑来证明这六个等价式。

1.2 自然演绎

1.2.1 自然演绎规则

- 合取规则：合取引入、合取消去
- 双重否定规则
- 蕴含消去规则：【最著名的分离规则】、
- 蕴含引入规则
- 析取规则
- 否定规则

1.2.2 派生规则

- 反证规则（PBC）：推出矛盾，说明前提不成立
- 排中律（LEM）：如果 φ∨¬φ 是真的，那么无论 φ 是什么，它一定是真的。

1.2.3 自然演绎总结：

自然演绎的基本规则：

	引入	消去
$\wedge$	$\dfrac{\phi \quad \psi}{\phi \wedge \psi} \wedge \mathrm{i}$	$\dfrac{\phi \wedge \psi}{\phi} \wedge \mathrm{e}_1 \quad \dfrac{\phi \wedge \psi}{\psi} \wedge \mathrm{e}_2$
$\vee$	$\dfrac{\phi}{\phi \vee \psi} \vee \mathrm{i}_1 \quad \dfrac{\psi}{\phi \vee \psi} \vee \mathrm{i}_2$	$\dfrac{\phi \vee \psi \quad \begin{array}{c}[\phi]\\ \vdots \\ \chi\end{array} \quad \begin{array}{c}[\psi]\\ \vdots \\ \chi\end{array}}{\chi} \vee \mathrm{e}$
$\rightarrow$	$\dfrac{\begin{array}{c}[\phi]\\ \vdots \\ \psi\end{array}}{\phi \rightarrow \psi} \rightarrow \mathrm{i}$	$\dfrac{\phi \quad \phi \rightarrow \psi}{\psi} \rightarrow \mathrm{e}$

图 1-2　命题逻辑的自然演绎规则

	引入	消去
$\neg$	$\dfrac{\begin{array}{c}[\phi]\\ \vdots \\ \bot\end{array}}{\bot \quad \phi} \neg \mathrm{i}$	$\dfrac{\phi \quad \neg \phi}{\bot} \neg \mathrm{e}$
$\bot$	（$\bot$ 没有引入规则）	$\dfrac{\bot}{\phi} \bot \mathrm{e}$
$\neg\neg$		$\dfrac{\neg\neg \phi}{\phi} \neg\neg \mathrm{e}$

一些有用的派生规则：

$$\frac{\phi \rightarrow \psi \quad \neg \psi}{\neg \phi} \mathrm{MT} \qquad \frac{\phi}{\neg\neg \phi} \neg\neg \mathrm{i}$$

$$\frac{\begin{array}{c}[\neg \phi]\\ \vdots \\ \bot\end{array}}{\phi} \mathrm{PBC} \qquad \frac{}{\phi \vee \neg \phi} \mathrm{LEM}$$

1.2.4 逻辑等价

定义：

> 逻辑命题之间的逻辑等价

定义 1.25　设 ϕ 和 ψ 是命题逻辑的公式。我们说 ϕ 和 ψ 是逻辑等价（provably equivalent）的，当且仅当矢列式 $\phi \vdash \psi$ 和 $\psi \vdash \phi$ 都是有效的；即可以从 ϕ 出发证明 ψ，反之亦然。正如前面看到的，我们用 $\phi \dashv\vdash \psi$ 表示 ϕ 和 ψ 是逻辑等价的。

1.3 形式化定义命题逻辑

被严格定义的字符串公式，这样的公式成为合式公式（*well-formed*）。

定义 1.27 命题逻辑的合式公式是且仅是那些有限次使用下列构造规则所得到的符号串：

原子：每个命题原子 p，q，r，…和 p_1，p_2，p_3，…是合式公式。

$\neg$：如果 ϕ 是合式公式，那么 $(\neg \phi)$ 也是合式公式。

$\wedge$：如果 ϕ 和 ψ 是合式公式，那么 $\phi \wedge \psi$ 也是合式公式。

$\vee$：如果 ϕ 和 ψ 是合式公式，那么 $\phi \vee \psi$ 也是合式公式。

$\rightarrow$：如果 ϕ 和 ψ 是合式公式，那么 $\phi \rightarrow \psi$ 也是合式公式。

认识到这个定义是计算机所期待的这一点很关键，而且我们没有使用前一节所约定的绑定优先级。

根据BNF（一种定义语法的范式，可以理解成一种规定），可以把命题逻辑的定义更加紧凑地写为：

$$\phi ::= p \mid (\neg \phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \qquad (1\text{-}3)$$

`::=` 的右边表示一个合式公式。

定义就是用来规定哪些公式属于命题逻辑，哪些不是的。那么怎样判断一个公式是否是合式呢？方法是对公式画一棵语法分析树。

语法分析树【重要！】

如何构造一棵合式公式的语法分析树?

举例：一个公式 φ:

$$(((\neg p) \wedge q) \rightarrow (p \wedge (q \vee (\neg r)))) \qquad (1\text{-}4)$$

它的语法分析树为：

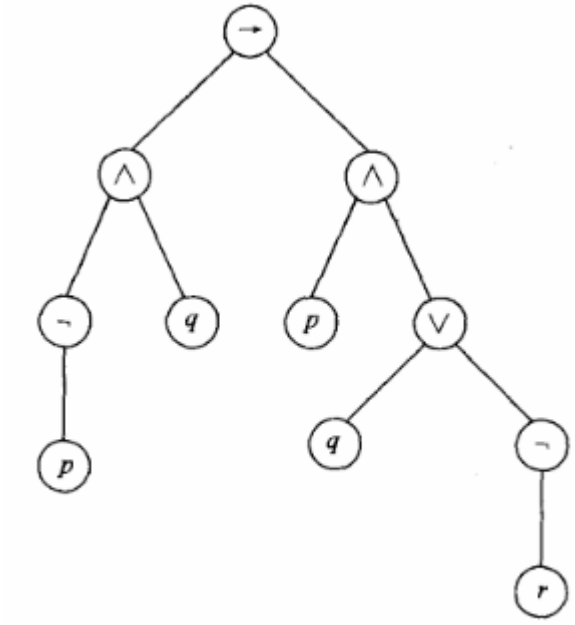

图 1-3　表示一个合式公式的语法分析树

- 如果树的所有叶子都是原子命题，且所有分支结点都是逻辑连接词，那么它对应的公式就是一个合式公式。

2 谓词逻辑

谓词逻辑也被称为一阶逻辑。它比命题逻辑更精确。

命题逻辑：处理“否”、“并”、“或”、“如果”

谓词逻辑：处理“存在...”，“所有...”，“在..中”，“只有...”

在谓词逻辑中，我们要先明白 谓词、变量、量词 这些词的含义是什么。

我们来看一个判断语句：“每个学生都比他的某个老师年轻”

- 谓词：S(andy)中的S是一个谓词，S(andy)表示Andy是一个学生
- 变量：变量是一个实际值的占位符。S(x)中，x是一个变量。
- 量词：引入两个量词 ∀（读作：对所有的）和 ∃（读作：存在某个），它们总是紧跟变量出现，比如 ∀x（对所有x）或 ∃z（存在z）

现在我们用完全符号的形式来写上述语句：

$$\forall x\ (S(x) \rightarrow (\exists\ y(I(y) \wedge Y(x,\ y))))$$

事实上，这种代码是对原句的一种表述。上面形式化语言重新翻译过来即：

对任意*x*，若*x*是一个学生，则存在某个*y*，*y*是一位老师，使得*x*比*y*年轻。

另一个例子：不是所有的鸟都可以飞。

2.2 作为形式语言的谓词逻辑

- 项
- 公式

我们的任务是构建谓词逻辑公式的语法规则。

注意，谓词逻辑公式中涉及两种对象：

1. x、v 这样的变量
2. m(a), g(x, y) 这样的对象

在谓词逻辑中，用来表示对象的表达式称为**项**（term）。

Y(x, m(x)) 是**公式。**

2.2.1 项：

$$t ::= x \mid c \mid f(t, \cdots, t)$$

2.2.2 公式：

$$\phi ::= P(t_1,\ t_2,\ \cdots,\ t_n) \mid (\neg\ \phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\forall x\phi) \mid (\exists x\phi) \quad (2\text{-}2)$$

