时序逻辑 TLA+ 的安全性，活性和公平性

(Safety, Liveness and Fairness)

Leslie Lamport

May, 2019

版权声明 (Copyright)

译者序

Lamport 的这篇《Safety, Liveness and Fairness》是 2019 年 5 月撰写。是离现在最近的一篇详细阐述 TLA 的许多重要概念和相关概念之间时序逻辑关系的文章，特别是对公平 (Fairness) 属性与活性 (Liveness) 属性的关系，弱公平性 (Weak Fairness) 和强公平性 (Strong Fairness) 的关系，给出了严格的数学描述和相关证明。

译者在对相关概念和推理阅读和翻译的同时，对一些略微复杂的证明给出了一些力所能及的注释。文章中，Lamport 在证明 Fairness 公平性属性一定是一个活性 Liveness 的部分，译者发现了一个证明和推理错误，在反复思考后，通过与 Lamport 的亲自讨论，Lamport 认为确实是一个错误，并给出了新的证明。我们在本文中采纳了 Lamport 给出的新的证明并给出了 Lamport 的英文原文（通过与译者的 email 交流）。对于时序逻辑 TLA 感兴趣的读者，建议可以考虑先阅读和理解相关的 TLA 的概念和之间的关系，然后再阅读 Lamport 在 1994 年发表在 ACM 上的文章《The Temporal Logic of Actions》。

Lamport 是当代重量级的逻辑学家，计算机科学家和图灵奖获得者。对整个人类在计算机理论，网络和分布式系统方面做出了重大的基础性的贡献。

Lamport 今年 80 岁高龄了，译者发现，Lamport 每天都还在非常勤奋和积极的研究工作，对于晚辈们的各种问题，都是很耐心的，花时间来解释。令人无限的尊敬和由衷的佩服。

2021 年 6 月 1 日于加州硅谷

序言 (Preface)

在整个 TLA+ 的各种文献中（包括在 TLA + 视频课程中），非正式地广泛的使用了术语“安全性 (Safety)，活性 (Liveness) 和公平性 (Fairness)”。本文假设你可以理解一个简单的 TLA + 规约，因此可能对这些术语的含义有所了解。在这篇文章中，我们会对这几个概念精确定义，并讨论其重要性。尽管不是用 TLA+ 来形式化描述，但都是通过数学的方法，包括相关的讨论。如果您不习惯阅读数学表达，这可能会很有点难。但如果您想更全面地了解时许逻辑 TLA +，努力读懂这篇文章是值得的。

The terms safety, liveness, and fairness are used informally throughout the TLA+ documentation—including in the TLA+ Video Course. I assume you can understand a simple TLA+ specification, so you probably have some idea what these terms mean. They are defined precisely here and their significance is discussed. Although they are not written formally in TLA+, the definitions are mathematical and so is the discussion. If you' re not used to reading mathematics, this may be tough going. It' s worth the effort if you want a more complete understanding of TLA+. Text colored like this in the table of contents or like this elsewhere is a clickable link.

目录

1 TLA+ 的部分语义介绍

值 (Values)

值是形式数学中的一个集合，这个集合包括可以表达为 TLA+ 常量表达式的所有对象。例如，$\sqrt{3}$ $x^2 + y^2 = 1$ 的实数对 (x，y 的集合也是一个值。这两个值可以在 TLA+ 中这样表示：

```
CHOOSE v \in Real : (v > 0) /\ (v^2 = 3)
{z \in Real \X Real : z[1]^2 + z[2]^2 = 1}
```

状态 (States)

状态是对变量的赋值分配[1]。TLA+ 里，存在着无限多个变量（假设 TLA+ 标识符的长度不受限制），但是任何范式只能包含有限数量的变量。步骤 (Step) 是一对状态，写成 $u \to v$ 而不是 $u\ v$。要记住的一个重点是，状态可以将任何值赋值给任何变量；变量没有类型。

行为 (Behavior)

