

Темы, которые будут на экзамене:

Темы, которые будут на экзамене:

\1. Введение. Проблема верификации

\2. Программная логика Флойда-Хоара

\4. Тимпоральные логики

\5. Алгоритмы верификации для логики CTL

\6. Алгоритмы верификации для логики LTL

\8. Структуры Крипке как модели параллельных программ

\9. Спецификация свойств параллельных программ

\11. Бинарные решающие диаграммы (BDD)

\12. Символьная верификация. Неподвижные точки операторов

\13. Вероятностная логика ветвящегося времени

\14. Верификация систем реального времени

ChatGPT 聊天记录

人：我一共发送了16个文件，请你在完成阅读后全面、详细地讲述每个PPT都阐述了哪些内容？请按照这样的格式给出：PPT标题 主要内容大纲 各章节要解决的问题、具体内容

GPT：

请稍等，我将逐一分析并汇总所有16个文件的内容并按照您指定的格式详细说明。

以下是对您提供的16个PPT的全面解读，按照格式分类整理如下：

PPT 1: Введение +

主要内容大纲

- 分布式系统验证的重要性。
- 验证的基本方法。
- Kripke 结构和形式化模型简介。

各章节要解决的问题、具体内容：

1. 什么是分布式系统验证？

- 定义与目标：确保系统满足设计规范。
- 强调形式化验证的重要性，避免高昂的错误代价。

2. 形式化方法简介：

- Hoare 三元组方法。
- 模型检测 (Model Checking) 的应用范围。

3. Kripke 结构：

- 描述系统的状态和转换关系。
- 用作形式化逻辑的基础模型。

PPT 2: Дедуктивная верификация +

主要内容大纲

- Hoare 逻辑和程序正确性证明。
- 弱化和强化条件的使用。
- Floyd 方法的推理逻辑。

各章节要解决的问题、具体内容：

1. 如何使用三元组验证程序？
 - 介绍 $\{P\} \rightarrow S \{Q\}$ 的语法和语义。
 - 演示如何用三元组表达程序的输入、操作和输出之间的关系。
2. 如何证明程序的部分正确性？
 - 通过求解最弱前置条件 (Weakest Precondition)。
 - 示例说明预条件和后条件的推导过程。
3. 方法扩展：
 - 推导强后置条件 (Strongest Postcondition) 以验证复杂程序的正确性。

PPT 3: Темпоральные логики +

主要内容大纲

- 时序逻辑的分类和应用。
- CTL、LTL 公式的语法和语义。
- CTL* 的综合表达能力。

各章节要解决的问题、具体内容：

1. 如何用时序逻辑表达系统行为？
 - LTL 公式适合线性时间系统。
 - CTL 表达树形结构中状态间的关系。
2. 如何结合实际需求选择逻辑？
 - 示例：控制系统的安全性与活性分析。
 - 通过实际案例对比 LTL 和 CTL 的适用性。
3. CTL 的扩展能力：
 - 结合 LTL 和 CTL 的优点，提升表达复杂性的能力。

PPT 4: МС для LTL

主要内容大纲

- 针对 LTL 公式的模型检测方法。
- Kripke 结构与 Büchi 自动机的结合。
- LTL 验证算法的实现。

各章节要解决的问题、具体内容：

1. 如何验证 LTL 公式？
 - 将 LTL 公式转换为 Büchi 自动机。
 - 利用 Kripke 结构描述系统状态。
2. 如何优化模型检测的效率？
 - 使用状态简化和路径搜索策略。
 - 分析复杂系统的路径选择问题。

PPT 5: Автоматические средства верификации (LTL)

主要内容大纲

- 自动工具验证的工作流程。
- SPIN 工具及其使用示例。
- Promela 语言的基础语法。

各章节要解决的问题、具体内容：

1. 如何利用 SPIN 进行验证？
 - 编写 Promela 模型描述系统行为。
 - 配置 SPIN 工具检测 LTL 公式。
2. 实际应用：
 - 验证通信协议和分布式算法的正确性。
3. 结果解释：
 - 读取验证输出，定位潜在错误。

PPT 6: Построение альтернирующего автомата

主要内容大纲

- 构建交替自动机以优化验证。
- 自动机与时序逻辑的联系。
- 交替自动机的状态转换规则。

各章节要解决的问题、具体内容：

1. 什么是交替自动机？
 - 定义：多路径状态转换的自动机。
 - 与 Büchi 自动机的对比。
2. 如何利用交替自动机验证复杂公式？
 - 转换 LTL 公式以减少状态数。
 - 优化验证算法的计算复杂性。

PPT 7: Построение автомата Бюхи

主要内容大纲

- Büchi 自动机的构造方法。
- 在验证中的具体应用。
- 转换和合并状态的规则。

各章节要解决的问题、具体内容：

1. 如何从公式构建 Büchi 自动机？
 - 分步骤示范公式到自动机的转换。
2. 如何通过自动机进行模型检测？
 - 结合 Kripke 结构，验证系统行为的正确性。

PPT 8: Алгоритм MC для CTL

主要内容大纲

- CTL 的模型检测方法。
- 基于 Kripke 结构的标记算法。
- 常见的 CTL 属性验证。

各章节要解决的问题、具体内容：

1. 如何验证 CTL 公式？
 - 使用递归标记算法检查每个状态。
2. 如何应对状态爆炸问题？
 - 通过状态压缩和分区减少计算复杂度。

PPT 9: Символьная верификация CTL

主要内容大纲

- BDD 的基本概念。
- 符号模型检测的方法。
- 克服状态爆炸问题的策略。

各章节要解决的问题、具体内容：

1. 什么是符号验证？
 - 利用布尔函数和 BDD 表示状态集合。
2. 如何提高验证规模？
 - BDD 在大型系统中的高效存储和操作。

PPT 10: Применение темпоральных логик

主要内容大纲

- 时序逻辑在工业中的应用。
- 编写逻辑公式的常见错误和改进方法。

各章节要解决的问题、具体内容：

1. 如何表达因果关系？
 - 示例说明如何用 LTL 表达实时系统行为。
2. 复杂逻辑公式的优化：
 - 减少冗余逻辑，提升公式可读性。

PPT 11: Вероятностная и количественная верификация

主要内容大纲

- PCTL 的定义与应用。
- 概率 Kripke 结构的扩展。
- 定量化验证的案例。

各章节要解决的问题、具体内容：

1. 如何对系统行为进行量化？
 - 用 PCTL 表达概率约束和时序属性。
2. 如何结合概率分析和模型检测？

- 使用马尔可夫链计算路径概率。

PPT 12: Верификация систем реального времени +

主要内容大纲

- 实时系统的时间属性验证。
- 时钟自动机的引入及应用。

各章节要解决的问题、具体内容：

1. 如何定义实时系统?
 - 建立时间限制和系统状态的关联。
2. 如何验证实时系统行为?
 - 利用 UPPAAL 等工具进行模型检测。

PPT 13: Структура Крипке

主要内容大纲

- Kripke 结构的理论与应用。
- 建模系统状态和转换的基础。

各章节要解决的问题、具体内容：

1. 如何构建 Kripke 结构?
 - 从系统需求提取状态和转换关系。
2. 如何用 Kripke 结构验证公式?
 - 通过路径搜索实现公式验证。

PPT 14: BDD и их применение +

主要内容大纲

- BDD 的构造与优化。
- 在 Model Checking 中的具体应用。

各章节要解决的问题、具体内容：

1. 什么是 BDD?
 - 定义和基本构造方法。
2. 如何用 BDD 优化存储?
 - 减少布尔表达的冗余。

PPT 15: Примеры верификации

主要内容大纲

- 具体验证工具的案例分析。
- Spin 和 Promela 的实际应用。

各章节要解决的问题、具体内容：

1. 如何选择验证工具?
 - 根据系统需求选择合适的验证方法。
2. 案例分析:
 - 多个系统验证的成功经验分享。

PPT 16: Количественный анализ

主要内容大纲

- 系统性能的定量化分析。
- 延迟、可靠性等指标的计算方法。

各章节要解决的问题、具体内容：

1. 如何结合模型检测和性能分析？
 - 使用统计方法评价系统设计的鲁棒性。

以上是对所有PPT内容的详细分类和分析。这些课件系统地介绍了分布式系统验证的理论和实际应用方法，为学习者提供了全面的知识框架。

人：请仔细阅读这门课程的考试题目。请说说，考试题目涵盖了哪些内容？可以在哪个PPT的哪个部分中找到对应的解题方法？

GPT：

通过分析考试题目，可以总结出以下涵盖的内容及其对应的PPT章节，提供解题参考：

考试题目涵盖的内容及解题对应章节

1. Kripke 结构与 CTL 公式验证

题目示例：

- 在 Kripke 结构中检查 CTL 公式的满足性，例如 $\text{A}[p \wedge E(q \wedge r)]$ 。

涉及内容：

- 定义 Kripke 结构及其状态转换。
- CTL 公式的语义及验证方法。

解题对应章节：

- PPT 1 (Введение): Kripke 结构基础。
- PPT 8 (Алгоритм МС для CTL): CTL 公式验证的算法。

2. LTL 公式表达与验证

题目示例：

- 将特定系统属性用 LTL 公式表达，例如：
 - 每个警报事件前必须有故障事件。
- 验证两条 LTL 公式是否等价。

涉及内容：

- LTL 公式的语法和语义。
- 使用 Büchi 自动机进行公式验证。

解题对应章节：

- PPT 3 (Темпоральные логики): LTL 公式的基本语法。
- PPT 4 (МС для LTL): LTL 验证方法。
- PPT 7 (Построение автомата Бюхи): Büchi 自动机构造及验证。