优先级：

- ¬、∀y、∃y 绑定优先级最高
- 其次是∨ ∧
- 最后是->，它是右结合的。

谓词逻辑的语法分析树：

谓词逻辑公式可以用语法分析树表示。例如，图 2-1 的分析树表示公式 $\forall x((P(x) \rightarrow Q(x)) \wedge S(x, y))$。

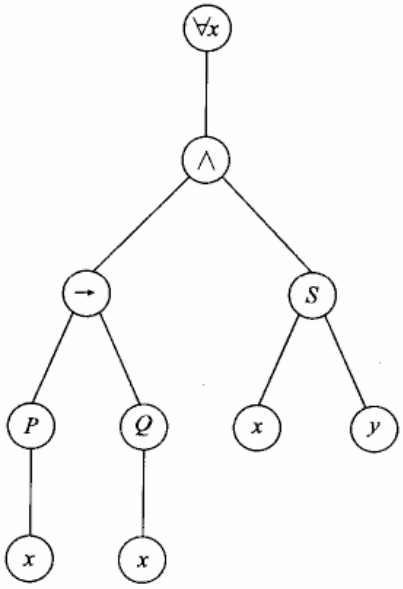

图 2-1　一个谓词逻辑公式的语法分析树

2.2.3 自由/约束变量

2.2.4 代换

定义 2.7　给定变量 x、项 t 和公式 ϕ，定义 $\phi[t/x]$ 为用 t 代替 ϕ 中变量 x 的每个自由出现而得到的公式。

2.3 谓词逻辑的证明

- 相等的证明规则
- 全称量词的证明规则

量词的等价

定理 2.13　设 ϕ 和 ψ 是谓词逻辑公式，具有下面的等价：

1. (a) $\neg \forall x \phi \dashv\vdash \exists x \neg \phi$
   (b) $\neg \exists x \phi \dashv\vdash \forall x \neg \phi$
2. 假设 x 在 ψ 中不是自由的，那么：
   (a) $\forall x \phi \wedge \psi \dashv\vdash \forall x(\phi \wedge \psi)$ ⊖
   (b) $\forall x \phi \vee \psi \dashv\vdash \forall x(\phi \vee \psi)$
   (c) $\exists x \phi \wedge \psi \dashv\vdash \exists x(\phi \wedge \psi)$

(d) $\exists x\,\phi \vee \psi \dashv\vdash \exists x(\phi \vee \psi)$

(e) $\forall x(\psi \rightarrow \phi) \dashv\vdash \psi \rightarrow \forall x\,\phi$

(f) $\exists x(\phi \rightarrow \psi) \dashv\vdash \forall x\,\phi \rightarrow \psi$

(g) $\forall x(\phi \rightarrow \psi) \dashv\vdash \exists x\,\phi \rightarrow \psi$

(h) $\exists x(\psi \rightarrow \phi) \dashv\vdash \psi \rightarrow \exists x\,\phi$

3. (a) $\forall x\,\phi \wedge \forall x\,\psi \dashv\vdash \forall x(\phi \wedge \psi)$

(b) $\exists x\,\phi \vee \exists x\,\psi \dashv\vdash \exists x(\phi \vee \psi)$

4. (a) $\forall x \forall y\,\phi \dashv\vdash \forall y \forall x\,\phi$

(b) $\exists x \exists y \phi \dashv\vdash \exists y \exists x\,\phi$

2.7 软件的围观模型

到目前为止，我们的两个中心问题是：

- 模型检测(model checking)：给定谓词逻辑的一个公式φ和一个相匹配的模型M，确定 M|=φ 是否成立
- 语义推导(semantic entailment)：给定谓词逻辑的一个公式集Γ，确定 Γ|=φ 是否有效

3 通过模型检测进行验证

3.1 验证的冬季

这一章的重要问题是：**逻辑怎样用来验证程序或系统的正确性。**

形式化验证技术有3部分：

- 描述语言：用于系统建模
- 规范语言：用于描述待验证的形式
- 验证方法：用来确定系统描述是否满足规范、

基于证明进行的验证：系统描述是一组公式Γ，而规范是另一个公式φ。验证的方法就是试图找到 Γ|-φ 的证明。

基于模型进行验证：系统由逻辑模型M表示。而规范是另一个公式φ。验证方法是由计算模型M是否满足φ构成（写作 M|=φ ）。

我们在本章关注一种叫做模型检测(model checking)的验证方法。根据分类，模型检测是一种自动的、基于模型的、性质验证处理方法。这种方法预期用于并发的、反应式系统。

模型检测基于时态逻辑。时态逻辑的思想是，在一个模型中公式的真与假不是静态的。时态逻辑的模型包含若干状态，一个公式在某些状态下为真，在其他状态下为假。公式可以随系统的状态演化而改变其真值。

在模型检测中，模型M是一个迁移系统（transition systems），而性质φ是时态逻辑公式。为了验证一个系统满足一个性质，我们必须做3件事：

- 用Promela语言对系统进行建模，得到模型M
- 用Promela语言对性质进行编码，产生一个时态逻辑公式φ
- 以M和φ作为输入，运行模型检测器。

这里的Promela语言可以换成满足条件的其他。

- 线性时间逻辑（LTL）把时间看成路径的集合，这里的路径是时间的一个序列。
- 分支时间逻辑（CTL）把时间表示为树，以当前时间为根向未来分叉。

本章我们研究一种时间是线性的逻辑，称为**线性时间逻辑**（Linear-time Temporal Logic，LTL）以及另一种时间是分支的逻辑，即**计算树逻辑**（Computation Tree Logic，CTL）。

模型检测就是对问题 `M,s|=ϕ` 是否成立计算答案的过程。

这里φ是LTL或CTL中的一个公式，M是系统的一个模型，s是系统的一个状态，|=是满足关系。

3.2 线性时间逻辑（LTL）

我们固定一个原子公式集合Atoms（原子公式 p, q, r, ...），它们代表了系统可能成立的原子事实。

3.2.1 LTL的语法（p131）

定义 3.1 线性时态逻辑(LTL)有下列用 Backus Naur 范式给出的语法：

$$\phi ::= \top \mid \bot \mid p \mid (\neg \phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi)$$
$$\mid (X\ \phi) \mid (F\ \phi) \mid (G\ \phi) \mid (\phi\ U\ \phi) \mid (\phi\ W\ \phi) \mid (\phi\ R\ \phi) \qquad (3\text{-}1)$$

其中 p 是取自某原子集 Atoms 的任意命题原子。

连接词 X, F, G, U, R 和 W 称为时态连接词（temporal connectives）

X - 下一个状态

F - 某个未来状态

G - 所有未来状态

U - 直到

R - 释放

W - 弱一直到

这里是LTL公式的一些例子：

- $(((\mathrm{F}\,p)\wedge(\mathrm{G}\,q))\rightarrow(p\ \mathrm{W}\ r))$
- $(\mathrm{F}(p\rightarrow(\mathrm{G}\,r))\vee((\neg q)\mathrm{U}\,p))$，图 3-1 所示的是该公式的语法分析树。

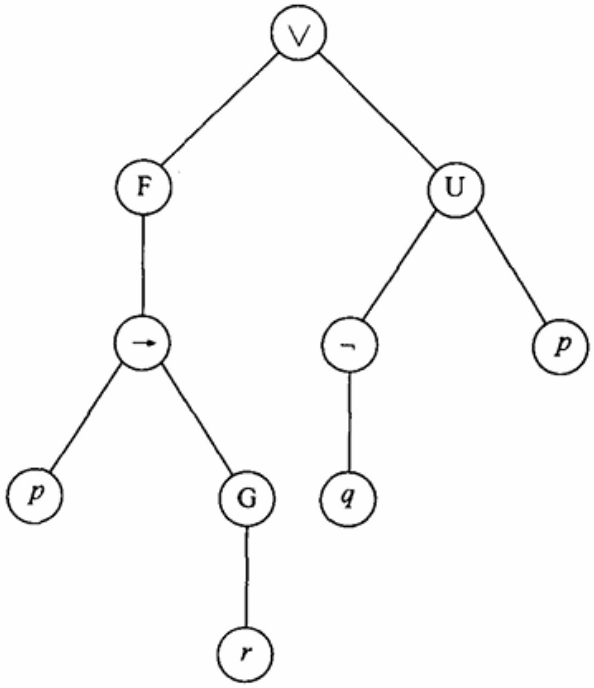

图 3-1 $(\mathrm{F}(p\rightarrow\mathrm{G}\,r)\vee((\neg q)\mathrm{U}\,p))$的语法分析树

- $(p\ \mathrm{W}(q\ \mathrm{W}\ r))$
- $((\mathrm{G}(\mathrm{F}\,p))\rightarrow(\mathrm{F}(q\vee s)))$。