一个行为是任何一个无限的状态序列。可以把状态序列 $s1, s2, s3, ...$ 表达为 $s1 \to s2 \to s3 \to \cdot\cdot\cdot.$ 一个行为中的一步 (Step) 表示的是行为中相邻的两个状态。例如，$s42 \to s43$ 是行为 $s1 \to s2 \to s3 \to \cdot\cdot\cdot$ 中间的一步。值得记住的是一个行为可以是状态的任何序列，不一定是满足规约的序列。

属性 (Property)

一个属性是对一个行为的谓词/断言，取值为布尔值 (TRUE 或者 FALSE)。我们用 $b \models P$ 来表示属性 P 被赋值在一个行为 b 上的真假值。如果 $b \models P$ 为真，我们说 b 满足属性 P。

TLA+ 的行为属性是通过时序逻辑的范式来表达的。我们把一个 TLA+ 范式与其表述的属性一起来理解。例如，如果 b 是 $s_1 \to s_2 \to s_3 \to \cdot\cdot\cdot$ 的一个行为，F 是一个 $(x = 1) \wedge [][x' = x+1]_x$ 的范式，那么当且仅当行为的初值是 1；然后对于每一步 $(s_i \to s_{i+1}$，变量 x 的值要么不变或者是加 1 的时候，$b \models F$ 保持为真。

[1]可以理解一个变量代表了一个状态。

TLA+ 中的每一个属性都是 stuttering 不敏感的。这意味着对于任意一个 TLA+ 范式 F 和任意一个行为 b，如果 $\hat{b}$ 是在行为 b 上增加或者减少了 stuttering 步骤得到的行为序列，那么 $b \models F$ 等价于 $\hat{b} \models F$。例如，如果 b 是一个行为序列 $s1 \to s1 \to s2 \to s2 \to s3 \to s3 \to s4 \to s4 \to s5 \to s5 \to s6 \to s6 \to \cdots$，然后 $\hat{b}$ 是 $s1 \to s2 \to s2 \to s3 \to s3 \to s3 \to s4 \to s4 \to s4 \to s4 \to s5 \cdots$[2]，那么 $b \models F$ 等价于 $\hat{b} \models F$。关于为什么 TLA+ 只考虑 stuttering 不敏感的范式，在 TLA+ 的高级讨论话题的网站上有相关的解释。因为我们只考虑这样的属性，因此 TLA+ 范畴下，所有的属性都是 stuttering-insensitive，不敏感或者说状态不变的。

2 安全性 (Safety) 和活性 (Liveness)

我们定义一个行为序列 b 的前缀 (Prefix) 为，给定一个大于 0 的整数 n，序列的前 n 个状态/序列。例如，$s1$ 和 $s1 \to \cdots \to s42$ 是行为 $s1 \to s2 \to s3 \to \cdots$ 的前缀。
对于任意一个有限状态序列 s，我们定义 $s^\sharp$ 为在 s 最后的状态后添加无限个重复的拷贝[3]。例如，如果 s 是 $s1 \to \cdots s42$，那么 $s^\sharp$ 为 $s1 \to \cdots \to s42 \to s42 \to s42 \to s42 \to \cdots$
$s^\sharp$ 的行为表达了一个系统在执行了前缀 s 个状态序列之后，停止不动 (Halt) 的行为。

安全性 (Safety)
对于状态的任何有限序列 s 和任何属性 P，我们将 $s \models P$ 定义为等价于 $s^\sharp \models P$。如果对所有行为 b 都满足以下条件，则将属性 P 定义为是一个**安全属性 (Safety)**：对于行为序列 b，
$b \models P$ 为真，当且仅当对于 b 的所有的前缀 s，$s \models P$ 为真。

[2]减少了一个 s1，增加了一个 s3，增加了 2 个 s4。
[3]可以理解在最后这个状态下递归，或者是无限循环，状态不再发生改变。

下面的几个表达方法是与上面的定义等价的方式:
* 行为 b 满足属性 P，当且仅当 b 的每一个前缀都满足 P[4]。
* $b \models P$ 为假，当且仅当至少存在一个前缀 s，$s \models P$ 为假。
* 行为 b 不满足属性 P，当且仅当存在一个前缀不满足 P。