3. 时序逻辑与因果关系

题目示例：

- 验证逻辑公式的正确性，例如：
 - $\text{FG}(p \rightarrow q)$ 。
- 判断 CTL 和 LTL 公式的语义差异。

涉及内容：

- 时序逻辑操作符的形式化定义。
- CTL 和 LTL 的比较。

解题对应章节：

- PPT 3 (Темпоральные логики)：时序逻辑的基础。
- PPT 10 (Применение темпоральных логик)：时序逻辑在因果关系表达中的应用。

4. 符号模型检测与状态爆炸问题

题目示例：

- 构建 BDD 以优化状态存储，例如 $f = a \text{ AND } (b \text{ OR } c)$ 。
- 确定符号验证的效率。

涉及内容：

- 符号验证方法及其在大型系统中的应用。
- 使用 BDD 表示布尔函数。

解题对应章节：

- PPT 9 (Символьная верификация CTL)：符号验证方法。
- PPT 14 (BDD и их применение)：BDD 的构造与优化。

5. 实时系统与时钟自动机

题目示例：

- 验证时间约束系统属性，例如 $E[x \leq y \text{ U } @s2(x > y)]$ 。
- 分析系统是否满足特定实时逻辑公式。

涉及内容：

- 时钟自动机的定义和建模。
- 实时系统的验证方法。

解题对应章节：

- PPT 12 (Верификация систем реального времени)：实时系统和时钟自动机的验证。

6. 概率逻辑与 PCTL

题目示例：

- 验证 PCTL 公式，例如 $P>0.5[X(a \cup b)]$ 。
- 分析概率 Kripke 结构中公式的满足性。

涉及内容：

- PCTL 的定义及语法。
- 使用概率 Markov 模型进行分析。

解题对应章节:

- **PPT 11 (Вероятностная и количественная верификация):** 概率 Kripke 结构和 PCTL 验证。

7. Hoare 逻辑与程序正确性

题目示例:

- 使用 Hoare 逻辑验证程序正确性，例如计算阶乘程序的正确性。

涉及内容:

- Hoare 三元组方法。
- 循环不变式和弱前置条件的推导。

解题对应章节:

- **PPT 2 (Дедуктивная верификация):** Hoare 逻辑和程序正确性。

8. Promela 与 SPIN 的应用

题目示例:

- 使用 Promela 语言为系统建模，并用 SPIN 工具验证属性。

涉及内容:

- 使用 Promela 描述系统行为。
- 利用 SPIN 工具验证分布式系统属性。

解题对应章节:

- **PPT 5 (Автоматические средства верификации):** Promela 语言和 SPIN 工具应用。

总结

考试题目全面涵盖了课程的核心知识点，包括时序逻辑、模型检测、符号验证、概率逻辑以及实时系统的验证。每个题目都可以通过上述 PPT 提供的内容和方法进行解决，为学生理解和应用分布式系统验证工具提供了完整的指导。

课程PPT 笔记

1. Введение
2. Метод Флойда-Хоара доказательства корректности программ
3. Исчисление взаимодействующих систем (CCS) Р.Милнера
4. Темпоральные логики
5. Алгоритм Model checking для проверки выполнимости формул CTL
- 6. Автоматный подход к проверке выполнимости формул LTL**
- 7. Система верификации Spin и язык Promela**. Примеры верификации****
8. Структура Крипке как модель реагирующих систем
9. Темпоральные свойства систем
10. Применения метода верификации model checking
11. BDD и их применение

12. Символьная проверка моделей

13. Количественный анализ дискретных систем при их верификации

14. Верификация систем реального времени

15. Консультации по курсовой работе

Дедуктивная верификация

Дедуктивная верификация. Метод Флойда-Хоара доказательства корректности программ

演绎验证。证明程序正确性的 Floyd-Hoare 方法

Дедуктивные методы верификации 演绎验证方法

定义

根据 PPT 2: Дедуктивная верификация 的 第4页:

Дедуктивная верификация – это метод доказательства правильности программ, основанный на логическом выводе, где корректность программы выражается в виде логических утверждений (триплетов Хоара) и доказывается с использованием правил вывода.

翻译如下：

演绎验证方法是一种基于逻辑推理的程序正确性证明方法，通过将程序的正确性形式化为逻辑断言（如 Hoare 三元组）并使用推理规则加以证明。

演绎验证的核心是使用逻辑方法为程序的所有可能执行路径提供数学证明。

1. 方法论（第6页）：

演绎验证方法依赖以下三大核心：

- **Hoare 逻辑**: 提供一组推导规则来验证程序的部分正确性和全局正确性。
- **弱化/强化条件**: 在验证循环和分支语句时，通过推导更弱的前置条件或更强的后置条件逐步推进证明过程。
- **循环不变式**: 在验证循环程序时，定义一个在循环执行过程中保持为真的谓词，用于证明循环体的正确性。

2. Hoare 三元组的验证（第7页）：

PPT 中给出了以下 Hoare 三元组的验证步骤：

1. 定义前置条件 P:
 - 描述程序开始执行前的状态。
2. 推导循环不变式:
 - 保证循环每次迭代后程序状态符合不变式。
3. 证明后置条件 Q:
 - 验证程序终止后，状态满足目标要求。

示例：

```
{N > 0}
f := 1;
x := N;
while x > 0 do
    f := f * x;
    x := x - 1;
end
{f = N!}
```

前置条件 P: $N > 0$: 输入的 N 是一个正整数。

循环不变式 I: $f = (N - x)!$: 每次迭代中保持为真。

后置条件 Q: $f = N!$: 程序终止时, f 存储 N 的阶乘。

3. 方法优缺点 (第9页) :

- **优点:**

- 提供程序正确性的数学证明, 确保严谨性。
- 可用于安全关键系统的验证 (如航空航天、医疗设备软件)。

- **缺点:**

- 对于复杂系统, 手动验证成本高, 可能需要辅助工具。
- 对程序员的逻辑和数学能力要求较高。

什么是“正确的程序”?

「正确的程序」是指一个程序在任何给定输入情况下, 都能够按照预期的逻辑正确地执行, 并满足给定的功能需求和行为规范。正确性通常包括两方面:

1. **部分正确性 (Partial Correctness):** 程序在终止时, 输出结果满足后条件。

2. **全局正确性 (Total Correctness):** 除了部分正确性, 还要求程序能够在有限时间内终止。

PPT中的定义

在 **PPT 2: Дедуктивная верификация** 的 第8页 提到:

Программа считается правильной, если для всех допустимых входных данных её выполнение завершается корректным результатом, соответствующим спецификации.

翻译如下:

如果程序在所有允许的输入下, 其执行都能返回与规格一致的正确结果, 则认为该程序是正确的。

「正确的程序」的数学表达

根据 Hoare 逻辑, 用三元组 $\{P\} \ S \ {Q}$ 表示程序的正确性:

- P: 程序的前置条件。
- S: 程序语句。
- Q: 程序的后置条件。

如果程序 S 在满足 P 的情况下执行, 结果能够满足 Q , 则程序被认为是部分正确的。如果进一步证明 S 一定能终止, 则是完全正确。

「正确的程序」的数学表达, 示例

谓词 (предикат)

- 值域只有T、F两个取值的函数称为谓词。例如 $a > b$ 就是一个谓词。
- 谓词可以描述一个集合的子集。比如谓词 $x > y$ 描述了 $\{x, y\}$ 这个集合内所有满足 $x > y$ 的数对。比如 $\{5, 3\}$ 、 $\{10, 6\}$... 等等。

状态、程序、变换性的、需求(требование)

- 状态就是变量值组成的向量组
- 程序就是在状态集合的转换器
- 这样的程序称之为变换性的
- 需求规定了程序的初始状态和最终状态应该是什么样的

Спецификация (описание) требований к программе 程序需求的规范化描述

规格说明是对程序需求的形式化描述：包括程序的初始条件和期望的运行结果，用于明确程序在正确运行时必须满足的功能要求。

规格说明的组成部分：

1. 前置条件 I (предусловие, Pre-condition) :

- 谓词，该谓词给出了我们能保证程序正确工作的所有值
- 比如，对于程序“提取平方数”来说，前置条件就是 $x \geq 0$

2. 后置条件 R (постусловие, Post-condition) :

- 谓词，该谓词定义了正确的结果，也就是程序运行后我们的期望结果。

需求与具体算法无关。

比如：

求最大公约数的需求可以表示为：

$I := \{ (m, n \in \mathbb{N}) \};$

$R := \{ H = \text{最大公约数}(m, n) \}$

Для программы P , вычисляющей НОД, сформулируем УТВЕРЖДЕНИЕ: “Если m и n – любые натуральные числа до выполнения программы P , то последовательность P переменная H будет равна НОД(m, n)”.

部分正确性、完全正确性

部分正确性

Частичная (partial) корректность программы: $\{ I \} P \{ R \}$:

"Если состояния программы P удовлетворяют предусловию I до начала работы программы,

И программа после P завершится,

ТО после завершения P состояния программы будут удовлетворять постусловию R ".

