开放系统中合作与竞争的时态逻辑

刘虎

澳大利亚新南威尔士大学计算机科学与工程学院

摘要: 时态逻辑模型检测是自动验证最重要的方法之一。近年来，模型检测技术与人工智能研究的结合，成为一个研究的热点。具体地，就是扩充或者修改模型检测的时态逻辑，使之能够刻画多agents系统的特征。交互时态逻辑（Alternating Time Temporal Logic），以下简称为ATL，是其中一种较为成功的框架。使用ATL，可以刻画多个agents的相互合作，即，agents通过相互合作保证计算系统进入预定的某个（些）状态。然而，agents之间的冲突，是现实中的计算系统的一个重要特征。本文基于ATL，扩充其为一种表达力更强的时态逻辑，称之为竞争交互时态逻辑（Competition Alternating Time Temporal Logic），简称为CATL。CATL的表达力，体现在它不仅可以刻画agents的合作，也能够刻画agents相互的竞争。而且，CATL的表达力并没有以提高计算复杂性为代价。

关键词: 竞争，时态模型检测，开放系统

中图分类号: B81 **文献标识码:** A

1. 引言

本文讨论的，是一种基于分叉时间结构上的时态逻辑。这种时态逻辑来源于计算机科学中的自动验证与人工智能多agents研究的结合。在过去二十年中，计算系统的自动验证技术得到了充分的发展。其中最为成功的方法是所谓的符号模型检测。符号模型检测使用时态逻辑来表达系统的性质，主要是使用*分叉时态逻辑*（Branching Time Temporal Logic）。*分叉时间*（Branching Time）指的是一种时间的抽象结构。形象来说，一个分叉时间结构是一个树形结构；其中的每一个时间点的过去都是一个线序，表明从每一时间点出发，过去总是唯一的；相反，一个时间点的未来则是分叉的，表明未来的不确定性。分叉时态逻辑本身是逻辑学研究的一个重要方向。本文所关心的是其中被用于自动验证的逻辑，即，*计算树逻辑*（Computation Tree Logic）[8]，简称为CTL。

CTL理解一个分叉时间结构为一个状态转换的计算系统。每一个时间点对应于一个当下的状态。一条路径，即一条极大线序，代表着在一种可能的系统运行下，系统状态的转换方式。计算树逻辑使用时态算子 X （“下一个”），G （“将来总是”），和 U （“直到”）以及一个路径量词 E 。路径量词用于表达某个性质在某些路径上成立：$E\varphi$ 被解释为存在一条计算路径，使得 φ 在该路径上成立。CTL模型检测问题指的是，给定一个CTL公式和一个系统，检测该公式在该系统的哪些状态中成立。时态模型检测的工具，例如SMV[12]，已经被应用于工业实践中[9]。

CTL是一种验证单过程系统的时态逻辑。因此，它不能够表达系统的不一致性，即，并行系统的各个不同的模块之间由于相互作用而产生的冲突。多值逻辑模型检测是一种分析不一致性的方法[7]。然而，它同时也增加了计算的复杂性。本文提出另一种刻画不一致性的

时态逻辑。我们的逻辑建基于*交互时态逻辑*（Alternating Time Temporal Logic，简称为ATL）。ATL是CTL的一个扩充，用于表达在开放系统中多agents的合作关系。我们扩充ATL，使得agents之间的竞争关系也得以表达。我们表明，尽管增加了ATL的表达力，却并没有增加模型检测问题的复杂性。

本文结构如下：下一章讨论本文的直观背景；第三章是对游戏结构（Game Structure）的简要介绍；本文的剩余部分详细讨论agents竞争的时态逻辑；最后一章给出模型检测的算法。

2. 建立开放系统中agent竞争的模型

一个开放系统（Open System）指的是一个与其环境进行交互的系统。在文献中已经有用于刻画开放系统特征的时态逻辑[10][11][1][3]。开放系统中验证的核心问题是所谓交互满足问题（Alternating Satisfaction）：某个性质被系统本身保证成立，而不管环境如何变化。交互时态逻辑ATL作为CTL的扩充，被用于刻画与验证类似游戏（game-like）的开放系统的性质。在一个类似游戏的系统中有多个游戏者或者agents，所有的agents的相互作用导致整个系统出现或者不出现某一状态。

使用博弈论意义下的游戏来建立并行系统的模型是一种自然的做法[13]。从博弈论的角度来看，不一致性来源于不同agents的行动之间的冲突。例如，两个agents同时试图访问某一个系统资源，或者两个agents同时希望将某个系统变元改变为不同的值。在类似这些情况下，冲突的agents之间将相互竞争，唯有胜利者的行动得到实际的实施。因为一般来说，我们无法预测哪个或者哪些agents将会获胜，所以一次竞争的结果将包括所有可能的情况。