如果一个行为 b 不满足一个安全属性 P，那么存在着一个最短的 b 的前缀 s^{min}，这个前奏不满足属性 P[5]。如果 s^{min} 含有 2 个或者更多的状态，我们认为 s^{min} 的最后一步就是违反属性 P 的那一步。如果 s^{min} 只含有一个单一的状态，我们认为这表示 b 的起始状态就不满足属性 P。考虑安全性属性的一种方法是，如果一个行为违反了安全性属性，我们可以在其行为序列中指出第一个违反的地方。TLA+ 中，任何一个 $Init /\backslash\ [][Next]_{vars}$ 是一个规范的安全性属性[6]。

活性 (Liveness)

行为 b 被称为是有限序列 s 的一个扩展 (Extension), 当且仅当 s 是 b 的一个前缀。如果任何一个有限状态序列都可以通过扩展并且最后满足一个属性 P，我们称这个属性 P 为**活性属性（Liveness)**。范式 <> $x = 3$ 是一个活性属性，因为任何一个有限的状态序列都可以被扩充，然后满足 $x = 3$，例如，只需将 x 等于 3 的状态附加到一个有限序列的后面，然后后面再跟随任意无限的状态序列即可。

安全性和活性

任何属性都可以表示为安全性属性和活性属性的结合。这个结果，以及对活性的精确定义，来自于 Alpern 和 Schneider[2]。然后由于他们并不认为所有属性都对 stuttering 不敏感，因此他们对安全性

[4]比较简单直接，因为对于 b 的所有的前缀 s，$s \models P$ 都为真
[5]或者理解为第一个出现的不满足属性 P 的序列前缀。
[6]一旦违反，从命题逻辑的角度，该规约为假。

的定义与 TLA+ 的上述定义有所不同。对于 stuttering 不敏感性的属性，两个定义是等效的。

3 公平性 (Fairness)

如果 S 是一个安全性属性，L 是一个活性属性，那么我们定义 S 和 L 的组合是“machine-closed” [7]，当且仅当满足下面的条件 [1]: 状态行为的每一个满足 S 的有限序列可以被扩充从而满足 $S \wedge L$[8]。

由于 L 是一个行为的活性属性，因此每个有限状态序列 s 都可以扩展从而满足 L。S,L 的 machine-closed 的机器闭合性意味着如果 s 也满足 S，则可以选择扩展从而同时满足 S 和 L。

从 TLA+ 规约的角度，公平性一词一直被以多种不同的方式使用着。它通常是指关于如何编写规范。该术语最常见的用法是关于进程执行上的公平调度；但在 TLA+ 中，进程的概念是规约描述的副产物。两个等价的 TLA+ 范式可能看起来像是具有不同进程数目的规约 [4]。我们能想到的，可以覆盖我们所见过的所有用法的关于公平的唯一定义是：如果 L 是 S 的公平属性，当且仅当 S,L 是 machine-closed 的。然后我们才可以把一个活性属性 L 称之为，对于一个行为的公平属性。

假设 S 等于 $Init/\backslash[][Next]_vars$。这其实对于所有的 TLA+ 系统规约都是这样的。如果 L 是一个规约的公平属性，意味着 $S \wedge L$ 必须允许初始状态满足 Init 和之后的任何一步满足 Next 范式的约束[9]。

例如，假设 S 和 L 定义如下：

S = `(x=1) /\ [][\E i \in 2..7 : x' = i*x]_x`

L = `[]<>(x % 2 = 1)`

范式 S 定义为在初始状态为 1，然后每一步总是乘于 2-7 之间的一

[7] 不太容易用中文表达，可以理解为是一个完全性，闭合性，系统可以自洽的概念。

[8] 扩充后，可以同时满足安全性 S 和满足活性 L。

[9] 原文的英文比较晦涩。Lamport 用了”$S \wedge L$ does not disallow any initial state satisfying Init or any step satisfying Next”：does not disallow 其实可以理解为：必须保证。