约定3.2 优先级：

- 一元连接词（包括 ¬ 和时态连接词 X, F, G）具有最高绑定级
- 接下来是 U, R, W
- 然后是 ∨ ∧
- 最后是 ->，它是右结合的。

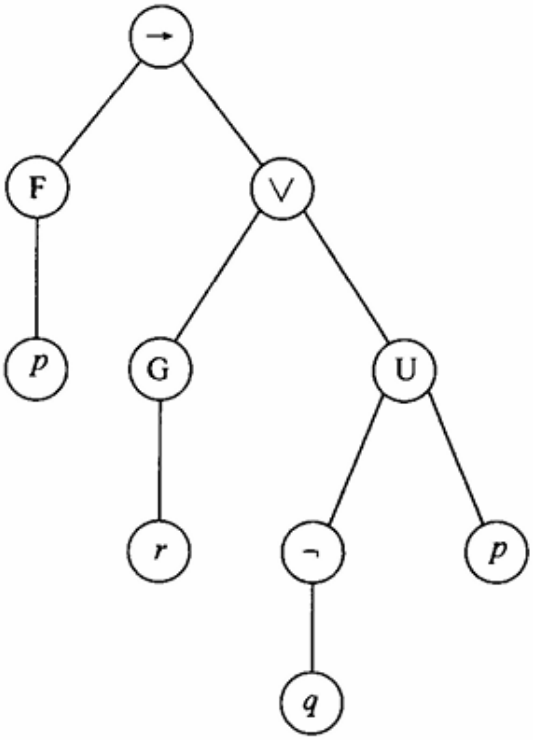

图 3-2　F p→G r ∨¬ q U p 的语法分析树，假定使用约定 3.2 的绑定优先级

补充：程序

LTL Formulas

An LTL formula may contain propositional symbols, boolean operators, temporal operators, and parentheses.

- Propositonal Symbols:

```
true, false
any lowercase string
```

- Boolean operators:

```
!   (negation)
->  (implication)
<-> (equivalence)
&&  (and)
||  (or)
```

- Temporal operators:

```
 G   (always) (Spin syntax : [])
 F   (eventually) (Spin syntax : <>)
 U   (until)
 R   (realease) (Spin syntax : V)
 X   (next)
```

3.2.2 LTL语义

定义3.4 迁移系统M

定义 3.4 一个迁移系统$\mathcal{M}=(S, \rightarrow, L)$是一个状态集合$S$，带有迁移关系$\rightarrow$($S$上的二元关系)，使得每个$s \in S$有某个$s' \in S$，满足$s \rightarrow s'$，以及一个标记函数$L: S \rightarrow \mathcal{P}(\mathrm{Atoms})$。

例子:

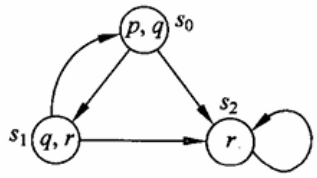

图 3-3 迁移系统$\mathcal{M}=(S, \rightarrow, L)$作为有向图的简明表示。用$l$标记状态$s$当且仅当$l \in L(s)$

定义3.5 路径

定义 3.5 模型$\mathcal{M}=(S, \rightarrow, L)$中的一条路径是$S$中状态的无限序列$s_1, s_2, s_3, \cdots$，对每个$i \geqslant 1$，有$s_i \rightarrow s_{i+1}$。我们将该条路径写为$s_1 \rightarrow s_2 \rightarrow \cdots$。

定义3.6 满足关系|=（路径满足一个LTL公式）【重要！】

定义 3.6 设$\mathcal{M}=(S, \rightarrow, L)$是一个模型，$\pi=s_1 \rightarrow \cdots$是$\mathcal{M}$中的一条路径。$\pi$是否满足一个 LTL 公式，由满足关系$\vDash$定义如下：

1. $\pi \vDash \top$
2. $\pi \nvDash \bot$
3. $\pi \vDash p$ 当且仅当 $p \in L(s_1)$
4. $\pi \vDash \neg \phi$ 当且仅当 $\pi \nvDash \phi$
5. $\pi \vDash \phi_1 \wedge \phi_2$ 当且仅当 $\pi \vDash \phi_1$ 且 $\pi \vDash \phi_2$
6. $\pi \vDash \phi_1 \vee \phi_2$ 当且仅当 $\pi \vDash \phi_1$ 或 $\pi \vDash \phi_2$
7. $\pi \vDash \phi_1 \rightarrow \phi_2$ 当且仅当只要 $\pi \vDash \phi_1$ 就有 $\pi \vDash \phi_2$
8. $\pi \vDash \mathrm{X}\, \phi$ 当且仅当 $\pi^2 \vDash \phi$
9. $\pi \vDash \mathrm{G}\, \phi$ 当且仅当对所有 $i \geqslant 1$，$\pi^i \vDash \phi$
10. $\pi \vDash \mathrm{F}\, \phi$ 当且仅当存在某个 $i \geqslant 1$，使得 $\pi^i \vDash \phi$
11. $\pi \vDash \phi\, \mathrm{U}\, \psi$ 当且仅当存在某个 $i \geqslant 1$，使得 $\pi^i \vDash \psi$ 并且对所有 $j=1, \cdots, i-1$，有 $\pi^j \vDash \phi$
12. $\pi \vDash \phi\, \mathrm{W}\, \psi$ 当且仅当存在某个 $i \geqslant 1$，使得 $\pi^i \vDash \psi$ 且对于所有的 $j=1, \cdots, i-1$，有 $\pi^j \vDash \phi$，或者对所有 $k \geqslant 1$，有 $\pi^k \vDash \phi$
13. $\pi \vDash \phi\, \mathrm{R}\, \psi$ 当且仅当或者存在某个 $i \geqslant 1$，使得 $\pi^i \vDash \phi$，并且对所有 $j=1, \cdots, i$，有 $\pi^j \vDash \psi$；或者对所有 $k \geqslant 1$，有 $\pi^k \vDash \psi$。

语句 1 和 2 反映⊤总为真、⊥总是假的事实。语句 3 ~ 7 与我们在命题逻辑中所看到的对应的句子类似。语句 8 从路径中移除第一个状态，为了创建一条始于“下一个”(第二个)状态的路径。

注意语句 3 意味着原子沿着所考虑路径的第一个状态下被赋值。然而，这并不意味着出现在 LTL 公式中的所有原子都指向路径的第一个状态。如果它们处于一个时态连接词的范围中(例如，在 $\mathrm{G}(p \rightarrow \mathrm{X}\, q)$中)，可满足性计算涉及到所考虑路径的后缀，而原子则参考那些后缀的第一个状态。

对于U的定义，一定要特别留意!