翻译：“如果程序P在开始前满足前置条件I，且程序P能够结束的条件下，那么结束程序P的时候状态将会满足后置条件R”

完全正确性

Полная (тотальная, total) корректность программы: $[I] P [R]$:

"Если предусловие I истинно до начала работы программы S ,

ТО P завершится,

И после завершения программы P будет истинно постусловие R "

翻译：“如果前置条件I在运行程序前为真，那么P一定能结束，且结束时必有后置条件R为真。”

部分正确性和完全正确性的关系

Total correctness = Partial correctness + Termination

霍尔三元组 (Hoare Triple)

假设存在一个程序 P , 前置条件 I , 后置条件 R (assertions), 那么 $\{ I \} P \{ R \}$ 就是一个霍尔三元组。

$\{ I \} P \{ R \}$ 这个三元组读作：只要 I 在 P 执行之前成立，则执行之后 R 也成立。注意如果程序 P 不终止，那就没有“之后”了，所以 Q 在此时可以是任何语句，甚至可以为假。

Утверждение $\{ I \} P \{ R \}$: Программа P частично корректна относительно предусловия I и постусловия R

翻译：霍尔断言 $\{ I \} P \{ R \}$: 程序 P 相对于前条件 I 和后条件 R 部分正确。

怎样证明断言的真实性(永远为真)?

换言之，怎样证明，对于任意满足谓词 I 的初始值，在执行程序 P 之后的状态都能保证满足谓词 R ?

谓词之间的关系

谓词之间的关系包括：等价关系、包含关系、互斥关系

谓词之间的运算包括：合取、析取

- 等价关系：

- 定义：如果在所有可能的程序状态下 P_1 和 P_2 的真假值总是相同，则谓词 P_1 和 P_2 等价。
- 表示： $P_1 \Leftrightarrow P_2$ 。

- 包含关系 / 强弱关系：

- 定义：如果 P_1 为真， P_2 一定为真
- 表示： $P_1 \Rightarrow P_2$ 表示 P_1 的真包含 P_2 的真。
- 也可以表述为 “ P_1 比 P_2 更强”
- 小技巧：小范围推大范围，所以 P_1 是小范围， P_2 是大范围！谁在前面谁更强！

- 一些特例：

- False 是最强的条件 (最强谓词)
- True 是最弱的条件 (最弱谓词)
- 记忆：至贱则无敌，真理是柔弱的。

sp (最强后置条件)

最强后置条件 strongest postcondition (sp) сильнейшее постусловие

弗洛伊德认为，程序是谓词的变换器。

以下我的理解，有点长：

之所以说程序是谓词变换器，是因为程序在运行之前满足一个谓词I，运行之后满足另一个谓词R。对于系统状态集合来说，程序使满足的谓词从I变成了R。（应该不是很充分，后续再研究下）

假设系统的初始状态分布在一个抛物线 $y = x^2 - 6x + 2$ 上，经过程序 $\{x += 1; y -= 2\}$ 之后，系统的最终状态分布到了另一个抛物线 $y = x^2 - 8x + 7$ 上。也就是说，系统满足的谓词从 $y = x^2 - 6x + 2$ 变成了 $y = x^2 - 8x + 7$ 。这种变化是由程序造成的。

那么我们知道了这个可以做什么？

我们现在知道了，程序可以让谓词A变成谓词B，如果我们把谓词替换成“条件”（这样的替换是合理的，因为程序的输入输出通常就是谓词，比如输入条件：x, y都是自然数，输出条件： $x=y+3$ ），那么这句话也可以等价为：程序可以让状态从满足条件A变为满足条件B。

结合最强后置定理：如果程序的最强后置条件 sp 可以推导出最终满足的状态 R，那么，这个程序必然满足部分正确性。

我们可以找到一个判断程序是否正确的方法！

同样用上面的抛物线举例。假设我现在有 前置状态 $y = x^2 - 6x + 2$ ，后置状态 $y = x^2 - 8x + 7$ ，和程序 $\{x += 1; y -= 2\}$ 。我想要知道这个程序能否满足我的需求？即经过程序的运行之后，所有满足前置状态的点都能分布到后置状态规定的集合里去。

一开始可能无从下手，因为我不可能对无限多的测试数据做验证。但是最强后置定理却给了我们一个从数学上证明程序正确性的方法。

根据最强后置定理，我只需要判断：程序的最强后置条件 sp 是否可以推导出最终状态 $y = x^2 - 8x + 7$ 即可（即，sp 谓词比最终状态谓词“强”，形式化表示： $sp \Rightarrow y = x^2 - 8x + 7$ ）

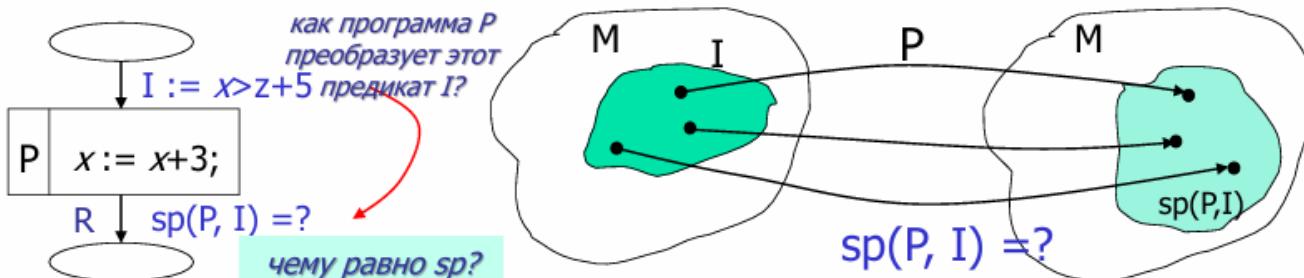
所以要做的事情非常清晰了，步骤如下：

- 求出 sp
- 判断 $sp \Rightarrow y = x^2 - 8x + 7$ 是否永远成立

如果 $sp \Rightarrow y = x^2 - 8x + 7$ 永远成立，说明程序永远正确。即对于任何满足初始条件的输入，我们都能100% 保证能得到满足最终状态的值。

我们用数学的力量，避免了对无限多数据的验证！

我的理解结束。 ◻



程序P如何把谓词I变为谓词R?

定义：程序P的最强后置条件 (strongest postcondition, sp) 就是一个谓词，它描述了所有可能的**终止状态**，这些状态由程序 P 从满足初始条件 I 的初始状态转换而来。

强后置条件只取决于程序 P 和初始条件 I，与后条件R无关。

对于上图中 $I := x > z + 5$ 的例子：

程序结束之后，所有 x 的值都自增3，所以如果 $I := x > z + 5$ 满足，那么 $sp(P, I) = x > z + 8$

部分正确性定理

如果存在程序P和3个谓词：

I - 前条件，程序在运行前应该满足的条件

R - 后条件，程序结束后的期望满足条件

$sp(P, I)$ - **最强后条件**，他描述了所有可能的 终止状态，这些状态由程序 P 从满足初始条件 I 的初始状态转换而来。

那么这条定理成立：如果 $sp(P, I) \Rightarrow R$ ，则 $\{I\} P \{R\}$ 。

证明略。

如何证明程序P的正确性？

步骤：

1. 找到最强后条件 $sp(P, I)$
2. 检查 $sp(P, I) \Rightarrow R$ 是否成立

由此我们可以得出：

- $\{\text{False}\} P \{\psi\}$ - 如果前置条件是 False，那么对于任何程序 P，它的后置条件都是成立的

这是一个永真三元组，因为前置条件是 False，意味着程序 P 永远不会被执行，所以任何后置条件都无所谓，所以这个三元组在任何情况下都是成立的

- $\{\phi\} P \{\text{True}\}$ - 对于任何程序 P 和任意的前置条件 ϕ ，当程序 P 执行完毕后，后置条件“True”总是成立

这个三元组也是永真的，因为后置条件是 True，表示任何结束状态都是符合的。因此，**无论前置条件或程序的具体形式如何**，后置条件为“真”总是成立的

如何求取程序的最强后条件 $sp(P, I)$ ？

需要根据程序的复杂程度分情况讨论：

- 赋值语句
- 顺序结构
- 分支结构
- 循环结构

(1) 赋值语句

- 形式： $x := e$

- 规则：

- 将初始条件 I 中的变量 x 替换成表达式 e

- 计算方法：

- $sp(x := e, I) = I[x \mapsto e]$

- 其中 $I[x \mapsto e]$ 表示将 I 中所有出现 x 的地方替换成 e

- 例子：

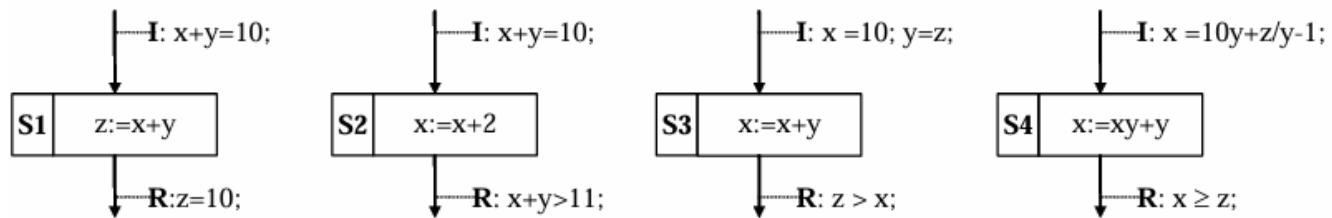
```
{x > 0}
x := x + 1
```

- 初始条件: $I: x > 0$
- 赋值后 $sp(x := x + 1, x > 0) = x - 1 > 0$
- 化简得 $sp = x > 1$

计算步骤 (我个人的理解) :

$$\begin{aligned}
 & sp(x := x_0 + 1, x_0 > 0) \\
 \Leftrightarrow & (x := x_0 + 1) \wedge (x_0 > 0) \\
 \Leftrightarrow & x - 1 > 0 \\
 \Leftrightarrow & x > 1
 \end{aligned}$$