个倍数，或者不变 stuttering 的状态。范式 L 定义为有无限多个 (Infinitely Many) 奇数的存在。显然，公式 L 是一个 Liveness 活性属性，因为任何有限的状态序列都可以通过附加无穷多个 x 等于奇数的状态来扩展，从而满足 L 范式。然而，L 不是一个对于安全属性 S 的公平属性，因为如果 s 包含将 x 乘以偶数的步骤，则满足 S 的有限状态序列 s 不能扩展为满足 $S \wedge L$ 的行为[10]。如果要让范式 L 为真，则不能允许 S 的下一状态动作出现 x 乘以 2、4 或 6 的步骤。

一个非 machine-closed 的规约是很难理解的，因为无法通过查看 NEXT 的操作来判断是否允许执行某个步骤。一个编写的系统规范应该是 machine-closed 的。但是，在验证一个规范是否实现了另一个规范时，有时需要使用非 machine-closed 的公式 [3]。

4 弱公平性 (WF) 和强公平性 (SF) 算子

在 TLA+ 中刻画一个公平属性的标准方法是使用 WF 和 SF 运算符。在定义这些运算符之前，我们需要定义一些符号和概念。在 TLA+ 中，一个动作范式 A 是关于步骤 (Step) 的谓词或者说断言[11]。我们说一个步骤是一个 A 步骤，当且仅当 A 在该步骤上为真。范式 ENABLED A 被定义为，对于一个状态 u，（由于 A 的规范的作用），存在一个状态 w，从而可以使得 u 被更新到 w 状态，即 $u \rightarrow w$。对于动作规范 A，动作 «A»_v 被定义为等于 A /\ $(v'/=v)$。最后，我们将行为 $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow ...$ 的后缀定义为: 行为 $s_i{\rightarrow}s_{i+1}{\rightarrow}s_{i+2}{\rightarrow}...$，其中 i 为正整数。

我们现在定义 WF_v(A) 和 SF_v(A)[12]。状态表达式 v 是用来确保范式 A 对 stuttering 不敏感，可以不做任何状态更新。我们定

[10] 一旦 x 乘于一个偶数，满足 S 的 s 的状态序列会一直是偶数，因为偶数乘于任何自然数，仍然是偶数。

[11] TLA Action 可以理解为是含有 primed 变量的赋值表达式，例如，x'=x+1。通过这个来描述状态更新的规则。

[12] WF: Weak Fairness，弱公平性; SF: Strong Fairness，强公平性。

义一个行为 b 满足 WF_v(A) 当且仅当如果满足以下任何彼此等价的条件。如果读者不太习惯考虑无限集合和序列，这些条件之间的等价性可能有点不容易理解。但当很清楚的认识到这些等价性，就会比较容易理解 WF_v(A) 的含义。

- b 的任何一个后缀序列，如果其状态一直满足 ENABLED $<< A >> \_v$，则一定包含一个 $<< A >> \_v$ 步骤。
- b 不包含没有 $<< A >> \_v$ 步骤的后缀序列，而这个无限的后缀序列其状态都满足 ENABLED «A»_v。
- b 的任何后缀必然包含 $<< A >> \_v$ 步骤，或者就是存在一个状态，ENABLED $<< A >> \_v$ 不满足等于 false 的状态。
- 如果 b 包含一个所有状态都满足 ENABLED $<< A >> \_v$ 的后缀，则 b 包含无限多个 $<< A >> \_v$ 步骤[13]。
- b 一定是包含无限多个 $<< A >> \_v$ 步骤或无限多个不满足 ENABLED $<< A >> \_v$ 的状态[14]。

关于 SF，我们通过定义行为 b 满足 SF_v(A) 当且仅当行为 b 满足以下的等价条件:

- 包含无限多个满足 ENABLED $<< A >> \_v$ 的状态的 b 的任何后缀状态序列都包含一个 $<< A >> \_v$ 步骤[15]。
- b 不存在一个有无限多个满足 ENABLED $<< A >> \_v$ 的状态但却没有 «A»_v 步骤的后缀。
- b 的任何后缀都必须包含一个 $<< A >> \_v$ 步骤或具有一个后缀，其中 ENABLED $<< A >> \_v$ 在其所有状态中都等于假 (false)。
- 如果 b 包含无限多个满足 ENABLED $<< A >> \_v$ 的状态，则