现在来考察处理二元时态连接词的语句 11 ~ 13。U(代表 Until)是其中最常见的一个。公式 ϕ_1 U ϕ_2 在一条路径上成立，如果 ϕ_1 连续地成立直到 ϕ_2 成立。此外，ϕ_1 U ϕ_2 实际上要求 ϕ_2 在某个未来状态确实成立。见图 3-6 中的解释：沿着所示的路径，s_3 到 s_9 的每个状态都满足 p U q，但 s_0 到 s_2 不满足。

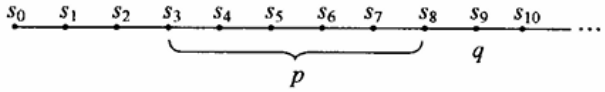

图 3-6　LTL 语义中 Until(直到)含义的解释。假设 p 在(且只在)s_3，s_4，s_5，s_6，s_7，s_8 点满足，q 在(且只在)s_9 点满足，那么沿着所示路径只有 s_3 到 s_9 满足 p U q

W和R暂时不讨论了。

注意！在语句9~13中，未来包括了当前。这意味着，当我们在说"所有未来状态"时，我们将当前状态作为一种未来状态包括了进来。这样做是一种约定。

定义3.8 `M,s|=ϕ`　　-------->【重要！】

定义 3.8　设 $\mathcal{M}=(S, \rightarrow, L)$ 是一个模型，$s \in S$，且 ϕ 是一个 LTL 公式。如果对 $\mathcal{M}$ 的每条始于 s 的执行路径 π，都有 $\pi \vDash \phi$，我们记为 $\mathcal{M}, s \vDash \phi$。

例子：

我们对图3-3的迁移系统做MC：

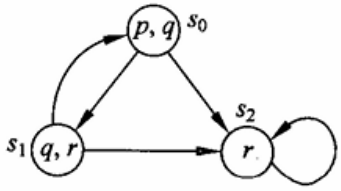

图 3-3　迁移系统 $\mathcal{M}=(S, \rightarrow, L)$ 作为有向图的简明表示。用 l 标记状态 s 当且仅当 $l \in L(s)$

把系统M展开成一棵语法树，如下图所示：

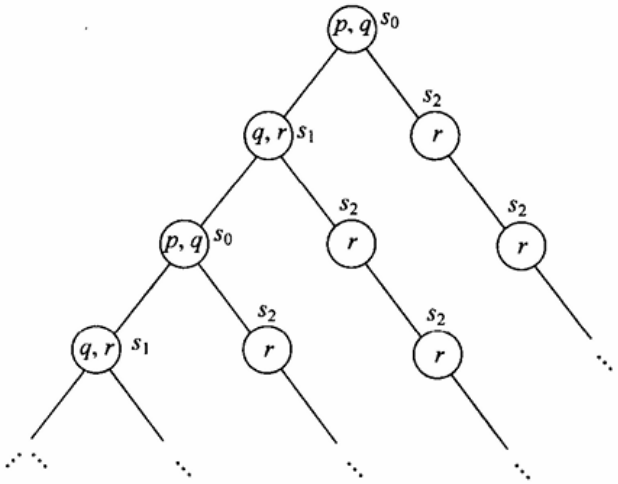

图 3-5　将图 3-3 的系统展开成一个从特定状态开始的所有计算路径的无限树

1. $\mathcal{M}, s_0 \vDash p \wedge q$ 成立，因为原子符号 p 和 q 含在结点 s_0 中：对每条以 s_0 开始的路径 π，$\pi \vDash p \wedge q$。
2. $\mathcal{M}, s_0 \vDash \neg r$ 成立，因为原子符号 r 不含在结点 s_0 中。
3. 由定义，$\mathcal{M}, s_0 \vDash \top$ 成立。
4. $\mathcal{M}, s_0 \vDash \mathrm{X}\, r$ 成立，因为从 s_0 开始的所有路径以 s_1 或以 s_2 作为其下一状态，而每个状态都满足 r。

5. $\mathcal{M}, s_0 \models \mathrm{X}(q \wedge r)$不成立，因为在图 3-5 中，最右边的计算路径 $s_0 \to s_2 \to s_2 \to s_2 \to \cdots$的第二个结点 s_2 包含 r，却不包含 q。

6. $\mathcal{M}, s_0 \models \mathrm{G} \neg(p \wedge r)$成立，因为始于 s_0 的所有计算路径都满足 $\mathrm{G} \neg(p \wedge r)$，即沿该路径的每个状态下都满足$\neg(p \wedge r)$。注意在一个状态下 $\mathrm{G}\,\phi$ 成立，当且仅当从给定状态可达的所有状态下，ϕ 成立。

7. 基于同样的原因，$\mathcal{M}, s_2 \models \mathrm{G}\,r$ 成立(注意 s_0 由 s_2 代替)。

8. 对$\mathcal{M}$的任何状态 s，有$\mathcal{M}, s \models \mathrm{F}(\neg q \wedge r) \to \mathrm{FG}\,r$。这说明：如果始于 s 的任何路径 π 到达一个满足 $(\neg q \wedge r)$的状态，那么路径 π 满足 $\mathrm{FG}\,r$。事实上这是真的，如果该路径有一个状态满足$(\neg q \wedge r)$，那么(因为该状态必是 s_2)该路径一定满足 $\mathrm{FG}\,r$。注意 $\mathrm{FG}\,r$ 关于路径所说的内容：最终，你会连续地得到 r。

9. 公式 $\mathrm{GF}p$ 表示沿着问题中的路径，p 无限多次地发生。直观地说，无论你沿着路径走多远(这是 G 的部分)，你都会发现仍有 p 在你前面(这是 F 的部分)。例如，路径 $s_0 \to s_1 \to s_0 \to s_1 \to \cdots$满足 $\mathrm{GF}\,p$，但路径 $s_0 \to s_2 \to s_2 \to s_2 \to \cdots$不满足。

10. 在我们的模型中，如果一条始于 s_0 的路径上有无限个 p，则该路径必定是 $s_0 \to s_1 \to s_0 \to s_1 \to \cdots$，而且此时该路径上也有无限多个 r，故$\mathcal{M}, s_0 \models \mathrm{GF}\,p \to \mathrm{GF}\,r$。但换一下位置就不然了！$\mathcal{M}, s_0 \models \mathrm{GF}\,r \to \mathrm{GF}\,p$ 不成立，因为可以找到始于 s_0 的路径有无限多个 r，但只有一个 p。

3.2.4 LTL公式之间的重要等价

定义3.9 LTL公式之间的等价　　----------> 也许重要，有题型

定义 3.9　我们说两个 LTL 公式 ϕ 和 ψ 是语义等价的(或简单说是等价的)并写为 $\phi \equiv \psi$，如果对所有模型$\mathcal{M}$以及$\mathcal{M}$中的所有路径 π：$\pi \models \phi$ 当且仅当 $\pi \models \psi$。

等价意味着他们之间可以互换。

在命题逻辑中，∨ ∧是互相对偶的。

在命题逻辑中已经看到∧和∨是互相对偶的，意思是如果你在∧前放一个¬，它就变成∨，反之也成立：

$$\neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi \qquad \neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi$$

在LTL中，F和G 是互相对偶的，X与其自身对偶：

$$\neg\,\mathrm{G}\,\phi \equiv \mathrm{F}\neg\phi \qquad \neg\,\mathrm{F}\,\phi \equiv \mathrm{G}\neg\phi \qquad \neg\,\mathrm{X}\,\phi \equiv \mathrm{X}\neg\phi$$

F 关于∨，G 关于∧的分配律也如此，即：

$$\mathrm{F}(\phi \vee \psi) \equiv \mathrm{F}\,\phi \vee \mathrm{F}\,\psi$$
$$\mathrm{G}(\phi \wedge \psi) \equiv \mathrm{G}\,\phi \wedge \mathrm{G}\,\psi$$

这里是 LTL 中的另两个等价关系：

$$\mathrm{F}\,\phi \equiv \top\,\mathrm{U}\,\phi \qquad \mathrm{G}\,\phi \equiv \bot\,\mathrm{R}\,\phi$$

第一个等价揭示了 Until 状态的两件事情：第二个公式 ϕ 必须变为真；且直到那时为止，第一个公式⊤必须成立。因此，如果我们对第一个公式“不加约束”，它就会转而要求第二个公式成立，这正是 F 所要求的。(公式⊤代表“不加约束”(no constraint)。如果要求让⊤成立，不需做任何事，它强调的就是无约束。在同样意义下，⊥表示“所有约束”(every constraint)。如果要求让⊥成立，必须满足存在的所有约束，而那是不可能的。)