判断程序是否满足部分正确性的例子:



我的求解过程:

11:13 12月14日周六

(1)

① 求解 sp

$$\begin{aligned}
 & sp(z := x+y, x+y=10) \\
 \Leftrightarrow & (z := x+y) \wedge (x+y = 10) \\
 \Leftrightarrow & z = 10
 \end{aligned}$$

② 判断 $sp \Rightarrow R$?

$$\begin{aligned}
 & (z = 10) \Rightarrow (z = 10) ? \\
 & \text{Yes.}
 \end{aligned}$$

③ 程序部分正确性满足 //

(2)

① $sp(x := x_0 + 2, x_0 + y_0 = 10)$

$$\begin{aligned}
 & \Leftrightarrow (x := x_0 + 2) \wedge (x_0 + y_0 = 10) \\
 & \Leftrightarrow (x - 2 + y_0 = 10) \\
 & \Leftrightarrow x + y_0 = 12 \\
 & \Leftrightarrow x + y = 12
 \end{aligned}$$

② $(x + y = 12) \Rightarrow (x + y > 11)$?

Yes

③ 程序部分正确性满足 //

(3)

① $sp(x := x_0 + y_0, x_0 = 10 \wedge y_0 = z_0)$

$$\begin{aligned}
 & \Leftrightarrow (x - z_0 = 10) \wedge (y_0 = z_0) \\
 & \Leftrightarrow x - z_0 = 10 \\
 & \Leftrightarrow x - z = 10 \\
 & \Leftrightarrow x = 10 + z
 \end{aligned}$$

② $sp \Rightarrow R \Leftrightarrow (x = 10 + z) \Rightarrow (z > x)$

$$\begin{aligned}
 & \Leftrightarrow z > 10 + z \Rightarrow \text{False} \\
 & \text{③ 故不成立 //}
 \end{aligned}$$

(4) ① $sp(x := x_0 y_0 + y_0, x_0 = 10y_0 + z_0 / y_0 - 1)$

$$\begin{aligned}
 & \Leftrightarrow \left(x_0 = \frac{x - y_0}{y_0} = \frac{x}{y_0} - 1 \right) \\
 & \Leftrightarrow \frac{x}{y_0} > x = 10 \cdot y_0 + \frac{z_0}{y_0} > x
 \end{aligned}$$

(4) ① $sp (x := x_0 y_0 + y_0, x_0 = 10y_0 + z_0 / y_0 - 1)$

$$\Leftrightarrow \left(x_0 = \frac{x - y_0}{y_0} = \frac{x}{y_0} - 1 \right)$$

$$\Leftrightarrow \frac{x}{y_0} - 1 = 10 \cdot y_0 + \frac{z_0}{y_0}$$

$$\Leftrightarrow \frac{x - z_0}{y_0} = 10y_0$$

$$\Leftrightarrow x = 10y^2 + z$$

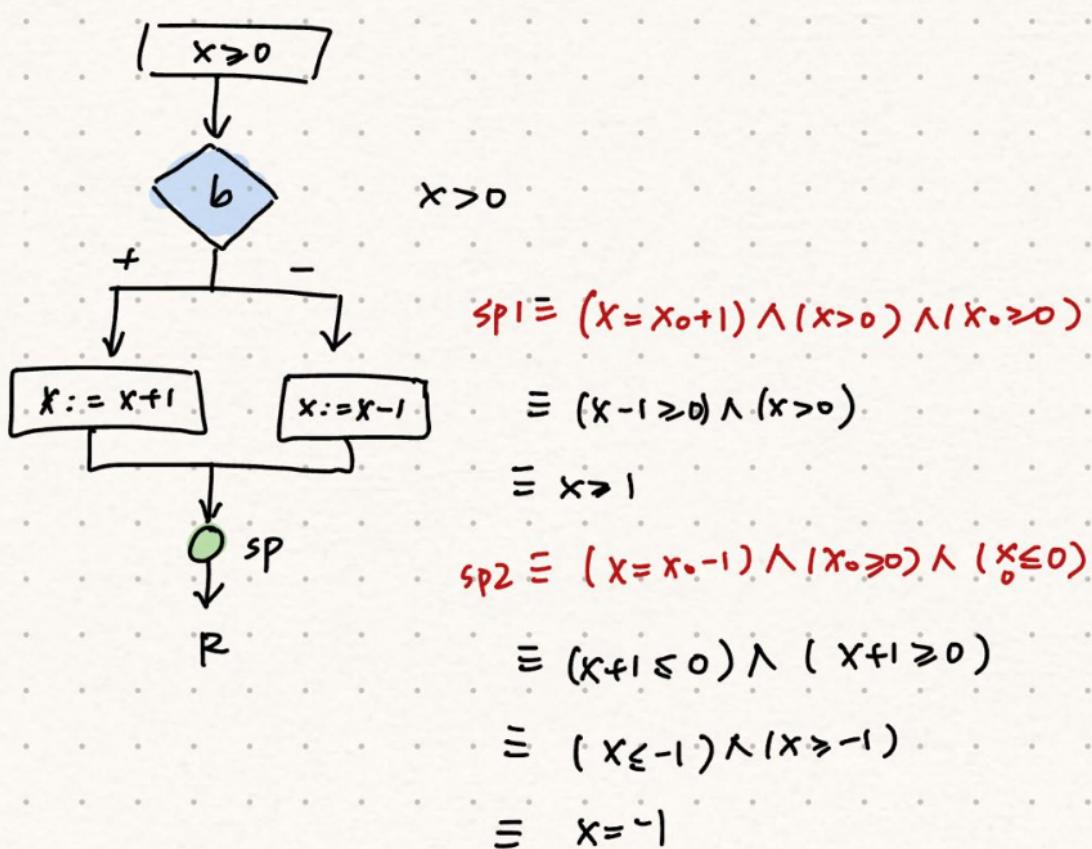
② $(x = 10y^2 + z) \Rightarrow x \geq z$

$$\Leftrightarrow 10y^2 + z \geq z$$

$$\Leftrightarrow 10y^2 \geq 0 \Rightarrow \text{True}$$

③ 程序部分正确性满足 //

答案：（见PPT[[Модуль 2. Дедуктивная верификация.pdf] p32]



$$sp = sp1 \vee sp2 = (x \geq 1) \vee (x = -1) //$$

(2) 顺序结构

- 形式: $P_1; P_2$ (两个程序顺序执行)
- 规则:
 - 先计算第一个子程序的最强后置条件 $sp(P_1, I)$
 - 将其作为第二个子程序的前条件
 - 最终得到的结果是整个顺序结构的最强后条件
- 计算方法:
 - $sp(P_1; P_2, I) = sp(P_2, sp(P_1, I))$

示例:

```
I = {y + x > z + 1}
x := 2z + x;
y := y - x;
```

我的求解步骤:

① 求解 $sp([s_1; s_2], I) \equiv sp([s_2], sp([s_1], I))$

先求 $sp([s_1], I)$

$$sp(x := 2z_0 + x_0, y_0 + x_0 > z_0 + 1)$$

$$\Leftrightarrow y_0 + (x - 2z_0) > z_0 + 1$$

$$\Leftrightarrow x + y > 3z + 1$$

再代回到 $sp([s_2], sp(\dots))$

$$sp([s_2], \dots) \Leftrightarrow sp(y := y_0 - x, x + y_0 > 3z + 1)$$

$$\Leftrightarrow x + (y + x) > 3z + 1$$

$$\Leftrightarrow y + 2x - 3z > 1$$

② $sp(\dots) \Rightarrow R$ 是否成立?

$$(y + 2x - 3z > 1) \Rightarrow (y + 2x > 3z) \equiv \text{True}$$

③ 因此根据部分性正确定理可以判断, 程序P对于初始条件I, 满足部分性正确 //

答案: (见PPT[[Модуль 2. Дедуктивная верификация.pdf] p37)

$I = \{ y+x > z+1 \}$ *Последовательно вычисляем sp:*

s1: $x := 2z + x;$
s2: $y := y - x;$

$R = \{ y + 2x > 3z \}$

1. Вместо X в I подставляем $x - 2z$

$sp([s1], I) = y + (x - 2z) > z + 1$

2. Вместо y в $sp([s1], I)$ подставляем $y+x$

$sp([s1; s2], I) = sp([s2], sp([s1], I)) =$

$(y + x) + (x - 2z) > z - 1 \equiv [y + 2x - 3z > 1]$

Программа частично корректна, iff $sp([s1; s2], I) \Rightarrow R$

$y + 2x - 3z > 1 \Rightarrow (?) y + 2x > 3z$ - проверить для всех $x, y, z!$

$y + 2x - 3z > 1 \Rightarrow y + 2x - 3z > 0$ - да, для всех x, y, z выполняется

Программа частично корректна!

