计算树逻辑公式

计算树逻辑（CTL）是一种时序逻辑，它的时间模型是一种无法确定将来的树状结构. 在未来有不同的路径，其中任何一个都可能是一个实际的路径。它使用原子命题作为基础，去描述系统的状态，然后用逻辑运算符和时序运算符去连接这些运算符形成公式。它被广泛的用于软件和硬件产品的形式验证，通过模型检测机去验证一个产品是否满足一些安全相关的属性。例如，CTL 可以描述：如果当一些初始条件被满足(如， 所有的程序变量都是正数，没有车在高速公路上跨越两个车道。)， 那么程序的所有可能的执行，避免一些不良状况（如，除以一个数零或两车在高速公路上发生碰撞）. 在这个例子中，这个安全属性可以被模型检测机验证，它会遍历所有满足初始条件的状态和以他们作为原始状态的变迁，确保所有的变迁都满足这个安全属性。下面，我们将详细的介绍 CTL 公式的语法，语义，还有一些例子。

CTL 公式语法

一个 CTL 公式 **ctl_expr** 可以用如下的递归语法进行定义:

```
ctl_expr ::
    simple_expr                 -- a simple boolean expression
    | ( ctl_expr )
    | ! ctl_expr                -- logical not
    | ctl_expr & ctl_expr       -- logical and
    | ctl_expr | ctl_expr       -- logical or
    | ctl_expr xor ctl_expr     -- logical exclusive or
    | ctl_expr xnor ctl_expr    -- logical NOT exclusive or
    | ctl_expr -> ctl_expr      -- logical implies
    | ctl_expr <-> ctl_expr     -- logical equivalence
    | EG ctl_expr               -- exists globally
    | EX ctl_expr               -- exists next state
    | EF ctl_expr               -- exists finally
    | AG ctl_expr               -- forall globally
    | AX ctl_expr               -- forall next state
    | AF ctl_expr               -- forall finally
    | E [ ctl_expr U ctl_expr ] -- exists until
    | A [ ctl_expr U ctl_expr ] -- forall until
```

在每一行语法定义的最后，都有每个符号的简短的意思的介绍。(例如, !表示逻辑非， |表示逻辑或， EG表示全局存在)。在语法定义中的所有符号，可以被划分为三个集合 ： 原子命题,逻辑运算符，时序运算符 。每个集合都有一些共性的特点和作用。下面讲结合例子给出每个集合的详细说明。

原子命题

simple_expr 是一个原子公式的集合. 原子公式的确切形式取决于它要考虑的逻辑：例如对于命题逻辑，它的原子公式就是一些命题变量。对于谓词逻辑，原子公式就是一些带参数的谓词符号，参数是一些术语。在模型理论中，原子公式就是符号串和一个给定的签名，这个符号串可能会满足也可能不满足某个模型所表达的模式。

例如，像公式 $P(x)\ \&Q(y,\ f(x))\ |\ R(z)$ 包含三个原子公式：$P(x)$，$Q(y,\ f(x))$，$R(z)$

逻辑运算符

!，&，|，XOR，XNOR，→,↔ 都是逻辑运算符. 逻辑运算符是一个符号或者一个单词，用于在语法有效的情况下连接两个或两个以上的句子，这样产生的复合句子有一个真值表，值取决于原始子句的值。

例如，字句 P 的意思是 P= 在下雨，Q = 我在户内， 可以通过如下的逻辑连接符进行连接，表达如下不同的意思:

- 没有下雨而且我在户内 (!P&Q)
- 如果下雨了，我将在户内 (P→Q)
- 如果我在户内，说明外面下雨了 (Q→P)
- 我在户内，当且仅当外面下雨了 (P↔Q)

通常考虑总是真的表达式或者总是假的表达式也很普遍:真值(⊤, 1 or T) 假值 (⊥, 0, or F)

时序运算符

A, E, X， F，G，U 都是时序运算符:

- A 意味着全部的路径
  1. **A φ** – **A**ll: **φ** 在从当前路径出发的全部路径上都必须满足
- E means along at least one path
  2. **E φ** – **E**xists: **φ** 在从当前路径出发的至少一条路径上都必须满足
- X， F， G， U are Path-specific quantifier
  1. **X φ** – Ne**x**t: **φ**在当前状态的下一个状态被满足 (这个运算符号有时也表示为 **N** 而不是 **X**).
  2. **G φ** – **G**lobally: **φ** 在随后的整个路径上都必须被满足.
  3. **F φ** – **F**inally: **φ** 最终会被满足(在随后路径的某个地方).
  4. **φ U** ψ – **U**ntil: **φ**在将来某个地方 ψ 被满足.这也就是说 ψ 在将来会被满足.