[13] 这个无限多个，对于理解 Weak Fairness 很重要。一旦无限（Always）使能 (ENABLED)，那么一定会出现无限多个 (Infinitely Many) 的 A 步骤发生。并且是改变了状态 v 的步骤，而非单纯的 stuttering。

[14] 如果含有有限个不满足 Enabled 的状态，那么在最后那个不满足的状态之后，就是永远 Always 满足，那么就可以推导出来，b 一定会有无限多个 A 步骤。

[15] 这个条件与 WF 的区别是：满足 WF 的后缀序列需要一直 (Always) 满足 ENABLED 的断言; 而对于 SF 只要有无限多个，即使断断续续，就可以导致动作 A 所包括的状态变化发生。

它包含无限多个 $<< A >> \_v$ 步骤。

显然，任何满足 SF_v(A) 的行为也满足 WF_v(A)[16]。
对于一个 TLA+ 的动作范式 A，WF_v(A) 和 SF_v(A) 是一个活性属性，因为根据活性属性的定义，对于一个以状态 s 结尾的任何有限状态序列，如果 ENABLED«A»_v 为真，那么我们可以向序列添加一个状态 t 使得 $s \to t$ 满足 $<< A >> \_v$。而且我们可以永远继续下去，直到我们遇到 ENABLED $<< A >> \_v$ 为假的状态。在这种情况下，我们可以通过添加无限多个状态 s 来完成这个扩充序列的过程。因为结果序列满足 <>[] ẼNABLED«A»_v，因此这意味着 WF_v(A) 和 SF_v(A) 为真。

【Lamport 在 2021 年 5 月 30 日与笔者的讨论】
”WF_v(A) and SF_v(A) are liveness properties for any A and v because for any finite sequence of states ending in a state s, if ENABLED $<< A >> \_v$ is true, then we can add to the sequence a state t such that $s \to t$ satisfies$<< A >> _v$. We can continue this forever unless we reach a state s in which ENABLED $<< A >> \_v$ is false. In that case, we can complete the sequence by adding infinitely many states s because the resulting sequence satisfied <>[] ẼNABLED$<< A >> _v$ which implies WF_v(A) and SF_v(A) are true.”

然而，WF_v(A) 和 SF_v(A) 不一定是公平属性。例如，让 S 等于 $(x = 0)/\backslash[][x' = x + 1]\_x$，让 A 等于 x’=x+2 并且让 v 等于 x。$<< x' = x + 2 >> \_x$ 步在 S 的任何可达状态下都是可能的，但这样的步骤是不满足递增 x’=x+1。(S 的可达状态是指满足 S 的行为状态。)。因此，不存在能够满足 S 的有限状态序列能够扩展

[16] 如果 A 是 Infinitely Many 的可能（Enabled），就一定会有 Infinitely Many 的出现。因此，如果是一直可能，那么当然一点会有 Infinitely Many 的出现。

到同时支持 S 和 WF_v(A) 或同时支持 S 和 SF_v(A)[17]。
现在我们将讨论范畴限制在标准情况下,S 等于 Init /\[][Next]_vars
。起初的状态为 Init、未来动作变换范式为 Next 和状态变量表达式为 vars。对于这个安全属性，WF_v(A) 和 SF_v(A) 中的 v 通常等于 vars，但不一定必须是。范式 WF_v(A) 和 SF_v(A) 是 S 的公平属性，当且仅当 $<< A >> \_v$ 是 Next 的子动作 (subaction)，其中动作 B 被定义为 Next 的子动作，当且仅当满足以下条件:
如果 u 是范式 S 可以抵达的一个状态，而且 $u \to v$ 是一个满足范式 B 的步骤, 那么 $u \to v$ 也一定满足 Next 范式。
要注意的是，不管 C 什么动作范式，B 都是 $(B \vee C)$ 的子动作。
理解了为什么如果 «A»_v 是 Next 的子操作,WF_v(A) 和 SF_v(A) 会是安全属性 S 的公平属性，将帮助我们理解 WF 和 SF。下面是关于公平性的证明。