ATL扩充了CTL的语言，使得它能够表达这样的性质：“某个名为a的agent能够保证系统永不会进入一个崩溃状态”，或者，“名为a和名为b的两个agents可以合作来保证系统最终将停机”。这些性质使用合作算子《》来表达。令 A 是一个agent的集合，《 A 》 ψ 读作“不管其它agents行为如何， A 中的agents能够通过合作来保证 ψ 成立”。相应于交互满足问题， A 中的agents被看作是系统的控制单元，而不在 A 中的agents则被作为系统运行的环境。令 Σ 为系统中所有的agents。CTL的路径量词 E 可以被定义为《 Σ 》； E 的对偶量词 ν 可以被定义为《 $\varnothing$ 》。尽管表达力强于CTL，ATL模型检测的计算复杂性与CTL的相同[1]：相对于被检测公式的大小，以及被检测系统的大小，其复杂性是PTIME-complete。一个称为MOCHA的基于ATL的模型检测工具已经被开发出来[2][4]。

在ATL的模型中，每个agetn被赋予一个策略集。假设从某个状态 q 出发，所有可能的未来的分叉集合为 C。直观上， C 包括所有在agents选择不同的策略组合的情况下，系统的所有的可能发展方向。某个agent选定了某个策略，则意味着 C 中的某些可能分叉由于该agent的选择而变得不可能。因此，agents选定策略等于是对 C 的限制。假设一个agent集合 A 中的agent都已经选定了它们各自的策略，以一个策略集 F_A 代表。令 $C' \subseteq C$ 是在它们选定策略 F_A 后系统的可能的分叉。令 ψ 是一个时态公式，则《 A 》 ψ 在状态 q 中为真，当且仅当，存在一个策略集 F_A，使得 ψ 在所有 C' 中的分叉为真。

ATL的意图是形式化一个系统中的agents通过合作来保证期望的性质。然而，实际的开放系统通常要更为复杂。例如，假如两个分别名为 a 和 b 的agents同时访问一个系统资源，那么这两个agents就不是合作的关系，而是相互竞争的关系。竞争的结果是只有其中一个agent能够访问该系统资源。刻画类似这样的竞争关系，是本文的目标。

我们使用类似于ATL的模型。我们接下来从直观上解释本文的做法。再以上述的两个agents为例。这两个agents在状态 q 下相互竞争的结果有两个：a 赢或者 b 赢。a 赢意味着“a 得以访问系统资源”。将上述引号中的陈述记为P。在这种情况下，从状态 q 出发所有可能的分叉C被限制到那些满足P的分叉上，记为 C'。类似地，陈述“b 得以访问系统资源”记为P’。P’显然是 b 赢的结果。令 C'' 为一个分叉的集合，由所有满足性质P’的分叉组成。因为不知道哪个agent会赢，所以作为这两个agents竞争的结果，从 q 出发将来所有可能的分叉是 $C' \cup C''$。一般地，我们有两个不相交的agents的集合 A 和 B。A（或者 B）中的agents相互合作，A 而 B 和之间则是相互竞争。

考虑一个在状态 q 的 AB 竞争。则 A 获胜的结果是：（1）A 的行动得以实施。令分叉集 C_1 由所有从 q 出发的满足性质（1）的分叉组成。则 A 获胜的结果是，从 q 出发的可能的分叉被限制为 C_1。同理，B 获胜的结果是：（2）B 的行动得以实施。令分叉集 C_2 由所有从 q 出发的满足性质（2）的分叉组成。则 B 获胜的结果是，从 q 出发的可能的分叉被限制为 C_2。由于 A 获胜和 B 获胜这两个结果都是可能的，所以，在状态 q 的 AB 竞争的结果是，所有可能的分叉集被限制为 $C_1 \cup C_2$。

agents行为之间的冲突通常表现为比上面的例子更复杂的状况。例如，令 A，B，C 是三个两两相互冲突的agents的集合。在这种情况下，可能的竞争结果将有三个：A 获胜，B 获胜，或者 C 获胜。本文将使用一个极大条件来指明可能的竞争的结果。简言之，一个竞争的可能的结果，就是一个极大的不冲突的行为集合中的所有行为都得以实施所产生的后果。

本文所建立的分叉时态逻辑建立在以上的分析之上。但是，应该注意，上面的分析是一个直观的说明，并不是准确的定义。例如，我们的形式定义中，实际上不是agents之间竞争，而是agents的行动之间的竞争。这意味着，agents只有当其行动在某一状态下冲突时才是竞争关系。这样的情况是可能的：两个agents在一个状态下是竞争关系，在另一个状态下则是合作关系。

3. 游戏结构（Game Structures）

在本章我们简要介绍游戏结构的定义。ATL 是直接定义在游戏结构上的。本文给出的竞争的时态逻辑将定义在一种游戏结构的变种上。本章的定义来源于[1]。形式上，

定义 3.1 一个*游戏结构*是一个六元组 $S = (\Sigma, Q, \Pi, \pi, d, \delta)$，其中

（1）$\Sigma = \{a_1, ..., a_k\}$ 是一个有穷的非空的 agents 的集合；

（2）Q 是一个有穷的非空的状态的集合；

（3）Π 是一个有穷的非空的命题的集合；

（4）对任一个状态 $q \in Q$，$\pi(q) \subseteq \Pi$ 是所有在状态 q 为真的命题的集合。函数 π 被称为*标记函数*（labeling function）；