37

如果程序由多个子程序组成: $[s1; s2; s3; s4; s5]$

那么: $sp([s1; s2; s3; s4; s5], I)$

$\equiv sp(s5, sp([s1; s2; s3; s4]))$

$\equiv sp(s5, sp([s4], [s1; s2; s3]))$

$\equiv \dots$

$\equiv sp(s5, sp([s4], sp([s3], sp([s2], sp([s1], I))))))$

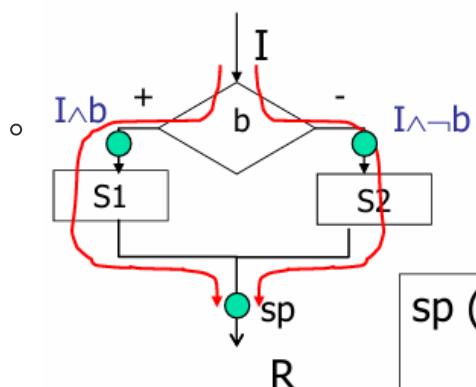
用自底向上的方法, 先求最里面的 $sp([s1], I)$, 再求 $sp([s2], \dots)$, 再求 $sp([s3], \dots)$, 再求 $sp([s4], \dots)$, 最后求 $sp([s5], \dots)$ 。

(3) 分支结构

- 形式: if C then P1 else P2
- 规则:
 - 计算分支条件成立时的最强后置条件 $sp(P1, I \cap C)$
 - 计算分支条件不成立时的最强后置条件 $sp(P2, I \cap \text{not } C)$
 - 将两个结果的析取作为整个分支结构的最强后置条件
- 计算方法:
 - $sp(\text{if } C \text{ then } P1 \text{ else } P2, I) = sp(P1, I \cap C) \cup sp(P2, I \cap \text{not } C)$
- PPT 中的位置:

Формальная семантика оператора выбора

Программа P корректна,
iff $sp(P, I) \Rightarrow R$



Семантика оператора выбора

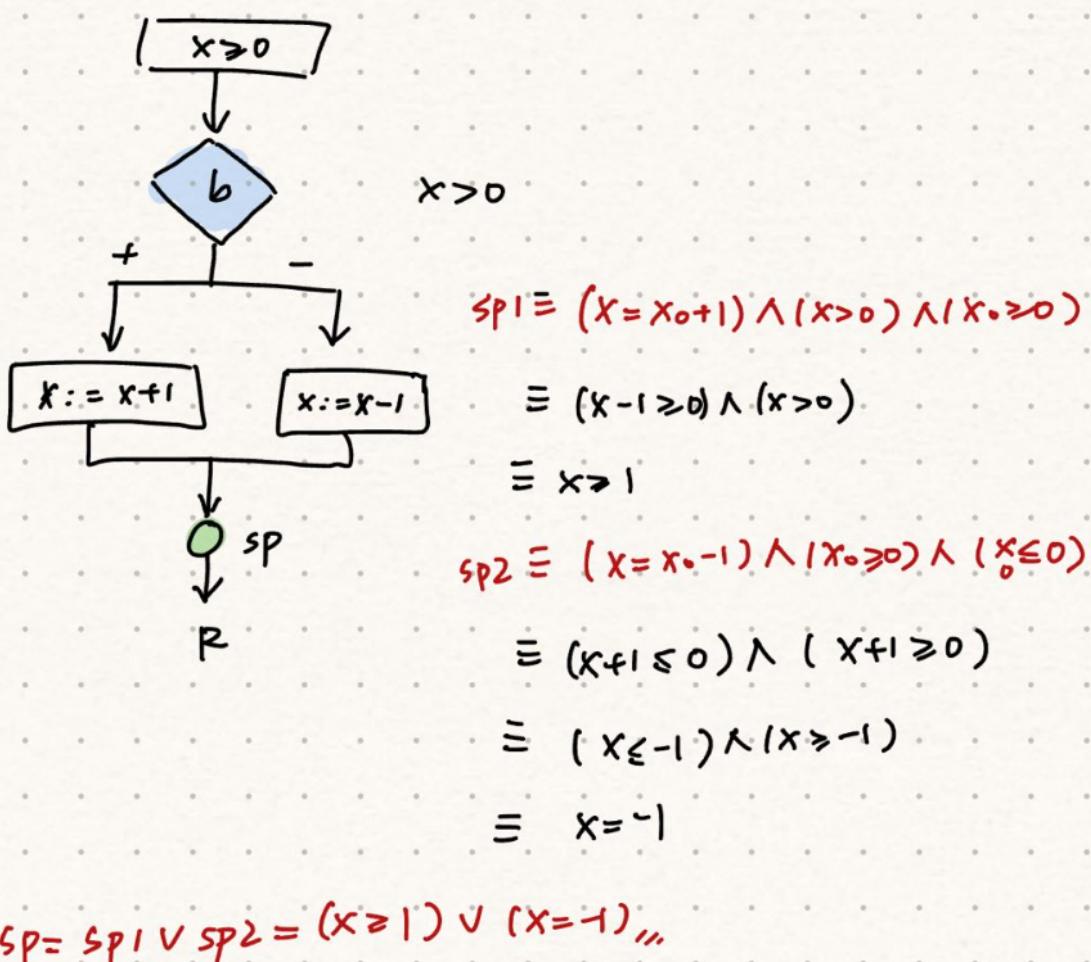
$P = \text{if } b \text{ then } S1 \text{ else } S2 -$

сильнейшее постусловие $SP (P, I)$:

$$sp ([\text{if } b \text{ then } S1 \text{ else } S2] , I) = \\ sp (S1, I \wedge b) \vee sp (S2, I \wedge \neg b)$$

- 例子：计算这个程序的最强后置条件

```
P = if x > 0
    then x := x + 1
    else x := x - 1
```

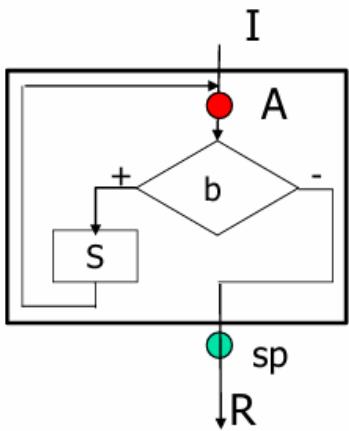


(4) 循环结构

- 形式: while C do P
- 规则:
 - 循环结构的最强后置条件需要依赖循环不变式 I_{inv} (invariant)
 - invariant цикла - утверждение, которое остается истинным при всех прохождениях цикла
 - 循环终止时的条件为 $\neg C$, 因此终止状态应满足 $I_{inv} \wedge \neg C$
- 计算方法
 - 定义循环不变式 I_{inv} , 确保 $sp(P, I_{inv}) \Rightarrow I_{inv}$
 - 最强后置条件为: $sp(\text{while } C \text{ do } P, I) = I_{inv} \wedge \neg C$
- PPT中的表述:

Формальная семантика оператора цикла

Как посчитать сильнейшее постусловие для цикла?



$sp([\text{while } b \text{ do } S \text{ od}], I) = // \text{ сколько раз цикл?}$

$\neg b \wedge I \vee$
 $\neg b \wedge sp(S, b \wedge I) \vee$
 $\neg b \wedge sp(sp(S, b \wedge I)) \vee$
...
=

обозначим: $L_0(I) = I // 0 \text{ раз цикл}$
 $L_{k+1}(I) = sp(S, b \wedge L_k(I)) // k+1 \text{ раз цикл}$

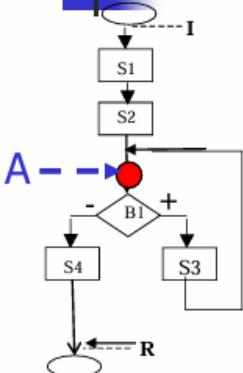
тогда:

= $\neg b \wedge (\exists k \in \mathbb{N}) L_k(I) // \text{ общая формула:}$
0 или больше раз выполнился цикл

Как быть?

Инвариант цикла – утверждение, которое остается истинным при всех прохождениях цикла

Инвариант цикла: утверждение о поведении программы в цикле



Для доказательства частичной корректности оператора цикла нужно ПРИДУМАТЬ **инвариант** – утверждение, которое **остается истинными при каждом прохождении цикла**.

Если инвариант цикла А построен, то доказательство циклической программы сводится к доказательству трех теорем:

T1: $sp([s1; s2], I) \Rightarrow A$ // А выполняется до входа в цикл

T2: $sp([B1+; s3], A) \Rightarrow A$ // А вып. в каждом прох. цикла

T3: $sp([B1- ; s4], A) \Rightarrow R$ // если цикл завершился, то R вып.

Инварианты отражают идею циклического алгоритма.

Для того чтобы придумать правильный инвариант, необходимо проникнуть в замысел разработчика алгоритма, **понять суть алгоритма**.