对于满足 S 的任何有限状态序列 s，我们需要证明可以将 s 扩展到满足 WF_v(A) 和满足 SF_v(A)。由于满足 SF_v(A) 的任何行为序列一定也都满足 WF_v(A)[18] 因此将 s 扩展到满足 SF_v(A) 就足够了。我们使用以下算法重复将状态附加到序列中。算法从等于 s 的序列开始，我们假设 u 是 s 序列的最后一个状态。

if `ENABLED <<A>>_v` equals `TRUE` in state u
  then append a state w such that $u \to w$ is an `<<A>>_v` step
    (the truth of `ENABLED <<A>>_v` implies the existence of w)
  else **if** `ENABLED Next` equals `TRUE` in state u
    then append a state w such that $u \to w$ is a `Next` step
      (the truth of `ENABLED Next` implies the existence of w)
    else append u

[17]WF 和 SF 可能会违反 Safety 属性，例如 x'=x+2。如果这个情况发生，WF 和 SF 就只是一个 Liveness 属性。当然这是一种纯逻辑的推理。从 TLA+ 的角度，这种情况就不应该发生。WF 和 SF 必须要以保证一个规约的 Safety 为前提。

[18]参阅 Lamport 1994 年 ACM Transaction 的 TLA 的原始文章。$WeakFairness$ $[]([]executed) \vee (<> impossible)$; $StrongFairness$: $[]([]executed) \vee (<> []impossible)$。因此可见，满足 SF 的行为，一定也满足 WF 的命题逻辑，因为满足 SF 的序列一定也保证 WF 断言所需要的了：1. 无限多的动作执行，或者无限多次的不可能。

如果最后的 else 子句被执行到，那么将会在下一次被再次执行，所以这个序列将永远的重复，进入 stuttering 的局面。$<< A >> \_v$ 是 Next 的子动作的假设意味着在任一个 then 的子句中添加的步骤 $u \to w$ 都是属于 Next 范式的步骤，因此算法构造出来的行为序列 b 一定满足 S。下面我们来看算法如何满足 SF_v(A) 的。如果 ENABLED $<< A >> \_v$ 对于无限多个添加状态为真，则算法添加无限多个 $<< A >> \_v$ 的步骤，因此 b 满足 SF_v(A)。

总的来说，任何有限数量的 WF_v(A) 或者 SF_v(A) 范式的并集，如果每一个都是 NEXT 范式的子行为，那么这个并集是 S 的一个公平属性。证明这一点需要推广用于扩展上述证明中使用的序列 s 的算法，以适用于有限的子动作 A。

这个推论也适用于可数无穷多个 WF 和 SF 公式的“联合”。更准确地说，假设 L 等于 $i \in Nat : F(i)$ ，其中对于 Nat 中的所有 i，每个 F(i) 等于 WF_v(A(i)) 或 SF_v(A(i)) 并且 $<< A(i) >> \_v$ 是 Next 的子操作。那么 L 是 S 的公平属性。证明的基本思想是修改将状态序列 s 扩展为满足有限连接的行为的算法，然后产生满足 L 的行为的算法。

新算法的工作原理如下。通过使用满足 SF_v(A(0)) 的算法找到第一个新状态；使用满足 SF_v(A(0)) /\SF_v(A(1)) 的算法找到第二个新状态；使用满足 SF_v(A(0)) /\SF_v(A(1)) /\SF_v(A(2)) 的算法找到第三个新状态；然后通过枚举，穷举所有的范式。

5 文献

1. Martín Abadi and Leslie Lamport. The existence of refinement mappings. Theoretical Computer Science, 82(2):253–284, May 1991.

2. Bowen Alpern and Fred B. Schneider. Defining liveness. Information Processing Letters, 21(4):181–185, October 1985.

3. Leslie Lamport. Auxiliary variables in TLA+. Web page.

4. Leslie Lamport. Processes are in the eye of the beholder. Theoretical Computer Science, 179:333–351, 1997.