（5）对任一个 agent $a_i \in \Sigma$，$d_{a_i}(q) \geq 1$ 是一个自然数，标明在状态 q 下 a_i 可能的行为的数量。我们使用各个自然数 $1, ..., d_{a_i}(q)$ 来标记这些可能的行为。对任一个状态 q，在 q 的一

个*行为矢量*（action vector）是一个多元组 $(j_{a_1},...,j_{a_k})$ 使得对于任一 $a_i \in \Sigma$，$1 \le j_{a_i} \le d_{a_i}(q)$。给定一个状态 q，$D(q)$ 是所有在 q 的行为矢量的集合，即，$\{1,...,d_{a_1}(q)\} \times ... \times \{1,...,d_{a_k}(q)\}$。

（6）一个状态 $q \in Q$，任一行为矢量 $(j_{a_1},...,j_{a_k}) \in D(q)$，$\delta(q, j_{a_1},...,j_{a_k}) \in Q$。函数 δ 被称为*转换函数*（transition function）。$\delta(q, j_{a_1},...,j_{a_k})$ 是在状态 q 下，agent a_i 分别选定行为 j_{a_i}，系统将进入的下一个状态。$\delta(q, j_{a_1},...,j_{a_k})$ 是一个状态而不是状态集，这说明如果所有 agents 都选定了它们的下一个行为，那么系统的下一个状态是确定唯一的。

对任意两个状态 q 和 q'，如果有一个行为矢量 $(j_{a_1},...,j_{a_k}) \in D(q)$ 使得 $q' = \delta(q, j_{a_1},...,j_{a_k})$，那么我们称 q' 是 q 的一个*后继*（succesor）。直观上，q' 是 q 的一个后继意味着，当系统处于状态 q，系统中的 agents 可以分别选择特定的行为使得系统的下一个状态是 q'。一个*计算*（computation）是一个无穷的序列 $\lambda = q_0 q_1 q_2 \ldots$ 使得对任意 $i \ge 0$，状态 q_{i+1} 是状态 q_i 的后继。我们称一个从状态 q 起始的计算为一个 q *计算*。对于一个计算 λ，对任意 $i \ge 0$，我们使用 $\lambda[i]$ 标记 λ 中的第 i 个状态；使用 $\lambda[0,i]$ 标记 λ 的有穷的前缀 $q_0 q_1 \ldots q_i$；使用 $\lambda[i,\infty]$ 标记 λ 的无穷的后缀 $q_i q_{i+1} \ldots$。

下面是一个来源于[1]的简单的游戏结构的例子。这个作为例子的系统有两个 agents，分别名为 a 和 b。a 以如下方式给系统变量 x 赋值：当 $x=0$ 时，a 保持 x 的值，或者将 x 的值变为 1；当 $x=1$ 时，a 保持 x 的值。类似地，b 以如下方式给系统变量 y 赋值：当 $y=0$ 时，b 保持 y 的值，或者将 y 的值变为 1；当 $y=1$ 时，b 保持 y 的值。对应于该系统的游戏结构是 $S_{xy} = (\Sigma, Q, \Pi, \pi, d, \delta)$，其中，

(1) $\Sigma = \{a, b\}$；

(2) $Q = \{q, q_x, q_y, q_{xy}\}$。状态 q 对应于 $x = y = 0$，q_x 对应于 $x=1$ 并且 $y=0$，q_y 对应于 $x=0$ 并且 $y=1$，q_{xy} 对应于 $x=1$ 并且 $y=1$；

(3) $\Pi = \{x=0, x=1, y=0, y=1\}$；

(4) π 的定义见（2）；

(5i) $d_a(q) = d_a(q_y) = 2$，$d_a(q_x) = d_a(q_{xy}) = 1$。$a$ 的行为 1 是不改变 x 的值，a 的行为 2 是改变 x 的值；

(5ii) $d_b(q) = d_b(q_x) = 2$，$d_b(q_y) = d_b(q_{xy}) = 1$。$b$ 的行为 1 是不改变 y 的值，b 的行为 2 是改变 y 的值；

(6) 状态 q 有四个后继：$\delta(q,1,1) = q$，$\delta(q,1,2) = q_y$，$\delta(q,2,1) = q_x$ 以及 $\delta(q,2,2) = q_{xy}$。状态 q_x 有两个后继：$\delta(q_x,1,1) = q_x$ 及 $\delta(q_x,1,2) = q_{xy}$。状态 q_y 有两个后继：$\delta(q_y,1,1) = q_y$ 及 $\delta(q_y,2,1) = q_{xy}$。状态 q_{xy} 只有一个后继：$\delta(q_{xy},1,1) = q_{xy}$。