3.4 分支时间逻辑（CTL）

在前面各节中对LTL（线性时态逻辑）的分析中，我们注意到LTL公式是在路径上赋值的。**我们定义系统的一个状态满足一个LTL公式，如果由给定状态出发的所有路径都满足它。** 于是，LTL隐含着对所有路径做全称量词限定。因此，断言一条路径存在的性质不能用LTL表达。

分支时间逻辑通过使用**路径变量**解决了这个问题。CTL中，除了有LTL中的时态算子 U, F, G, X外，还有路径量词A和E。分别表达“对所有路径”和“对一条路径”。例如：

- 存在一个可达状态满足 q：这可以写为 EF q。
- 对所有满足 p 的可达状态，可以连续地保持 p，直到到达一个满足 q 的状态，这可以写为 AG(p→E[p U q])。
- 只要满足 p 的状态是可达的，系统可以永远连续不断呈现 q：AG(p→EG q)。
- 存在一个可达状态，由其出发的所有可达状态都满足 p：EF AG p。

3.4.1 CTL的语法 (p152)

定义 3.12 CTL公式的形式化定义

计算树逻辑(或简称 CTL)是一种分支时间逻辑，即它的时间模型是一个树状结构，其中未来是不确定的。未来有不同的路径，其中的任何一个都可能是现实的“实际”路径。

和以前一样，我们固定一个的原子公式/描述的集合(例如 p，q，r，…或 p_1，p_2，…)。

定义 3.12 如 LTL 一样，通过 Backus-Naur 范式归纳定义 CTL 公式：

$$\phi ::= \bot \mid \top \mid p \mid (\neg \phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid \mathrm{AX}\,\phi \mid \mathrm{EX}\,\phi \mid \mathrm{AF}\,\phi \mid \mathrm{EF}\,\phi \mid \mathrm{AG}\,\phi \mid \mathrm{EG}\,\phi \mid \mathrm{A}[\phi\ \mathrm{U}\ \phi] \mid \mathrm{E}[\phi\ \mathrm{U}\ \phi]$$

此处 p 取遍原子公式的集合。

注意，每个 CTL 时态连接词都是一对符号。对中的第一个是 A 或 E。A 是“沿所有路径”(无一例外)，E 的含义是“沿至少(存在)一条路径”(可能)。符号对中的第二个符号是 X、F、G 或 U，含义分别为“下一状态”，“某个未来状态”，“所有未来状态(全局)”和“直到”。例如，$\mathrm{E}[\phi_1\ \mathrm{U}\ \phi_2]$中的符号对是 EU。在 CTL 中，像 EU 这样的符号对是不可分的。注意 AU 和 EU 是二元的。符号 X，F，G 和 U 在前面没有 A 或 E 的情况下不能单独出现。类似地，每个 A 或 E 必须伴随着 X，F，G 或 U 之一出现。

约定 3.13 CTL连接词的优先级

约定 3.13 对 CTL 连接词，假定与命题逻辑和谓词逻辑中类似的绑定优先级。一元连接词(包括¬和时态连接词 AG，EG，AF，EF，AX 和 EX)有最紧密的绑定；其次是∧和∨；然后是→，AU 以及 EU。

来看看哪些是合法的CTL公式，哪些不是：

非合式 CTL 公式的例子。假设 p，q 和 r 是原子公式。下面是一些合式 CTL 公式：

- AG(q→EG r)，注意它与 AG q→EG r 不一样，因为根据约定 3.13，后一个公式指

(AG q)→(EG r)

- EF E[r U q]
- A[p U EF r]
- EF EG p→AF r，还要注意它的括号绑定为(EF EG p)→AF r，不是 EF(EG p→AF r)或 EF EG(p→AF r)
- A[p_1 U A[p_2 U p_3]]
- E[A[p_1 U p_2]U p_3]
- AG(p→A[p U(¬p ∧ A[¬p U q]))]。

值得花一些时间看看语法规则是如何允许构造这些公式的。下面的式子不是合式公式：

- EF G r
- A ¬ G ¬ p
- F[r U q]
- EF(r U q)
- AEF r
- A[(r U q)∧(p U r)]。

特别值得理解的是为什么语法规则不允许构造这些公式。以 EF(r U q)为例，这个字符串的问题在于 U 只能与 A 或 E 配对出现，E 又是与 F 成对出现的。为了将这个串改成一个合式 CTL 公式，只能写 EF E[r U q]或 EF A[r U q]。

注意，当在 A 或 E 后面配对的算子是 U 时，使用方括号。这样做并没有什么特别的理由。可以用通常的圆括号代替。然而，这常有助于人们阅读公式(因为我们能更容易地找到相应的关闭括号的位置)。使用方括号的另一个原因是 SMV 一直坚持这样做。

A[(r U q)∧(p U r)]不是合式公式的原因是语法不允许将布尔连接词(例如∧)直接放在 A[ ]或 E[ ]中。A 或 E 的出现必须紧跟着 G，F，X 或 U 之一出现。如果 U 跟着它们出现，一定是 A[ϕ U ψ]的形式。现在 ϕ 和 ψ 可以包含∧，因为它们是任意公式。因此，A[(p ∧ q)U(¬ r→q)]是一个合式公式。

注意 AU 和 EU 是混和使用中缀和前缀记号的二元连接词。按严格的中缀形式，应该写 ϕ_1 AU ϕ_2，而按严格的前缀形式，应该写 AU(ϕ_1，ϕ_2)。

就任何形式语言而言，如前两章，画出合式公式的语法分析树是有用的。A[AX ¬ p U E[EX(p ∧ q) U ¬ p]]的语法分析树如图 3-18 所示。

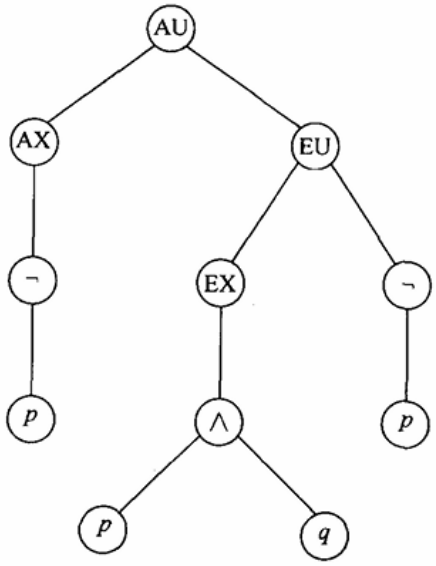

图 3-18　一个无中缀记号的 CTL

3.4.2 计算书逻辑的语义

CTL 公式在迁移系统(定义 3.4)上进行解释。设$\mathcal{M}$ = (S，→，L)是一个模型，$s \in S$，ϕ

是一个 CTL 公式。$\mathcal{M}, s \vDash \phi$ 是否成立的定义是就 ϕ 的结构进行递归，可以大致理解如下：

- 若 ϕ 是原子的，满足关系由 L 确定。
- 若 ϕ 的顶级连接词(即出现在 ϕ 的语法分析树中最顶层的连接词)是一个布尔连接词($\wedge$，$\vee$，$\neg$，$\top$等)，则满足问题由普通的真值表定义以及 ϕ 的下一步递归来回答。
- 若顶级连接词是一个以 A 开始的算子，则满足关系成立，如果从 s 出发的所有路径均满足移去符号 A 后所得到的“LTL 公式”。
- 类似地，若顶级连接词开始于 E，则满足关系成立，如果从 s 出发的某个路径满足移去符号 E 后所得到的“LTL 公式”。

定义3.15 M,s|=φ 的形式定义 ---------> 【重要！】

定义 3.15 设 $\mathcal{M}=(S, \rightarrow, L)$ 是关于 CTL 的一个模型，s 属于 S，ϕ 是一个 CTL 公式。关系 $\mathcal{M}, s \vDash \phi$ 由对 ϕ 做结构归纳来定义：

