# AEB控制器核心逻辑的形式化验证报告 (Z3)

### 1. 为什么要进行形式化验证？(Why Formal Verification?)

在我们开始之前，一个重要的问题是：我们已经有了SysML模型和Simulink仿真，为什么还需要Z3？

* **测试与仿真的局限性**: 传统的测试和仿真，无论我们设计多少测试用例，都只能覆盖有限的、离散的场景。我们永远无法“穷尽”所有可能的输入组合（例如，TTC可以是无穷多个实数）。因此，测试只能证明**存在Bug**，但永远无法证明**不存在Bug**。
* **形式化验证的威力**: 与测试不同，形式化验证使用严格的数学和逻辑推理，来**证明**一个系统模型在**所有可能**的情况下都满足某个给定的属性。它提供的不是“我们没找到问题”，而是“我们从数学上证明了问题不可能发生”。

我们的目标是：为AEB这个安全关键系统，提供最高等级的质量保证。

### 2. 我们要验证什么？—— 关键安全属性

对于一个AEB系统，最可怕的两种失效模式是：

1. **该刹车时不刹车 (False Negative)**
2. **不该刹车时乱刹车 (False Positive)**

后者，即“幽灵刹车”，会极大地影响驾乘体验和用户信任。因此，我们选择验证以下这个至关重要的**安全属性 (Safety Property)**：

**“系统绝不应该在情况安全时，却错误地执行了全力制动。”**

这个属性保证了系统的行为是可预测且合理的，避免了最危险的误触发场景。

### 3. 验证策略：用Z3寻找“反例”

我们的核心策略是**通过矛盾来证明 (Proof by Contradiction)**。我们不直接让Z3去证明我们的属性是“对的”，而是反过来，我们扮演一个“攻击者”的角色，要求Z3去努力寻找一个能推翻我们属性的**反例 (Counterexample)**。

我们向Z3提出的挑战是：

**“请你（Z3）寻找一组输入（TTC, FBtime等），使得‘系统执行了全力制动’和‘当时情况其实是安全的’这两个条件能够同时成立。”**

这个挑战被我们编码在Python脚本中。

### 4. Python代码解析：将系统逻辑翻译为数学约束

我们的veri.py脚本精确地执行了上述策略，它主要包含以下几个步骤：

#### 4.1 定义变量

我们将AEB控制器的所有相关输入、输出和参数，都定义为Z3可以理解的数学变量（在这里是Real实数）。

TTC = Real('TTC') # 碰撞时间  
FCWtime = Real('FCWtime') # 预警时间阈值  
FBtime = Real('FBtime') # 全力制动时间阈值  
decel = Real('decel') # 最终输出的减速度  
FBdecel = Real('FBdecel') # 全力制动的目标减速度值

#### 4.2 建模控制器逻辑

我们将SysML状态机图中的决策逻辑，使用Z3的If(condition, then\_value, else\_value)函数，翻译成了一个精确的数学表达式。这就像一个嵌套的决策树，完美复现了Stateflow的行为。

# 定义进入全力制动区的条件  
is\_full\_braking\_zone = And(Abs(TTC) < FBtime, TTC < 0)  
  
# ... (其他区域的定义) ...  
  
# 核心决策逻辑：如果满足全力制动条件，则decel等于FBdecel；否则，进入下一层判断...  
controller\_logic = If(is\_full\_braking\_zone, decel == FBdecel,  
 # ... (其他制动级别的逻辑) ...  
 )

#### 4.3 定义“反例”

这是最关键的一步，我们用Z3的语言来描述我们想要寻找的那个“坏”情况。

* **“系统执行了全力制动”**:  
  full\_brakes\_applied = (decel == FBdecel)
* “当时情况其实是安全的”:  
  这在逻辑上等同于“不满足进入全力制动状态的条件”。我们使用Not()来实现。  
  situation\_is\_safe = Not(And(Abs(TTC) < FBtime, TTC < 0))
* **最终要检查的“坏”情况**:  
  property\_to\_check = And(full\_brakes\_applied, situation\_is\_safe)

#### 4.4 请求Z3求解

最后，我们将所有这些规则（控制器逻辑、参数值、以及我们想找的“坏”情况）全部交给Z3求解器，然后命令它开始计算。

solver.add(controller\_logic, param\_constraints, property\_to\_check)  
result = solver.check()

### 5. 结果解读：

--- Z3 Solver Result ---  
Result: unsat

**unsat是“Unsatisfiable”（不可满足）的缩写。这个结果的含义是：**

**Z3在穷尽了所有可能的输入组合后，得出结论：我们要求它寻找的那个“坏”情况（即property\_to\_check）在逻辑上是永远无法成立的。描述“全力制动”和“情况安全”的这两个条件，存在根本性的数学矛盾，它们永远不可能同时为真。**

换句话说，Z3没能找到任何一个反例。由于Z3的搜索是完备的，这就从数学上**证明**了我们的安全属性是成立的。

### 6. 结论与项目意义

* 我们获得了什么？  
  我们为AEB控制器的一个核心安全属性提供了形式化的、数学级别的保证。这远比任何数量的仿真测试都更有说服力，极大地增强了我们对设计正确性的信心。
* **对团队的价值**
  + **设计验证**: 证明了我们的SysML状态机逻辑是健全的。
  + **风险规避**: 在开发的极早期阶段，就排除了一类严重的设计缺陷。
  + **知识提升**: 团队掌握了形式化验证这一先进的系统工程技术，提升了整体技术实力。
* **后续步骤与优化建议**
  1. **验证更多属性**: 我们可以继续定义并验证其他属性，例如“活性属性”（Liveness Property），如：“如果TTC持续处于危险区域，系统是否**最终必须**会进入制动状态？”
  2. **作为“黄金参考”**: 这份经过验证的逻辑模型，应作为我们实现Simulink Stateflow和未来编写软件代码的“黄金参考标准”，确保实现与设计严格一致。
  3. **参数敏感性分析**: 我们可以放开对FBtime等参数的固定约束，反过来问Z3：“请问FBtime在什么取值范围内，系统才能保证安全？”，从而指导我们的参数标定工作。

这份验证工作是我们项目的一个重要里程碑，有力地展示了我们采用先进MBSE方法论解决复杂安全关键系统问题的能力。