无穷序列 $qqq_x q_x q_{xy}^{\omega}$ 是一个 S_{xy} 中的 q 计算。

令 q 为初始状态。直观上，a 能够保证 x 的值总是为0；a 不能够保证 x 和 y 的值都总是为0；a 和 b 相互合作则能够保证 x 和 y 的值都总是为0。上面三种情况用ATL公式分别表达，就是下面三个公式为真：

$$《\{a\}》\, G(x=0)，\ \neg《\{a\}》\, G(x=0 \wedge y=0)，《\{a,b\}》\, G(x=0 \wedge y=0)。$$

形式上，《$\{a\}$》$G(x=0)$为真是因为，a有一个策略，即在任何状态下都不改变x的值。使用这个策略在状态q，系统的计算路径被限制到了只有那些使$G(x=0)$为真的计算才是可能的。我们在这里不给出ATL的详细语义，有兴趣的读者可参见[1]。

4. 竞争交互时态逻辑（CATL）

4．1 句法

我们称本文给出的刻画 agents 竞争的时态逻辑为*竞争交互时态逻辑*（Competition Alternating-time Temporal Logic），简称为 CATL。本节将给出它的句法定义。CATL 的初始符号包括逻辑符号$\neg$和$\vee$，时态算子X（“下一个”）、G（“将来总是”）和U（“直到”），一个有穷的命题集合Π以及一个有穷的 agents 的集合Σ。CATL 公式有以下几种形式：

(S1) p，其中$p\in\Pi$是一个命题；

(S2) $\neg\varphi$或者$\varphi_1\vee\varphi_2$，其中φ、φ_1和φ_2是 CATL 公式；

(S3) $[A]X\varphi$，$[A]G\varphi$或者$[A]\varphi_1U\varphi_2$，其中φ，φ_1和φ_2是 CATL 公式，$A\subseteq\Sigma$是一个 agent 的集合。

布尔联接词$\rightarrow,\wedge,\leftrightarrow$按通常方式可定义。$X$，$G$和$U$是时态算子。$X\varphi$读作“$\varphi$在下一个状态下为真”；$G\varphi$读作“$\varphi$从现在开始总是为真”；$\varphi_1U\varphi_2$读作“$\varphi_1$为真直到$\varphi_2$为真”。

算子$[A]$是*（计算）路径量词*（computation operator）。公式$[A]\psi$将被理解为：不管处于何种环境中，A中的 agents 通过相互之间的合作与竞争，能够保证ψ为真。为表达方便，有时我们使用A中的所有元素来代表A，例如，记$[\{a,b\}]$为$[a,b]$。

应该注意，按照 CATL 的句法定义，一个纯时态公式不是 CATL 公式，即，形如$X\varphi$，$G\varphi$，$\varphi_1U\varphi_2$的表达并不是这里的合式公式。一个纯时态公式前面必须有路径量词。同样地，算子$[A]$也不能离开时态算子而单独出现。

我们将在下面给出给语言的语义。为了方便起见，我们首先给出纯时态公式的语义。由上，纯时态公式不是合式公式。所以下面定义的并不是 CATL 的语义。我们将使用它们来简化对 CATL 语义的表述：

（1）λ‘ $X\varphi$，当且仅当，$\lambda[1]$‘ φ。

（2）λ‘ $G\varphi$，当且仅当，对任意$i\geq 0$，$\lambda[i]$‘ φ。

（3）λ‘ $\varphi_1U\varphi_2$，当且仅当，存在一个$i\geq 0$使得$\lambda[i]$‘ φ_2，并且对任意$0\leq j<i$，我们有$\lambda[j]$‘ φ_1。

4．2 广义游戏结构

我们在本节给出 CATL 的模型。该模型由扩充游戏结构而来，我们称之为*广义游戏结构*（General Game Structure）。广义游戏结构保留所有游戏结构的元素，另外加入一个新的函数，称之为*冲突函数*（conflict function）。具体地，我们有以下定义：

定义 4.1 一个*广义游戏结构*是一个七元组 $S=(\Sigma,Q,\Pi,\pi,d,\beta,\delta)$，其中 Σ,Q,Π,π,d 的定义与游戏结构的相同，详见定义 3.1。

冲突函数 β 定义如下：

- 令 $\{1_{a_i},...,d_{a_i}(q)\}$ 为 agent a_i 在状态 q 下所有可能的行为集合。令 $D'(q)=\{1_{a_1},...,d_{a_1}(q)\}\cup...\cup\{1_{a_k},...,d_{a_k}(q)\}$ 是所有 agents 在状态 q 下所有可能的行为集合。则 $\beta: D'(q)\to 2^{D'(q)}$ 是一个满足以下条件的函数：