1. $\mathcal{M}, s \vDash \top$ 且 $\mathcal{M}, s \nvDash \bot$
2. $\mathcal{M}, s \vDash p$ 当且仅当 $p \in L(s)$
3. $\mathcal{M}, s \vDash \neg \phi$ 当且仅当 $\mathcal{M}, s \nvDash \phi$
4. $\mathcal{M}, s \vDash \phi_1 \wedge \phi_2$ 当且仅当 $\mathcal{M}, s \vDash \phi_1$ 且 $\mathcal{M}, s \vDash \phi_2$
5. $\mathcal{M}, s \vDash \phi_1 \vee \phi_2$ 当且仅当 $\mathcal{M}, s \vDash \phi_1$ 或者 $\mathcal{M}, s \vDash \phi_2$
6. $\mathcal{M}, s \vDash \phi_1 \rightarrow \phi_2$ 当且仅当 $\mathcal{M}, s \nvDash \phi_1$ 或者 $\mathcal{M}, s \vDash \phi_2$
7. $\mathcal{M}, s \vDash \mathrm{AX}\,\phi$ 当且仅当对所有使得 $s \rightarrow s_1$ 的 s_1，有 $\mathcal{M}, s_1 \vDash \phi$。于是，AX 含义为“在所有下一状态。”
8. $\mathcal{M}, s \vDash \mathrm{EX}\,\phi$ 当且仅当对某个使得 $s \rightarrow s_1$ 的 s_1，有 $\mathcal{M}, s_1 \vDash \phi$。于是，EX 含义为“在某个下一状态”。E 与 A 对偶，与谓词逻辑中 $\exists$ 与 $\forall$ 对偶的情况完全一样。
9. $\mathcal{M}, s \vDash \mathrm{AG}\,\phi$ 成立当且仅当对于所有满足 $s_1 = s$ 的路径 $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$，及沿该路径上的所有 s_i，有 $\mathcal{M}, s_i \vDash \phi$。方便记忆：对所有以 s 开始的计算路径，性质 ϕ 全局地成立。注意：“沿该路径”包括路径的初始状态 s。
10. $\mathcal{M}, s \vDash \mathrm{EG}\,\phi$ 成立当且仅当存在一条满足 $s_1 = s$ 的路径 $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$，及沿该路径上的所有 s_i，有 $\mathcal{M}, s_i \vDash \phi$。方便记忆：存在一条以 s 开始的路径，使得沿着该路径，ϕ 全局地成立。
11. $\mathcal{M}, s \vDash \mathrm{AF}\,\phi$ 成立当且仅当对于所有满足 $s_1 = s$ 的路径 $s_1 \rightarrow s_2 \rightarrow \cdots$，有某个 s_i，使得 $\mathcal{M}, s_i \vDash \phi$。方便记忆：对于所有以 s 开始的计算路径，存在某个未来状态使 ϕ 成立。
12. $\mathcal{M}, s \vDash \mathrm{EF}\,\phi$ 成立当且仅当存在一条满足 $s_1 = s$ 的路径 $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$，并且沿该路径存在某个 s_i，使得 $\mathcal{M}, s_i \vDash \phi$。方便记忆：存在一条以 s 开始的计算路径，使得 ϕ 在某个未来状态下成立。
13. $\mathcal{M}, s \vDash \mathrm{A}[\phi_1\ \mathrm{U}\ \phi_2]$ 成立当且仅当对于所有满足 $s_1 = s$ 的路径 $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$，该路径满足 $\phi_1\ \mathrm{U}\ \phi_2$，即沿着该路径存在某个 s_i，使得 $\mathcal{M}, s_i \vDash \phi_2$，且对于每个 $j < i$，有 $\mathcal{M}, s_j \vDash \phi_1$。方便记忆：所有以 s 开始的计算路径满足 ϕ_1，直到 ϕ_2 在其上成立。
14. $\mathcal{M}, s \vDash \mathrm{E}[\phi_1\ \mathrm{U}\ \phi_2]$ 成立当且仅当存在一条满足 $s_1 = s$ 的路径 $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$，且该路径满足 $\phi_1\ \mathrm{U}\ \phi_2$，如在 13 中说明的那样。方便记忆：存在一条以 s 开始的计算路径满足 ϕ_1，直到 ϕ_2 在其上成立。

上面的语句 9 ~ 14 均参考了模型中的计算路径。因此，从给定状态 s 出发的所有可能的计算路径可视化是非常有用的，这可以将迁移系统展开得到一个无限计算树，因而得名“计算树逻辑”。图 3-19 ~ 图 3-22 的图形模式化地显示了初始状态分别满足公式 EF ϕ，EG ϕ，AG ϕ 和 AF ϕ 的系统。当然，可以在这些图形的任何一个添加更多的 ϕ，并且仍保持满足关系，尽管对 AG 已经没什么可添加的了。这些图形解释了满足公式的“最小”方式。