Если невозможно доказать частичную корректность циклической программы, то или программа неверна, или неверно придуманы утверждения на стрелках программы (инварианты циклов).

45

- 如果构造了循环不变量 A，则循环程序的证明就简化为三个定理的证明：

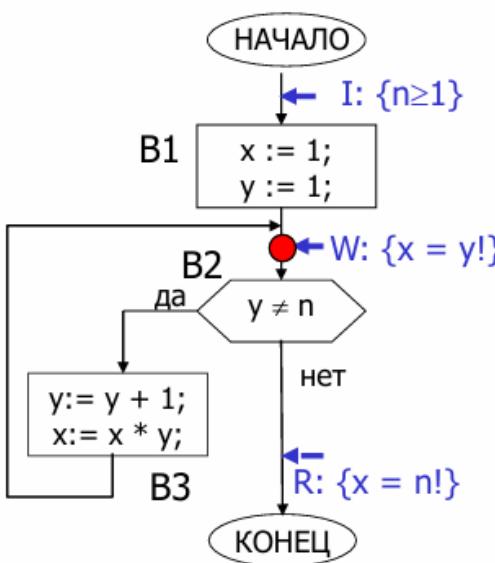
- T1: $sp([s1; s2], I) \Rightarrow A$
- T2: $sp([B1+; s3], A) \Rightarrow A$
- T3: $sp([B1- ; s4], A) \Rightarrow R$

- 如果无法证明循环程序的部分正确性，那么要么该程序不正确，要么程序箭头上的语句（循环不变量）被错误地编写。

示例：证明一个阶乘程序的正确性

Доказательство частичной корректности программы вычисления факториала

Доказательство теорем



T1:: $\text{sp}(B1, I) \Rightarrow (?) W$

$$\text{sp}(B1, I) = (n \geq 1 \wedge x=1 \wedge y=1)$$

T1:: $(n \geq 1 \wedge x=1 \wedge y=1) \Rightarrow (?) (x = y!)$ Да!

T2:: $\text{sp}([B2\text{да}; B3], W) \Rightarrow (?) W$

$$\begin{aligned} \text{sp}([B2\text{да}; B3], W) &= \\ \text{sp}(B3, x = y! \wedge y \neq n) &= // x \leftarrow x/y; y \leftarrow y - 1 \\ (x/y = (y-1)! \wedge (y-1) \neq n) & \end{aligned}$$

$(x/y = (y-1)! \wedge (y-1) \neq n) \Rightarrow (?) (x = y!)$ Да!

T3:: $\text{sp}(B2\text{нет}, W) \Rightarrow (?) R$

$$\text{sp}(B2\text{нет}, W) = (x = y! \wedge \neg y \neq n)$$

$(x = y! \wedge y = n) \Rightarrow (?) (x = n!)$ Да!

我的求解过程：

① $T1: sp(B1, I) \Rightarrow W$ (?)

$$\begin{aligned} \because sp(B1, I) &= sp(x=1; y=1, n \geq 1) \\ &\equiv (n \geq 1) \wedge (x=1) \wedge (y=1) \end{aligned}$$

$$W = x=y!$$

$$sp = (x=1) \wedge (y=1) \wedge (n \geq 1) \Rightarrow W$$

$$\therefore T1 == \text{True}$$

② $T2: sp(y \neq n; B3, W) \Rightarrow W$ (?)

$$\begin{aligned} \because sp(y \neq n; B3, W) &\equiv sp(B3, W \wedge (y \neq n)) \\ &\equiv sp(B3, x=y! \wedge y \neq n) \quad // x = \frac{x}{y} \quad y = y - 1 \\ &\equiv (\frac{x}{y} = (y-1)!) \wedge (y-1 \neq n) \\ &\equiv (x = y!) \wedge (y \neq n+1) \Rightarrow (x = y!) \equiv W \end{aligned}$$

$$\therefore T2 == \text{True}$$

③ $T3: sp(B2-, W) \Rightarrow R$ (?)

$$\begin{aligned} sp(B2-, W) &= sp(y=n, x=y!) \\ &\equiv x=n! \end{aligned}$$

$$R = x=n!$$

$$\therefore sp \Rightarrow R \text{ 成立}$$

$$\therefore T3 == \text{True}$$

综上所述，定理 $T1, T2, T3$ 均成立。

因此根据部分性正确定理可以判断，程序 P 对于初始条件 I，满足部分性正确。

这里的难题在于，循环不变式怎么找？

循环不变式的定义：

循环不变式 (Loop Invariant) 是指在循环开始前、循环每次迭代时以及循环结束后，始终保持为真的某个条件或谓词。它用来帮助证明程序的正确性，尤其是证明程序在完成时是否满足后置条件。

循环不变式的找法：

找循环不变式通常遵循以下步骤：

1. 分析程序的目标：

- 确定程序的功能和目标是什么，比如计算阶乘、排序数组等。

2. 理解程序的状态变化：

- 观察程序中的变量在每次循环迭代时如何变化。
- 需要找出一个条件，它在每次迭代后都为真，并且帮助描述程序的目标。

3. 验证初始条件和循环终止条件：

- 循环不变式应当在初始时成立，并且在每次循环执行后依然成立。
- 循环退出时，不变式应有助于证明程序满足后置条件。

4. 使用归纳法：

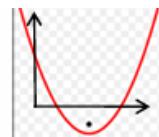
- 通过归纳法来验证：假设不变式在某次迭代中成立，证明在下一次迭代中依然成立。

求解抛物线移动之后的图像，原理是求解程序的最强后置条件。【小测原题】

Школьный пример

Задача для 7 класса:

“Если параболу $y = x^2 - 6x + 2$ сдвинуть на одну единицу вправо и на две единицы вниз, то график какой функции получится?”



Задачу можно сформулировать в наших терминах:

$$I = \{y = x^2 - 6x + 2\}$$

S1: $x := x + 1;$
S2: $y := y - 2;$

$sp(S, I) = ?$

$$sp([S_1; S_2], I) = sp([S_2], sp([S_1], I))$$

Сначала предикат “протаскиваем” через оператор S1, затем – через оператор S2

$$\begin{aligned} sp(S, I) &= sp([x:=x+1; y:=y-2], \{y = x^2 - 6x + 2\}) \equiv \\ &= sp([y:=y-2], \{sp([x:=x+1], \{y = x^2 - 6x + 2\})\}) \equiv \\ &= sp([y:=y-2], \{y = (x-1)^2 - 6(x-1) + 2\}) \equiv \\ &= \{y = x^2 - 8x + 9\} \underset{y \leftarrow y + 2}{\equiv} \\ &= \{(y+2) = x^2 - 8x + 9\} \equiv \\ &= \{y = x^2 - 8x + 7\} \end{aligned}$$

Ответ: после сдвига получим график параболы $y = x^2 - 8x + 7$

Ю.Г.Карпов

40

在移动之前满足抛物线 $I = \{y = x^2 - 6x + 2\}$ ，经过移动(程序)之后会得到怎样的图像？
这个问题等价于，求解程序的最强后置条件。

$\{I\} S \{R\}$ – утверждение：“ Если состояние программы S завершается, то после завершения состояния S удовлетворяет постусловию R ”

wp (最弱前置条件)

$wp(S, R)$ - максимально широкое предусловие, такое, что из любого состояния, удовлетворяющего $wp(S, R)$, после завершения программы S состояние S будет удовлетворять R

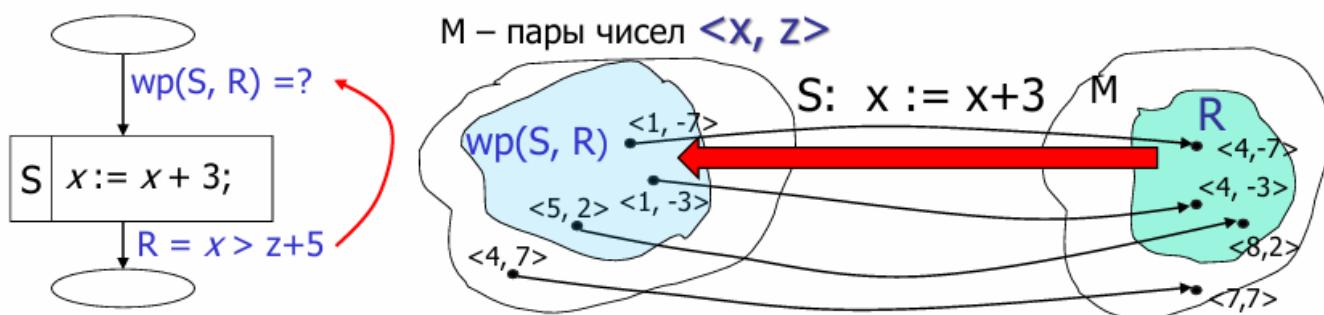
Определение. Слабейшее предусловие $wp(S, R)$ программы S задает множество ВСЕХ тех состояний S, из которых в результате выполнения S попадем в те состояния, которые удовлетворяют предикату R

wp(P, Q) 是程序 P 执行之前需要满足的最弱条件，使得程序 P 在执行后能够保证满足后置条件 Q。

简而言之：

- $wp(P, Q)$ 是所有能够确保程序 P 执行完后满足 Q 的前置条件中最弱的一个。
- "最弱" 表示：它包含所有更强的前置条件（即满足 wp 的条件也满足所有更强的条件）。

例子：



$$w(S, R) = (x + 3) > z + 5$$

定理： $\{ I \} S \{ R \}$ 成立当且仅当 $I \Rightarrow wp(S, R)$

怎样求 wp?