  （1）$\beta(i_a)\cap\{1_a,...,d_a(q)\}=\varnothing$；

  （2）如果 $j_b\in\beta(i_a)$，那么 $i_a\in\beta(j_b)$。

状态 q 下的一个*极大不冲突集*（maximal non-conflict set）MNS 是一个满足以下条件的行为的集合：

（1）$MNS\subseteq\{j_{a_1},...,j_{a_k}\}$，其中 $(j_{a_1},...,j_{a_k})\in D(q)$ 是一个状态 q 的行为矢量；

（2）MNS 不包含任何冲突的行为，即，对任意 $j_a\in MNS$，$\beta(j_a)\cap MNS=\varnothing$；

（3）MNS 是极大的，即，对任意 $j_b\in\{j_{a_1},...,j_{a_k}\}\setminus MNS$，$\beta(j_b)\cap MNS\neq\varnothing$。

转化函数 δ 定义如下：

- 对任意状态 q 下的一个极大不冲突集 MNS，$\delta(q,MNS)\in Q$。

冲突函数 β 映射每个行为 i_a 到一个行为的集合 $\beta(i_a)$。直观上，$\beta(i_a)$ 中的行为是与 i_a 冲突的行为。限制条件（1）决定了一个 agent 的自身行为不相互冲突。限制条件（2）决定了冲突总是相互的，即如果 i_a 和 j_b 冲突则 j_b 和 i_a 冲突。我们经常把 agents 行为之间的冲突简称为 agents 之间的冲突，例如，两个行为 i_a 和 j_b 冲突被简称为 agent a 和 b 冲突。

广义游戏结构与游戏结构一样，仍然是*确定的*：系统将要进入的下一个状态由参与游戏的 agents 完全决定。在游戏结构中，如果所有的 agents 作出了决定，那么系统进入的下一个状态也就决定了；而在广义游戏结构中，由于存在冲突，所有 agents 的决定也许不能够都得到实施。因此，需要使用极大不冲突的概念。

在广义游戏结构中，后继、计算等概念的定义类似其在游戏结构中的定义。

4．3 策略及其输出

为了给出 CATL 的语义，首先需要给出一些必要的定义。类似的定义已在[1]中出现。我们修改[1]中的定义以适应当前的上下文。

首先我们定义 angnts 的策略（strategies）。一个 agent 的策略决定了该 agent 在每个给定状态下将会采取的行为。对于任一给定状态，agent 的行为是用该行为所产生的后果来表示，即，采取该行为后系统可能进入的下一个状态。我们有以下的定义：

定义 4.2 令 $S=(\Sigma,Q,\Pi,\pi,d,\beta,\delta)$ 是一个广义游戏结构。令 ω 是自然数的集合。名为 a 的

agent 的一个*策略* f_a 是一个函数 $f_a : Q \to \omega$ 使得，对任意 $q \in Q$，我们有 $1_a \le f_a(q) \le d_a(q)$。

显然，agent a 的一个策略 f_a 决定了他在任意给定的系统状态 q 下将要采取的行为，即，$f_a(q)$。对于任意 agents 的集合 $A \subseteq \Sigma$，一个 A 的*策略集*（collective strategies）是一个策略的集合 $F_A = \{f_a \mid a \in A\}$。接下来我们要定义在给定状态 q 下，A 中的 agents 采用了策略集 F_A 将使系统进入何种状态。A 中的 agents 根据 F_A 中的策略来各自行为，有可能会相互冲突，即，F_A 中所指明的行为按照冲突函数 β 是冲突的。在这种情况下，agents 之间将相互竞争。我们并不关心具体的竞争的过程，而只关心竞争的可能的结果。直观上，竞争的结果必然是某些 agents 的行为得以实施，某些 agents 的行为不被实施。那些得以实施的 agents 的行为，相互之间必然是不冲突的，否则他们将会通过进一步地竞争来消除冲突。而如果两个 agents 的行为不冲突，那么必然有一种可能的竞争结果，使得他们两个的行为都得以实施。

由于 agents 之间行为冲突的模式是复杂的。在任意一个状态下，我们并不总是能够指出一个 agents 的集合 A 使得 A 中的 agents 相互合作，A 中的 agents 与 $\Sigma - A$ 中的 agents 相互竞争。譬如，这样的情况是可能的：a 和 b 冲突；c 和 a、b 都不冲突。在这种情况下，显然没有办法找到上述性质的 A。假设另一种情况：a、b 和 c 三者之间两两都冲突。在这种情况下，我们只可能有三个相互不冲突的 agent 的集合，即 $\{a\}$、$\{b\}$ 和 $\{c\}$。我们将把极大的不冲突集作为一种可能的竞争后的结果。形式上，我们有下面的定义：

定义 4.3 令 A 是一个 agents 的集合，$F_A = \{f_a \mid a \in A\}$ 是他们的一个策略集，q 是一个状态。令 $j_A = \{f_a(q) \mid a \in A\}\}$ 是由 F_A 指定的他们在 q 将采取的行为。我们称一个 j_A 的子集 MNS_{F_A} 是一个在 q 的 A 赢集，如果以下条件成立：