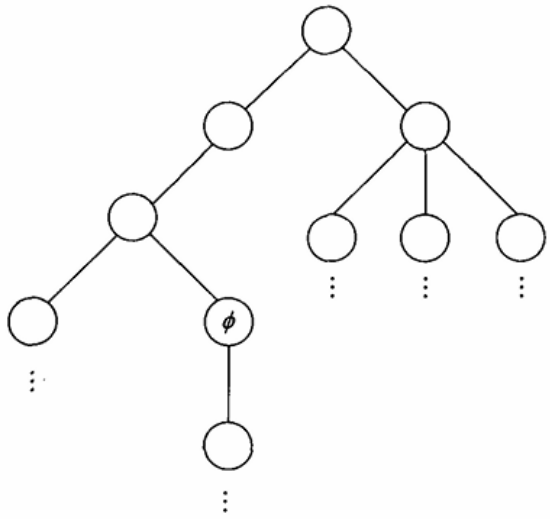

图 3-19　初始状态满足 EF ϕ 的一个系统

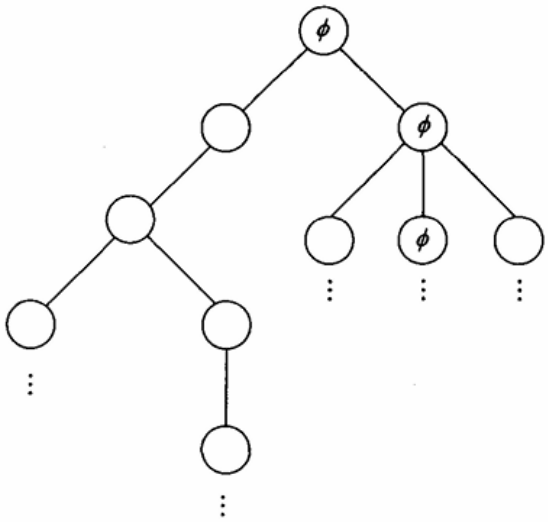

图 3-20　初始状态满足 EG ϕ 的一个系统

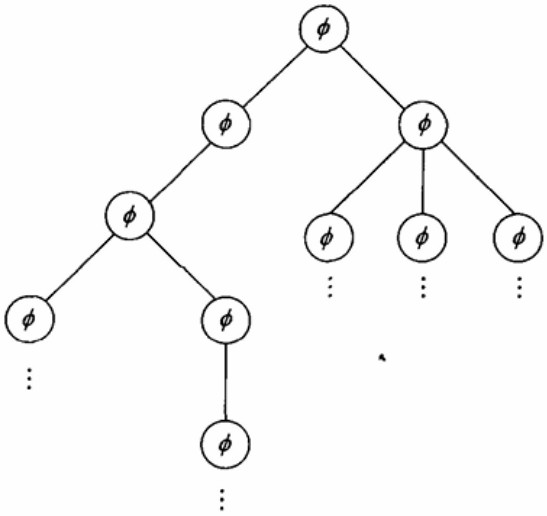

图 3-21　初始状态满足 AG ϕ 的一个系统

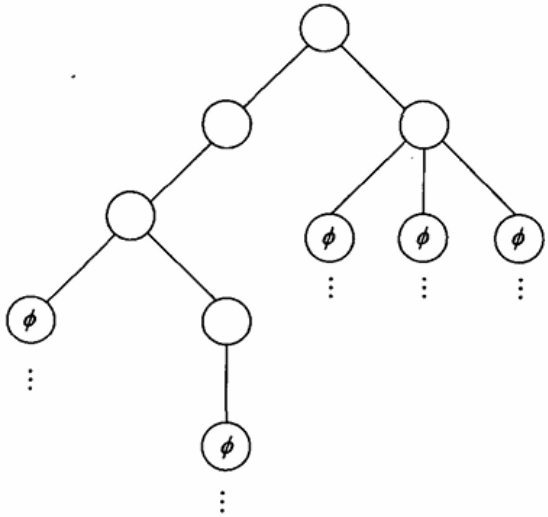

图 3-22　初始状态满足 AF ϕ 的一个系统

现在来看一个例子，图3-5为例看看这个系统的一些公式检测：

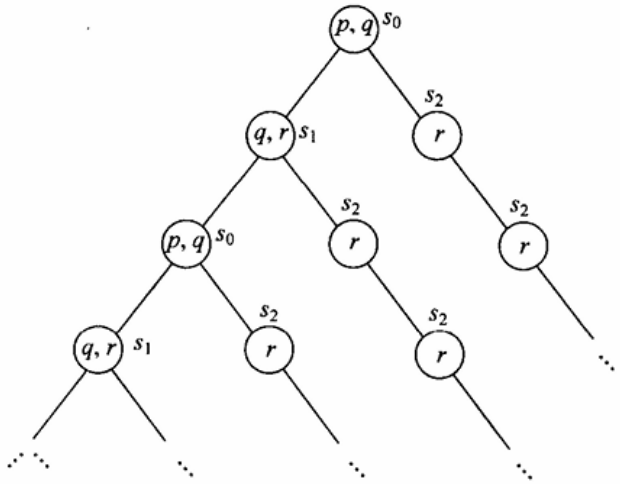

图 3-5　将图 3-3 的系统展开成一个从特定状态开始的所有计算路径的无限树

1. $\mathcal{M}, s_0 \models p \wedge q$ 成立，因为原子符号 p 和 q 包含在 s_0 结点中。

2. $\mathcal{M}, s_0 \models \neg r$ 成立，因为原子符号 r 不包含在 s_0 结点中。

3. 由定义，$\mathcal{M}, s_0 \models \top$ 成立。

4. $\mathcal{M}, s_0 \models \mathrm{EX}(q \wedge r)$ 成立，因为有图 3-5 中最左边的计算路径 $s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow \cdots$，它的第二个结点 s_1 包含 q 和 r。

5. $\mathcal{M}, s_0 \models \neg\, \mathrm{AX}(q \wedge r)$ 成立，因为图 3-5 中有最右边的计算路径 $s_0 \rightarrow s_2 \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots$，它的第二个结点 s_2 仅包含 r，不包含 q。

6. $\mathcal{M}, s_0 \models \neg\, \mathrm{EF}(p \wedge r)$ 成立，因为不存在以 s_0 开始的计算路径能够达到使 $p \wedge r$ 成立的一个状态。这不过是因为在这个系统中不存在使 p 和 r 同时成立的状态。

7. $\mathcal{M}, s_2 \models \mathrm{EG}\, r$ 成立，因为存在一条以 s_2 开始的计算路径 $s_2 \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots$，使得 r 对于该路径上的所有未来状态都成立；这是以 s_2 开始的唯一计算路径，因此 $\mathcal{M}, s_2 \models \mathrm{AG}\, r$ 也成立。

8. $\mathcal{M}, s_0 \models \mathrm{AF}\, r$ 成立，因为对以 s_0 开始的所有计算路径，系统会达到一个使 r 成立的状态（s_1 或 s_2）。

9. $\mathcal{M}, s_0 \models \mathrm{E}[(p \wedge q)\,\mathrm{U}\, r]$ 成立，因为有图 3-5 中最右边的计算路径 $s_0 \rightarrow s_2 \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots$，其第二个结点 $s_2 (i=1)$ 满足 r，但它前面的所有结点都（只有 $j=0$，即结点 s_0）满足 $p \wedge q$。

10. $\mathcal{M}, s_0 \models \mathrm{A}[p\,\mathrm{U}\, r]$ 成立，因为 p 在 s_0 成立，且 r 在 s_0 的任何可能后继状态都成立，故 $p\,\mathrm{U}\, r$ 对以 s_0 开始的所有计算路径都为真（我们可以不依赖于路径而选取 $i=1$）。

11. $\mathcal{M}, s_0 \models \mathrm{AG}(p \vee q \vee r \rightarrow \mathrm{EF}\ \mathrm{EG}\, r)$ 成立，因为在从 s_0 可达且满足 $p \vee q \vee r$ 的所有状态（此时就是所有状态）下，系统可以达到一个满足 $\mathrm{EG}\, r$ 的状态（此处为状态 s_2）。

3.4.4 CTL公式间的重要等价

定义 3.16 两个 CTL 公式 ϕ 和 ψ 称为语义等价的，如果任何模型中的任何状态，只要满足其中一个，就满足另一个。我们记为 $\phi \equiv \psi$。

我们已经注意到 A 是关于路径的全称量词，E 是相应的存在量词。此外，G 和 F 也是沿一个特殊路径上状态的全称和存在量词。鉴于这些事实，发现它们之间存在德·摩根法则就不足为奇了：

$$\begin{aligned} \neg\, \mathrm{AF}\, \phi &\equiv \mathrm{EG} \neg \phi \\ \neg\, \mathrm{EF}\, \phi &\equiv \mathrm{AG} \neg \phi \\ \neg\, \mathrm{AX}\, \phi &\equiv \mathrm{EX} \neg \phi \end{aligned} \tag{3-6}$$

我们还有等价

$$\mathrm{AF}\, \phi \equiv \mathrm{A}[\top\, \mathrm{U}\, \phi] \qquad \mathrm{EF}\, \phi \equiv \mathrm{E}[\top\, \mathrm{U}\, \phi]$$

这与 LTL 中相应的等价是类似的。

CTL中其他一些值得注意的等价：

$$\begin{aligned} \mathrm{AG}\, \phi &\equiv \phi \wedge \mathrm{AX}\, \mathrm{AG}\, \phi \\ \mathrm{EG}\, \phi &\equiv \phi \wedge \mathrm{EX}\, \mathrm{EG}\, \phi \\ \mathrm{AF}\, \phi &\equiv \phi \vee \mathrm{AX}\, \mathrm{AF}\, \phi \\ \mathrm{EF}\, \phi &\equiv \phi \vee \mathrm{EX}\, \mathrm{EF}\, \phi \\ \mathrm{A}[\phi\, \mathrm{U}\, \psi] &\equiv \psi \vee (\phi \wedge \mathrm{AX}\, \mathrm{A}[\phi\, \mathrm{U}\, \psi]) \\ \mathrm{E}[\phi\, \mathrm{U}\, \psi] &\equiv \psi \vee (\phi \wedge \mathrm{EX}\, \mathrm{E}[\phi\, \mathrm{U}\, \psi]) \end{aligned}$$

3.4.5 CTL连接词的适当集

- AU，EU和EX可以形成一个适当集合
- EG，EU，EX也可以形成一个适当集合

定理3.17 CTL中的适当集

定理 3.17 CTL 中的一个时态连接词集合是适当的，当且仅当它至少包含{AX，EX}中之一、{EG，AF，AU}中之一以及 EU。

3.5 CTL*

CTL* 是将 LTL 和 CTL 的表达能力结合并除去 CTL 对每个时态算子(X，U，F，G)必须与唯一路径量词(A，E)伴随使用的约束而得到的一种逻辑。它允许诸如以下形式的公式：

- $A[(p\ U\ r) \vee (q\ U\ r)]$：沿所有路径，要么 p 是真的直到 r，要么 q 是真的直到 r。
- $A[X\ p \vee XX\ p]$：沿所有路径，p 在下一个状态或者再下一个状态是真的，但只有一种情况成立。
- $E[G\ F\ p]$：存在一条路径，沿着它 p 无限多次为真。

这些公式并不分别等价于 $A[(p \vee q)\ U\ r]$，$AX\ p \vee AX\ AX\ p$ 和 $EG\ EF\ p$。可以证明其中的第一个可写成(非常长的)CTL 公式。第二和第三个则没有与之等价的 CTL 公式。