如何求程序的最弱前置条件 $wp(S, R)$

根据 **PPT 3: Модуль 3. Тимпоральные логики** 提供的方法，对于一个程序 SS，给定前条件 II 和后条件 RR，求解最弱前置条件 $wp(S, R)$ 可以分不同的程序结构讨论。

(1) 赋值语句

形式: $x := e$

- 赋值语句表示将表达式 e 的值赋给变量 x 。

计算规则:

- 将后置条件 R 中的 x 替换为 e , 表示执行赋值前变量 x 的值应该是 e 的计算结果。

$$wp(x := e, R) = R[x \mapsto e]$$

PPT 示例:

位置: PPT 第4页

plaintext

复制代码

```
x := x + 1;  
R: x > 10
```

计算:

- 执行后 $x = x + 1$, 后置条件 R 表示 $x > 10$ 。
- 替换 x 为 $x + 1$:

$$wp(x := x + 1, x > 10) = (x + 1 > 10)$$

- 化简:

$$wp(x := x + 1, x > 10) = x > 9$$

(2) 顺序结构

形式: $S_1; S_2$

- 顺序执行的两个子程序。

计算规则:

1. 先计算第二个子程序的最弱前置条件 $wp(S_2, R)$ 。
2. 再将其作为第一个子程序的后置条件计算 $wp(S_1, wp(S_2, R))$ 。

$$wp(S_1; S_2, R) = wp(S_1, wp(S_2, R))$$

PPT示例:

位置: PPT 第5页

plaintext

□ 复制代码

```
x := x + 1;  
y := x * 2;  
R: y > 20
```

计算:

1. 对 $y := x * 2$:
 - $wp(y := x * 2, y > 20) = x * 2 > 20$ 。
 - 化简得 $x > 10$ 。
2. 对 $x := x + 1$:
 - 后置条件为 $x > 10$ 。

- $wp(x := x + 1, x > 10) = x + 1 > 10$ 。
- 化简得 $x > 9$ 。

结果:

$$wp(S, R) = x > 9$$

(3) 条件分支

2.3. 条件分支

形式:

if C then S_1 else S_2

计算规则:

- 计算条件成立和不成立时分别执行 S_1 和 S_2 的最弱前置条件。
- 合并两种情况的结果:

$$wp(\text{if } C \text{ then } S_1 \text{ else } S_2, R) = (C \implies wp(S_1, R)) \wedge (\neg C \implies wp(S_2, R))$$

PPT示例:

位置: PPT 第6页

```
plaintext 复制代码
if x > 0 then
    x := x + 1
else
    x := x - 1
end;
R: x > 5
```

计算:

1. 分支 1:

- 条件: $x > 0$ 。

- $wp(x := x + 1, x > 5) = x + 1 > 5$ 。
- 化简: $x > 4$ 。

2. 分支 2:

- 条件: $x \leq 0$ 。
- $wp(x := x - 1, x > 5) = x - 1 > 5$ 。
- 化简: $x > 6$ 。

合并:

$$wp(S, R) = (x > 0 \implies x > 4) \wedge (\neg(x > 0) \implies x > 6)$$

化简:

$$wp(S, R) = (x > 4) \vee (x > 6)$$

(4) 循环结构

形式:

$\text{while } C \text{ do } S$

计算规则:

- 对于循环结构, 最弱前置条件的计算依赖于循环不变式 I :

- 找出循环不变式 I , 它在每次迭代前后保持为真。
- wp 是不变式在循环终止条件下的结果。

$$wp(\text{while } C \text{ do } S, R) = I \wedge \neg C$$

PPT示例:

位置: PPT 第7页

plaintext

复制代码

```
x := N;  
y := 1;  
while x > 0 do  
    y := y * x;  
    x := x - 1  
end;  
R: y = N!
```

计算:

1. 循环不变式:

- $I : y = (N - x)!$ 。
- 初始时 $x = N$, $y = 1$, 满足 $y = (N - N)! = 1$ 。
- 循环每次迭代后 y 更新为 $y * x$, x 减 1, 保持 I 。

2. 终止条件:

- $\neg C : x = 0$ 。
- 此时 $y = N!$ 。

结果:

$$wp(\text{while } C \text{ do } S, R) = y = (N - x)! \wedge x = 0$$

总结

- 赋值语句:** 用目标变量的表达式替换后置条件。
- 顺序结构:** 递归计算子程序的 $wpwpwp$ 。
- 条件分支:** 分别计算分支条件下的 $wpwpwp$ 并合并。

4. 循环结构：利用循环不变式计算终止条件下的 wwww。

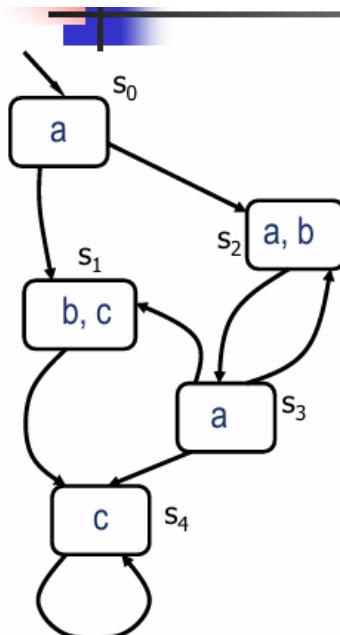
时态逻辑

- 时序逻辑是一种扩展的逻辑形式，用于描述系统的时间行为。
- 它允许表达系统状态随时间的变化，例如某个属性在未来是否成立。
- LTL (线性时序逻辑)**：表达“在未来某时刻某事件必然发生”。
- CTL (分支时序逻辑)**：表达“从当前状态出发，有一个路径满足某条件”。
- 应用场景：常用语验证交互式系统
 - 验证系统的安全性 (Safety)。
 - 验证系统的活性 (Liveness)。

Структура Крипке (Kripke 结构)

定义：Kripke 结构是描述系统状态及其状态间转移的数学模型，用于形式化验证和时序逻辑公式的语义解释。它是时序逻辑（如 LTL 和 CTL）的基础。

时序逻辑传统上是由克里普克结构进行解释的。



Структура Крипке – конечная система переходов с непомеченными переходами, с каждым состоянием которой связано некоторое множество атомарных утверждений, истинных в этом состоянии

- Формально:** $M = (S, S_0, AP, R, L)$, где:
 - S – конечное множество состояний
 - S_0 – множество начальных состояний
 - $R \subseteq S \times S$ – множество переходов;
 $(\forall s)(\exists s'): (s, s') \in R$
 - AP – множество атомарных утверждений
 - $L: S \rightarrow 2^{AP}$ – функция пометок
- Путь в M** – любая бесконечная цепочка, начинающаяся с s_0 , например: $s_0 s_1 s_2 s_3 \dots$

Kripke 结构由以下三部分组成：

- 状态集合 S ，其中初始状态集合 S_0 是 S 的子集
- 转移关系集合 R ：描述状态间可能的转移。
- 原子命题集合 AP ：系统中的原子命题集合。
- 标签函数 L ： $L(s)$ 表示在状态 s 下为真的命题。

Kripke 结构的图形表示

在 Kripke 结构中：

- 状态**：用圆圈表示（如 s_0, s_1, s_2 ）。
- 转移关系**：用有向边连接状态（如 $s_0 \rightarrow s_1$ ）。
- 标签**：标注在状态内的原子命题（如 p, q ）。

时态逻辑 (Tense Logic, TL)

Modal logic - any formal logical system in which modal operators are present.

模态逻辑是其中存在模态运算符的任何形式逻辑系统。

Примеры модальных операторов: 模态运算符

- Mp : "возможно, что p истинно" p 可能为真
- Lp : " p обязательно истинно" p 必然为真
- Fp : "когда-нибудь в будущем p будет истинным" 在未来 p 将会为真
- Gp : " p будет истинным всегда в будущем" p 在未来永远为真
- K_Ap : "агент A знает, что p истинно" 代理人知道 p 为真
- B_Ap : "агент A считает (полагает), что p истинно" 代理人相信 p 为真

时态逻辑是模态逻辑中的一种。

线性时序逻辑 (LTL)

LTL 的逻辑公式描述系统在线性时间上的行为。

常用操作符:

- **X (Next):** 下一个状态
- **F (Eventually):** 未来某个状态
- **G (Globally):** 所有未来状态
- **U (Until):** 直到

LTL 是一种形式语言。形式语言有语义和语法。

我们先说语法:

LTL公式 ϕ 是指:

- 原子语句 $p, q \dots$
- 逻辑运算符: 非 (\neg) 、与 (\wedge) 、或 (\vee) 、条件 (\rightarrow)
- 时态运算符: X, F, G, U, R 等

Грамматика: $\phi ::= p \mid \phi \vee \phi \mid \neg \phi \mid X\phi \mid \phi \mathbf{U} \phi$

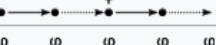
其他时态运算符:

$$\mathbf{F}p \equiv \text{true } \mathbf{U} p \quad \mathbf{G}p \equiv \neg \mathbf{F} \neg p$$