（1）$MNS_{F_A} \subseteq MNS$，其中 MNS 是一个在 q 的极大不冲突集；

（2）对任意集合 M 使得 $MNS_{F_A} \subset M \subseteq j_A$，$M$ 不满足条件（1）。

定义 4.4 令 A 是一个 agents 的集合，$F_A = \{f_a \mid a \in A\}$ 是他们的一个策略集，q 是一个状态。策略集 F_A 在 q 的*输出*（outcome）$out(q, F_A)$ 是一个 q 计算的集合使得，一个计算 $\lambda = q_0 q_1 \ldots \in out(q, F_A)$ 当且仅当以下条件成立：

（1）$q_0 = q$；

（2）对任意 $i \ge 0$，存在一个在 q_i 的 A 赢集 MNS_{F_A} 使得，存在一个在 q_i 的极大不冲突集 MNS 满足以下条件：

（i） $MNS_{F_A} \subseteq MNS$；

（ii） $\delta(q_i, MNS) = q_{i+1}$。

定义 4.4 是符合我们在第二章中所讨论的直观背景。其中，条件（2i）表明 MNS_{F_A} 所代表的 agents 获胜，其行为 MNS_{F_A} 得以实施；条件（2ii）则是 MNS 实施的结果。

4．4 语义

使用以上定义的概念，我们在本节中给出 CATL 的语义定义。广义游戏结构 S 中的状态 q 满足 CATL 的公式 φ（使 φ 为真），记为 $S, q \text{‘}_{CATL} \varphi$。如果上下文允许，我们将其简记为 $q \text{‘}_{CATL} \varphi$。

定义 4.5 真值关系 ‘_{CATL} 的语义规则定义如下：

（1）$q \text{‘}_{CATL} p$，当且仅当，$p \in \pi(q)$，其中 p 是一个命题。

（2）$q \text{‘}_{CATL} \neg\varphi$，当且仅当，$q \not\text{‘}_{CATL} \varphi$。

（3）$q \text{‘}_{CATL} \varphi_1 \vee \varphi_2$，当且仅当，$q \text{‘}_{CATL} \varphi_1$ 或者 $q \text{‘}_{CATL} \varphi_2$。

（4）$q \text{‘}_{CATL} [A]\psi$，其中 $\psi \in \{X\varphi,\ G\varphi,\ \varphi_1 U \varphi_2\}$，当且仅当，存在一个策略集 F_A 使得对所有的计算 $\lambda \in out(q, F_A)$，我们有 $\lambda \text{‘}\ \varphi$。

注意在上面的定义 4.5（4）中，使用了在 4.1 节定义的纯时态公式的语义“$\lambda \text{‘}\ \varphi$”，它本身并不是 CATL 语义定义的一部分。“$\lambda \text{‘}\ \varphi$”是作为一个缩写，简化我们的表述。定义 4.5（4）的含义是，公式 $[A]\psi$ 在状态 q 下为真，当且仅当，存在一个策略集 F_A 使得，当 A 中的 agents 在状态 q 中分别使用 F_A 中策略，在作为他们相互合作与竞争的结果而产生的那些计算（路径）中，纯时态公式 ψ 必然为真。

下面是一个简单的 CATL 的例子。系统有两个变元 x，y。系统初始状态 q 下 $x=0$，$y=0$。两个 agents，分别名为 a 和 b，以如下方式给 x 赋值：当 $x=0$ 时，a 保持 x 的值，或者将 x 的值变为 1；b 保持 x 的值，或者将 x 的值变为 2。a 以如下方式给 y 赋值：当 $y=0$ 时，a 保持 y 的值，或者将 y 的值变为 1。因为本例中只考虑初始状态的下一个状态，所以我们省略在别的状态下 agents 的行为定义。假定一个 agent 在同一时间只能做一个行为，即一个 agent 不能同时操作两个变元。该系统形式化为一个广义游戏结构 $S_x = (\Sigma, Q, \Pi, \pi, d, \beta, \delta)$，其中，

(1) $\Sigma = \{a, b\}$；

(2) $Q = \{q, q_{x1}, q_{x2}, q, q_{x1y}, q_{x2y}\}$。状态 q 对应于 $x=0$，q_{x1} 对应于 $x=1$，等等；

(3) $\Pi = \{x=0, x=1, x=2, y=0, y=1\}$；

(4) π 的定义见（2）；

(5) $d_a(q)=4$，$d_b(q)=2$。a 的行为 1_a 是保持 x 的值，2_a 是将 x 的值变为 1，3_a 是保持 y 的值，4_a 是将 y 的值变为 1。b 的行为 1_b 是保持 x 的值，2_b 是将 x 的值变为 2；

（6） $\beta(1_a)=\{2_b\}$，$\beta(2_a)=\{1_b, 2_b\}$，$\beta(3_a)=\beta(4_a)=\varnothing$，$\beta(1_b)=\{2_a\}$，$\beta(2_b)=\{1_a, 2_a\}$。