CTL* 的语法涉及两类公式：

- 状态公式(state formulas)，用状态来赋值：

$$\phi ::= \top \mid p \mid (\neg \phi) \mid (\phi \wedge \phi) \mid A[\alpha] \mid E[\alpha]$$

此处 p 是任何原子公式，而 α 是任意路径公式；以及

- 路径公式(path formulas)，沿着路径赋值：

$$\alpha ::= \phi \mid (\neg \alpha) \mid (\alpha \wedge \alpha) \mid (\alpha\ U\ \alpha) \mid (G\ \alpha) \mid (F\ \alpha) \mid (X\ \alpha)$$

此处 ϕ 是任何状态公式。这是使用互递归(mutually recursive)归纳定义的例子：每类公式的定义依赖于另一类的定义，其基本情况是 p 和 $\top$。

3.6 模型检测算法

3.6.1 CTL模型检测算法（标记算法）

对一个状态展开成无限树做模型检测更容易，但计算机不能将迁移系统展开成无限树。我们需要对有限的数据进行检测。在φ不满足的情况下，产生一条实际路径，来证明M不能满足φ。

我们把“检测模型M是否满足公式φ”的问题记为：

$$\mathcal{M},\ s_0 \stackrel{?}{\models} \phi$$

模型检测机的实现：

1. 可以将模型M，公式φ，状态s0作为输入，然后期待一个“yes”或“no”的输出
2. 可以只输入M和φ，输出是模型M的满足φ的所有状态s

实现2就能实现1，因为只需要判断s0在不在s中就行了。

我们使用【标记算法】来实现模型检测：

输入：一个 CTL 模型 $\mathcal{M} = (S, \rightarrow, L)$ 和一个 CTL 公式 ϕ。
输出：满足 ϕ 的 $\mathcal{M}$ 的状态集合。

首先，把 ϕ 变成 TRANSLATE(ϕ)的输出，即用本章前面给出的等价将 ϕ 用连接词 AF，EU，EX，$\wedge$，$\neg$和$\perp$表示。其次，从 ϕ 的最小子公式开始，用 ϕ 的被满足的子公式来标记 $\mathcal{M}$的状态，并由里向外逐步扩展 ϕ。

假定 ψ 是 ϕ 的一个子公式，且满足 ψ 的所有直接子公式的状态都已经被标记。我们通过情况分析来确定哪些状态用 ψ 标记。如果 ψ 是

- $\perp$：则没有状态用$\perp$标记。
- p：若 $p \in L(s)$，则用 p 标记 s。
- $\psi_1 \wedge \psi_2$：若 s 已用 ψ_1 和 ψ_2 标记，则 s 用 $\psi_1 \wedge \psi_2$ 标记。
- $\neg \psi_1$：若 s 还没有用 ψ_1 标记，则用 $\neg \psi_1$ 标记 s。
- AF ψ_1：
  - 若任何状态 s 用 ψ_1 标记了，则用 AF ψ_1 标记它。
  - 重复：用 AF ψ_1 标记任何状态，如果所有后继状态用 AF ψ_1 标记，直到不再发生改变。图 3-24 说明了这个步骤。

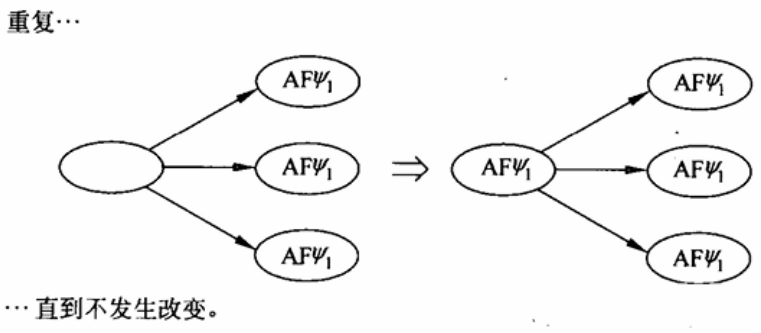

图 3-24　用形如 AF ψ_1 的子公式标记状态过程的迭代步骤

- E[ψ_1 U ψ_2]：
  - 若任何状态 s 用 ψ_2 标记，则用 E[ψ_1 U ψ_2]标记它。
  - 重复：用 E[ψ_1 U ψ_2]标记任何状态，如果该状态用 ψ_1 标记，并且其后继状态中至少有一个用 E[ψ_1 U ψ_2]标记，直到不再发生改变。图 3-25 说明了这个步骤。

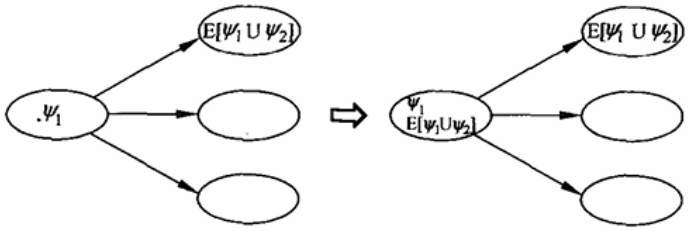

图 3-25　用形为 E[ψ_1 U ψ_2]的子公式标记状态过程的迭代步骤

- EX ψ_1：用 EX ψ_1 标记任何状态，如果其后继之一用 ψ_1 标记。

对 ϕ 的所有子公式(包括 ϕ 本身)完成标记后，输出标记为 ϕ 的状态。

这个算法的复杂度为 $O(f \cdot V \cdot (V+E))$，此处 f 是公式中连接词的个数，V 是状态个数，而 E 是迁移的个数。该算法关于公式的大小为线性的，而关于模型的大小是二次的。

还有别的更有效的方法：直接处理EG、聪明版直接处理EG... 不展开了。

3.6.3 LTL模型检测算法（Büchi 自动机）

基本策略

基本策略 设$\mathcal{M}=(S, \rightarrow, L)$是一个模型，$s \in S$，$\phi$是一个LTL公式。我们要确定是否$\mathcal{M}, s \vDash \phi$，即沿$\mathcal{M}$的从$s$出发的所有路径，$\phi$是否满足。几乎所有的LTL模型检测算法都遵循以下三步进行。

1. 为公式$\neg \phi$构造一个自动机，也叫做布景(tableau)。关于ψ的自动机称为A_{ψ}。于是，我们构造$A_{\neg \phi}$。该自动机有一个接受迹(accepting a trace)的概念。迹是命题原子的赋值序列。从一条路径出发，可以抽象出它的迹。该构造有如下性质：对所有路径π：$\pi \vDash \psi$当且仅当π的迹为A_{ψ}所接受。换言之，自动机A_{ψ}精确地编码满足ψ的迹。

因此，我们为$\neg \phi$所构造的自动机$A_{\neg \phi}$有性质：编码满足$\neg \phi$的所有迹；即所有不满足ϕ的迹。

2. 将自动机$A_{\neg \phi}$与系统的模型$\mathcal{M}$结合。结合运算的结果产生一个迁移系统，其路径既是自动机的路径又是系统的路径。

3. 在结合的迁移系统中搜寻，看看是否存在由s导出的状态出发的路径。如果存在一条，可以解释为$\mathcal{M}$中以s开始不满足ϕ的一条路径。

如果没有这样的路径，则输出“Yes，$\mathcal{M}$，$s \vDash \phi$”。否则，如果存在这样的路径，输出“No，$\mathcal{M}$，$s \nvDash \phi$”。在后一种情形中，可从找到的路径中提取出反例。

具体需要补充如何构造Buchi自动机

Problem: Does Transition system satisfy LTL formula φ?

In other word we can check: Does Transition system satisfy LTL formula !φ? (这里开始我用!代替¬)

如果找到一条路径满足了!φ，那就能证明模型不满足LTL公式。反之，如果不存在任何一条路径满足!φ，说明模型可以满足公式φ。

所以问题转化为：是否存在一条路径满足LTL公式 !φ?

如何解决?

1. 我们把LTL公式!φ 转化为一个 NBA buchi 自动机 $A_{!\varphi}$
2.

4 程序验证

6 二叉判定图