我们已经介绍了 CTL 公式的语法相关的详细信息。给出了形式化的公式语法定义和符号的意义说明。同时还给出了一些例子。讲全部的符号分为原子命题，逻辑运算符，时序运算符，对每个集合给出了详细的说明。结合上面的描述，我们还能发现，CTL 公式中，有一个运算符的最小集合。CTL 公式可以就通过这些符号进行表示。这在模型检测中十分的有用。最小集合是{true, &, !, EG, EU, EX}. 然后，其它的操作符都可以通过如下的方式进行定义:

- EFϕ == E[trueU(ϕ)] (because Fϕ == [trueU(ϕ)])
- AXφ == ¬EX(¬φ)
- AGφ == ¬EF(¬φ) == ¬E[trueU(¬φ)]
- AFφ == A[trueUφ] == ¬EG(¬φ)
- A[φUψ] == ¬(E[(¬ψ)U¬(φ∨ψ)] ∨EG(¬ψ))

CTL 公式语义

CTL 公式是在变迁系统上解释的. 一个变迁系统是一个元组 M=(S, →, L) , S 是一个状态集合, → 是一个变迁关系, 可以被假设为是串行变迁关系, 例如. 每个状态都只有一个后继, L 是一个标识函数,给状态赋值一些命题字母. 让 M=(S, →, L) 表示一个变迁模型，满足:

$s \in S$, $P \in$ ctl_expr, $Q \in$ ctl_expr, 和 $\Phi \in$ ctl_expr

那么，CTL 运算符可以直观的语义可以定义为：

- **EX** P 在状态 s 中为真,如果存在一个状态 si,s 和 si 之间有一个变迁，然后 P 在状态 si 中成立。
- **AX** P 在状态 s 中为真,如果对任意的状态 si,s 和 si 之间有一个变迁，然后 P 在状态 si 中成立。
- **EF** P 在状态 s 中为真，如果存在一条变迁序列 s→ s1, s1→s2, . . . ,s(n-1) → sn, 然后 P 在状态 sn 中成立.
- **AF** P 在状态 s 中为真，如果对于全部的变迁序列 s→ s1, s1→s2, . . . ,s(n-1) → sn, 然后 P 在 sn 中成立.
- **EG** P 在状态 s 中为真，如果存在一条变迁序列 s→ s1, s1→s2, . . . p 在全部的 si 中成立.
- **AG** P 在状态 s 中为真， 若果对于全部的变迁序列 s→ s1, s1→s2, . . . P 在全部的 si 中成立.
- **E[P U Q]** 在状态 s 中为真， 如果存在一个变迁序列 s→ s1, s1→s2, . . . ,s(n-1) → sn， P 在 s 到 s(n-1) 之间的状态全部成立， Q 在状态 sn 中成立.
- **A[P UQ]** 在状态 s 中为真， 如果对于全部的变迁序列 s→ s1, s1→s2, . . . ,s(n-1) → sn， P 在 s 到 s(n-1) 之间的状态全部成立， Q 在状态 sn 中成立.

建立在模型上的表达式 （M, s |= Φ）的语义 被定义为建立在Φ的归纳结构，如下所示:

$$\Big((\mathcal{M},s)\models p\Big)\Leftrightarrow\Big(p\in L(s)\Big)$$
$$\Big((\mathcal{M},s)\models\neg\phi\Big)\Leftrightarrow\Big((\mathcal{M},s)\not\models\phi\Big)$$
$$\Big((\mathcal{M},s)\models\phi_1\wedge\phi_2\Big)\Leftrightarrow\Big(\big((\mathcal{M},s)\models\phi_1\big)\wedge\big((\mathcal{M},s)\models\phi_2\big)\Big)$$
$$\Big((\mathcal{M},s)\models\phi_1\vee\phi_2\Big)\Leftrightarrow\Big(\big((\mathcal{M},s)\models\phi_1\big)\vee\big((\mathcal{M},s)\models\phi_2\big)\Big)$$
$$\Big((\mathcal{M},s)\models\phi_1\Rightarrow\phi_2\Big)\Leftrightarrow\Big(\big((\mathcal{M},s)\not\models\phi_1\big)\vee\big((\mathcal{M},s)\models\phi_2\big)\Big)$$
$$\Big((\mathcal{M},s)\models\phi_1\Leftrightarrow\phi_2\Big)\Leftrightarrow\bigg(\Big(\big((\mathcal{M},s)\models\phi_1\big)\wedge\big((\mathcal{M},s)\models\phi_2\big)\Big)\vee\Big(\neg\big((\mathcal{M},s)\models\phi_1\big)\wedge\neg\big((\mathcal{M},s)\models\phi_2\big)\Big)\bigg)$$
$$\Big((\mathcal{M},s)\models AX\phi\Big)\Leftrightarrow\Big(\forall\langle s\rightarrow s_1\rangle\big((\mathcal{M},s_1)\models\phi\big)\Big)$$
$$\Big((\mathcal{M},s)\models EX\phi\Big)\Leftrightarrow\Big(\exists\langle s\rightarrow s_1\rangle\big((\mathcal{M},s_1)\models\phi\big)\Big)$$
$$\Big((\mathcal{M},s)\models AG\phi\Big)\Leftrightarrow\Big(\forall\langle s_1\rightarrow s_2\rightarrow\ldots\rangle(s=s_1)\forall i\big((\mathcal{M},s_i)\models\phi\big)\Big)$$
$$\Big((\mathcal{M},s)\models EG\phi\Big)\Leftrightarrow\Big(\exists\langle s_1\rightarrow s_2\rightarrow\ldots\rangle(s=s_1)\forall i\big((\mathcal{M},s_i)\models\phi\big)\Big)$$
$$\Big((\mathcal{M},s)\models AF\phi\Big)\Leftrightarrow\Big(\forall\langle s_1\rightarrow s_2\rightarrow\ldots\rangle(s=s_1)\exists i\big((\mathcal{M},s_i)\models\phi\big)\Big)$$
$$\Big((\mathcal{M},s)\models EF\phi\Big)\Leftrightarrow\Big(\exists\langle s_1\rightarrow s_2\rightarrow\ldots\rangle(s=s_1)\exists i\big((\mathcal{M},s_i)\models\phi\big)\Big)$$
$$\Big((\mathcal{M},s)\models A[\phi_1 U\phi_2]\Big)\Leftrightarrow\bigg(\forall\langle s_1\rightarrow s_2\rightarrow\ldots\rangle(s=s_1)\exists i\Big(\big((\mathcal{M},s_i)\models\phi_2\big)\wedge\big(\forall(j<i)(\mathcal{M},s_j)\models\phi_1\big)\Big)\bigg)$$
$$\Big((\mathcal{M},s)\models E[\phi_1 U\phi_2]\Big)\Leftrightarrow\bigg(\exists\langle s_1\rightarrow s_2\rightarrow\ldots\rangle(s=s_1)\exists i\Big(\big((\mathcal{M},s_i)\models\phi_2\big)\wedge\big(\forall(j<i)(\mathcal{M},s_j)\models\phi_1\big)\Big)\bigg)$$

对于时序运算符的语义，我们给出一些例子加以更详细的说明: 让 ″P″表示 ″我喜欢巧克力″， Q 表示 ″外面很暖和″

AG. P

″从今往后，我都很喜欢巧克力″

EF. P

"以后的某一时刻我可能会喜欢巧克力"

AF.EG.P

"总是有可能(AF)， 从某一个时刻开始，在我剩下的时候我都会很喜欢巧克力"

E((**EX**.P)**U**(**AG**.Q))

"存在这种可能：总有那么一天，从那天开始，以后都很暖和(AG.Q)， 那之前每一天，我都会在第二天喜欢巧克力(EX.P)."

下面的例子将会描述建立在系统之上公式的语义：M 表示系统模型，s 是初始状态，A 是设备名称，B 是设备名称，D 是设备名称：

- ((M ,s)|= AG [(D.V>minV) & (D.V< maxV)])

  系统模型必须满足，任何时刻，装置的运行速度在安全范围[minV,maxV]之内
- ((M ,s)|=AG [(D.P>minP) & (D.P<maxP)])

  系统模型必须满足，任何时刻，装置的目标位置在安全范围[minP,maxP]之内
- ((M ,s)|=AG [(A.interlock & B.interlock)→(A.V || B.V)])

  系统模型必须满足，任何时刻，装置碰到限位能在允许位移误差范围[0 maxE]停下

- （(M ,s)|=AG [(A.syn & B.syn)→(|A.V - B.V| < maxs)])

  系统模型必须满足，任何时刻，装置碰到限位能在允许位移误差范围[0 maxE]停下

- （(M ,s)|=AG [(A.syn & B.syn)&(|A.V - B.V| < maxs) → (|A.P-B.P|< maxp)]）

  系统模型必须满足，任何时刻，同步运行的装置，运行速度误差在[0 maxS]之内