(7) q 有 5 个后继：$q, q_{x1}, q_{x2}, q_y, q_{x2y}$。$\delta(q, 1_a, 1_b)=\delta(q, 3_a, 1_b)=q$，$\delta(q, 2_a)=q_{x1}$，$\delta(q, 3_a, 2_b)=q_{x2}$，$\delta(q, 4_a, 1_b)=q_y$，$\delta(q, 4_a, 2_b)=q_{x2y}$。

按照 CATL 的语义，显然有 $q \text{‘}_{CATL} [a,b]X(y=0)$，$q \text{‘}_{CATL} [a,b]X(y=1)$。直观上，这是因为 a 控制 y 的值。容易验证 $q \text{‘}_{CATL} [a,b]X(x=0)$。直观上，在状态 q，如果 a，b 都使用策略保持 x 的值，则 x 的值在下一个状态保持为 0。同理，有 $q \text{‘}_{CATL} [a,b]X(x=2)$，

这是因为在状态 q，如果 a 使用策略使得不影响 x 的值，b 使用策略使得给 x 赋值 2，则 x 的值在下一个状态为 2。但是，我们有，$q \not\models_{CATL} [a,b]X(x=1)$。直观上，要想使 $x=1$，a 必须使用行为 2_a，而该行为与 b 的行为都冲突，竞争的结果并不总是能够保证 x 的值被变为 1。

4．5 CATL 的表达力

ATL 是 CATL 的一个特例：如果我们仅仅考虑游戏结构，或者说，无冲突的广义游戏结构，那么 ATL 的语义与 CATL 相同。在这个意义下，我们有

定理 4.1 CATL 的表达力强于 ATL。

证明概要：由于所考虑的广义游戏结构是没有冲突的。所以，我们可以将其中的转换函数 δ 扩充为一个定义在 $Q \times D(q)$ 的全函数 δ'。所得到的（广义）游戏结构中，任意 ATL 公式《A》ψ 与 CATL 公式 $[A]\psi$ 语义等价。□

5. CATL符号模型检测

我们在本章中给出 CATL 的符号模型检测的算法。CATL 符号模型检测问题指的是，给定一个 CATL 公式 φ 和一个广义游戏结构 S，找出所有在 S 中满足 φ 的状态。

我们的算法中需要用到一个反像函数（pre-image function），记为 $Preimage$。直观上，当输入一个 agents 的集合 A 和一个状态集 Q_1，函数 $Preimage$ 输出一个状态集 Q_2 使得，一个状态 $q \in Q_2$，当且仅当，当系统位于状态 q 时，A 中的 agents（通过合作与竞争）能够保证系统的下一个状态落在 Q_1 中。换句话说，如果 Q_1 是所有满足公式 φ_1 的状态的集合，那么 Q_2 就是所有满足公式 $[A]X\varphi_1$ 的状态的集合。形式上，

- $Preimage$ 是一个函数 $Preimage: 2^{\Sigma} \times 2^{Q} \to 2^{Q}$ 使得 $q \in Preimage(A, Q_1)$，当且仅当，存在一个行为矢量 $(j_1, ..., j_k) \in D(q)$，对于所有在 q 的 A 赢集 MNS_{F_A} 使得 $MNS_{F_A} \subseteq \{j_1, ..., j_k\}$，所有在 q 的极大不冲突集 MNS 使得 $MNS_{F_A} \subseteq MNS$，我们有 $\delta(q, MNS) \in Q_1$。

CATL 模型检测算法的控制结构与 ATL 的类似[1]，因此也类似与 CTL 的算法[6]。算法的核心是一个函数 $Check$。$Check$ 的输入是一个 CATL 公式 φ 和一个广义游戏结构 S，输出一个 S 中满足 φ 的状态集。$Check$ 如图 1 所示：

```
function Check(φ, S)
    if φ ∈ Π then
        return {q | φ ∈ π(q)}
    else if φ = ¬φ1 then
        return Q \ Check(φ1, S)
    else if φ = φ1 ∨ φ2 then
        return Check(φ1, S) ∪ Check(φ2, S)
```

else if $\varphi = [A]X\varphi_1$ **then**

return *Preimage*(*A*, *Check*(φ_1, *S*))

else if $\varphi = [A]G\varphi_1$ **then**

$Q_1 := Q$; $Q_2 := Q_3 :=$ *Check*(φ_1, *S*)

while $Q_1 \not\subseteq Q_2$ **do** $Q_1 := Q_1 \cap Q_2$; $Q_2 :=$ *Preimage*(A, Q_1) $\cap Q_3$ **od**

return Q_1

else if $\varphi = [A]\varphi_1 U \varphi_2$ **then**

$Q_1 := \varnothing$; $Q_2 :=$ *Check*(φ_2, *S*); $Q_3 :=$ *Check*(φ_1, *S*)

while $Q_2 \not\subseteq Q_1$ **do** $Q_1 := Q_1 \cup Q_2$; $Q_2 :=$ *Preimage*(A, Q_1) $\cap Q_3$ **od**

return Q_1

图1

容易验证该算法的正确性，参见[1]。我们有下面的定理，证明从略。

定理 5.1 令 φ 是一个 CATL 公式， S 是一个广义游戏结构。则

$S, q \text{ ‘}_{CATL} \varphi$ ，当且仅当， $q \in Check(\varphi, S)$ 。

最后是 CATL 模型检测的计算复杂性。

定理 5.2 对于一个有 m 个转换的广义游戏结构，一个长度为 l 的 ACTL 公式，ACTL 模型检测问题可以在时间 $O(m \cdot l)$ 内完成。该问题属于 PTIME-complete。

证明：这里仅给出证明概要。要证明的是图1的算法可以在时间 $O(m \cdot l)$ 内完成。由于公式的长度限制为 l ，我们只需要证明图1中的每一个“**else if**”都可以在时间 $O(m)$ 内完成。我们使用与[1]中定理5.2类似的策略来证明这一点。首先把任意给定的一个广义游戏结构 S 转换为一个只有两个agents的轮转同步的游戏结构（turn-based synchronous） S_A ，即，在每一个状态下只实施一个agent行为的游戏结构。构造的过程类似于[1]命题5.1，我们需要替换其中的行为矢量为极大不冲突集，其它部分的构造做相应修改。如果 S 有 m 个转换，则 S_A 有 $O(m)$ 个状态和转换。在 S 检测CATL公式的任务可以通过在 S_A 上检测相应的公式来完成。而轮转同步游戏结构的检测问题可以在线形时间内完成[5]，即， $O(m)$ 。□

参考文献：

[1] Alur, R., Henzinger, T. A., Kupferman, O., Alternating-time temporal logic, in *Proceedings of the 38th IEEE Symposium on Fondations of computer science*, 1997. pages 100-109.

[2] Alur, R., Henzinger, T. A., Mang, F. Y. C., Qadeer, S. , Rajamani, S. K., and Tasiran, S., Mocha: modularity in model checking, in *Proceedings of the Tenth International Conference on Computer Aided Verification (CAV98)*, LNCS Vol. 1427, Springer Verlag, Germany, 1998. pp. 521-525.

[3] Alur, R., Henzinger, T. A., Reactive Modules, *Formal Methods in System Design*, Vol. 15-1, 1999. pp. 7-48.

[4] Alur, R., de Alfaro, L., Henzinger, Krishnan, S. C., Mang, F. Y. C., Qadeer, S. , Rajamani, S. K., and Tasiran, S., MOCHA user manual, University of California, Berkeley Report, 2000.

[5] Beeri, C., On the membership problem for functional and multivalued dependencies in relational databases, ACM Transactions on Database Systems, Vol. 5, 1980. pp. 241-259.

[6] Burch, J. R., Clarke, E. M., Mcmillan, K. L., Dill, D. L., and Hwang, L. J, Symbolic model checking: 10^{20} states and beyond, *Information and Comptation*, Vol. 98-2, 1992. pp. 142-170.

[7] Chechik, M., Easterbrook, S. M., and Devereux, B., Model checking with multi-valued temporal logics, in *Proceedings of the 31st IEEE International Symposium on Multiple-Valued Logic (ISMVL01)*, IEEE Computer Society, 2001. pp. 187-192.

[8] Clarke, E. M., Emerson, A., Design and synthesis of synchronization skeletons using branching-time temporal logic, in *Logic of Programs: Workshop, Yorktown Heights*, in *Lecture Notes in Computer Science*, Vol. 131, Spring-Verlag, Berlin, 1981, pages 52-71.

[9] Clarke, E. M., Grumberg, O., Peled, D. A., *Model Checking*, The MIT Press, Cambridge, MA, 2000.

[10] Hoare, C. A. R., *Communicating Sequential Processes*, Prentice Hall, 1985.

[11] Lynch, N. A., *Distributed Algorithms*, Morgan Kaufmann, 1996.

[12] Mcmillan, K. L., *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

[13] Osborne, M. J., Rubinstein, A., *A Course in Game Theory*, The MIT Press, Cambridge, MA, 1994.

A temporal Logic of cooperation and competition in open systems

Abstract: Model checking techniques based on branching time temporal logics have been widely used for formal specification and verification of concurrent systems. Alternating-time temporal logic (ATL) has been proposed recently to formalize a game-like specification of open systems. The property that some players can cooperate to ensure that a system enters certain states is characterized by cooperation operators in ATL. A practical system usually contains processes that may be in conflict with each other. That motivates us to extend ATL to more expressive formalisms in which competitions among players can be characterized. We show that the verification languages presented in this paper share the tractability of model checking with ATL and CTL.

Key words: competition, temporal model checking, open systems.