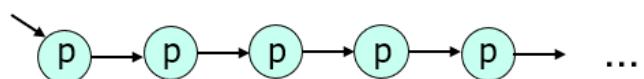
总之:

- LTL 有4个模态运算符: F, G, U, X
- F, G 通过 U 来表示:
 - $Fp \equiv \text{true } U p$
 - $Gp \equiv \neg F \neg p$

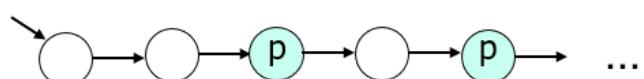
语义上，每个时态运算符 X, F, G, U, R, W, M 都有对应的含义：

字母表示	符号表示	说明	克里普克路径示意图
一元运算：			
X φ	$\bigcirc\varphi$	next (下一刻) : φ 在下一时刻为真	
F φ	$\Diamond\varphi$	Finally (最终) : φ 在以后某个时刻 (最终) 会真	
G φ	$\Box\varphi$	Globally (总是) : 从当前时刻起, φ 总是为真	
二元运算：			
$\psi \mathbf{U} \varphi$	$\psi \mathcal{U} \varphi$	Until (直到) : ψ 总是为真, 直到某一时刻 φ 为真; 该时刻可以为当前时刻或者以后某个时刻	
$\psi \mathbf{R} \varphi$	$\psi \mathcal{R} \varphi$	Release (释放) : φ 总是为真, 直到某一时刻 ψ 和 φ 同时为真; 如果 ψ 从未为真, 则 φ 必须保持永远为真	
$\psi \mathbf{W} \varphi$	$\psi \mathcal{W} \varphi$	Weak until (弱直到) : ψ 总是为真, 直到某一时刻 φ 为真; 如果 φ 从未为真, 则 ψ 必须保持永远为真	
$\psi \mathbf{M} \varphi$	$\psi \mathcal{M} \varphi$	Strong release (强释放) : φ 总是为真, 直到某一时刻 ψ 和 φ 同时为真; 该时刻可以为当前时刻或者以后某个时刻	

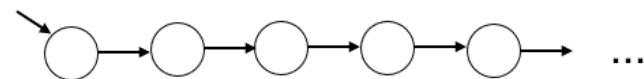
G p - всегда в будущем p



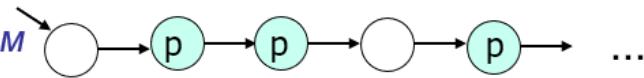
F p - хотя бы раз в будущем p



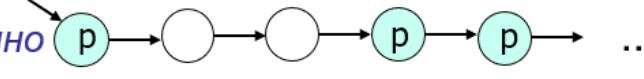
$\neg F p$ - никогда в будущем p



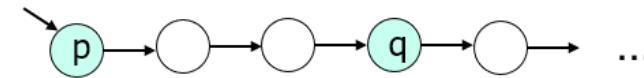
GFp - бесконечно много раз в будущем



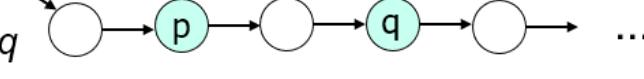
FGp - с какого-то момента постоянно



$p \Rightarrow F q$ - на p в s_0 когда-нибудь в будущем будет реакция q



$G[p \Rightarrow F q]$ - всегда на p будет реакция q



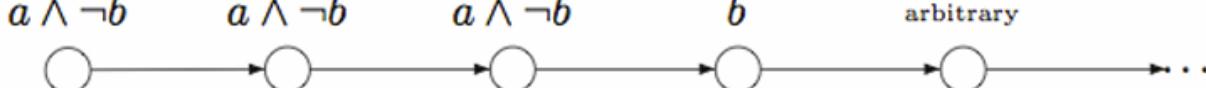
$a \wedge \neg b$
 $a \mathbf{U} b$

$a \wedge \neg b$

$a \wedge \neg b$

b

arbitrary



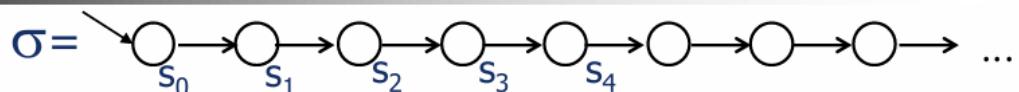
- G p - всегда в будущем p
- F p - хотя бы раз в будущем p
- $\neg F p$ - никогда в будущем p
- GFp - бесконечно много раз в будущем
- FGp - с какого-то момента постоянно
- $p \Rightarrow F q$ - на p в s_0 когда-нибудь в будущем будет реакция q
- $G[p \Rightarrow F q]$ - всегда на p будет реакция q

- $G[p \Rightarrow Fq]$ – всегда на p будет реакция q

时间运算符的解释：

- $\times p$ ：在下一个状态， p 为真。
- $F p$ ：在未来的某个状态， p 为真。
- $G p$ ：包括现在的所有未来状态， p 为真。（包含现在！！）
- $p \cup q$ ： p 一直为真，直到 q 为真。

Семантика операторов LTL. Формулы LTL интерпретируются на вычислениях



$\sigma = s_0 s_1 s_2 \dots$; $\sigma_i |= \varphi$ означает: в состоянии s_i вычисления σ истинно φ
Базовые операторы \vee, \neg, U, X

$\sigma_i = p$	<u>iff</u>	в состоянии s_i истинно атомарное утверждение p
$\sigma_i = \neg\phi$	<u>iff</u>	$\sigma_i \not\models \phi$
$\sigma_i = \phi \vee \psi$	<u>iff</u>	$\sigma_i \models \phi$ или $\sigma_i \models \psi$
$\sigma_i = X \phi$	<u>iff</u>	$\sigma_{i+1} \models \phi$
$\sigma_i = \phi U \psi$	<u>iff</u>	$(\exists j \geq i) \sigma_j \models \psi$ и $(\forall k: i \leq k < j) \sigma_k \models \phi$

Выводимые операторы Fp, Gp

$\sigma_i \models G \phi$	<u>iff</u>	$(\forall j \geq i) \sigma_j \models \phi$
$\sigma_i \models F \phi$	<u>iff</u>	$(\exists j \geq i) \sigma_j \models \phi$

Истиинность темпоральной формулы на ВЫЧИСЛЕНИИ определяется относительно начального состояния вычисления, т.е.

Ф выполняется на вычислении σ iff $\sigma_0 \models \Phi$

Ю.Г.Карпов

41

解释：对于一条路 σ , 定义 σ 满足公式 p 为 $\sigma \models p$ 。归纳如下：

1. $\sigma_i \models p$: 在状态 s_i 的时候语句 p 为真
2. $\sigma_i \models \neg p$: 在状态 s_i 的时候语句 p 为假
3. $\sigma_i \models p \vee q$: 在状态 s_i 的时候, $p \vee q$ 为真
4. $\sigma_i \models X p$: 在 s_i 的下一个状态 (s_{i+1}) p 为真
5. $\sigma_i \models p U q$: 在 s_i 之前, p 一直为真, 从 s_i 开始(包括 s_i) q 一直为真
6. $\sigma_i \models G p$: 从 s_i 开始(包括 s_i) p 一直为真
7. $\sigma_i \models F p$: 从 s_i 开始(包括 s_i) 至少存在一个 p 为真的状态

LTL 通过 X 进行定义：

Определение операторов LTL через neXtTime

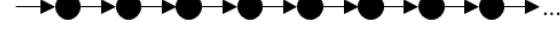
- $pUq \equiv q \vee p \wedge Xq \vee p \wedge Xp \wedge XXq \vee \dots$



- $Fq \equiv q \vee Xq \vee XXq \vee \dots$



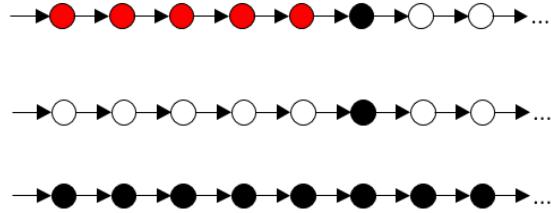
- $Gq \equiv q \wedge Xq \wedge XXq \wedge \dots$



LTL的递归定义:

Рекурсивное определение операторов LTL

- $pUq \equiv q \vee p \wedge X(pUq)$
- $Fq \equiv q \vee XFq$
- $Gq \equiv q \wedge XGq$



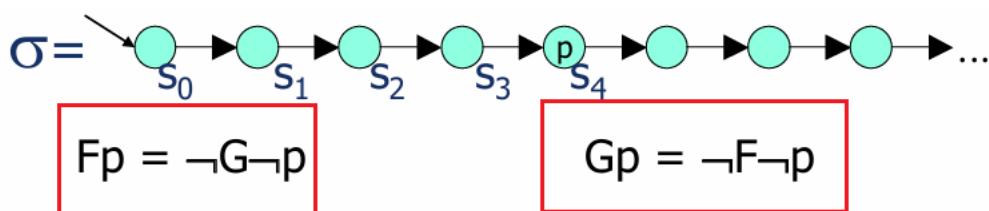
Ю.Г.Карпов

34

F 和 G 之间的关系:

$$Fp = \neg G \neg p$$

$$Gp = \neg F \neg p$$



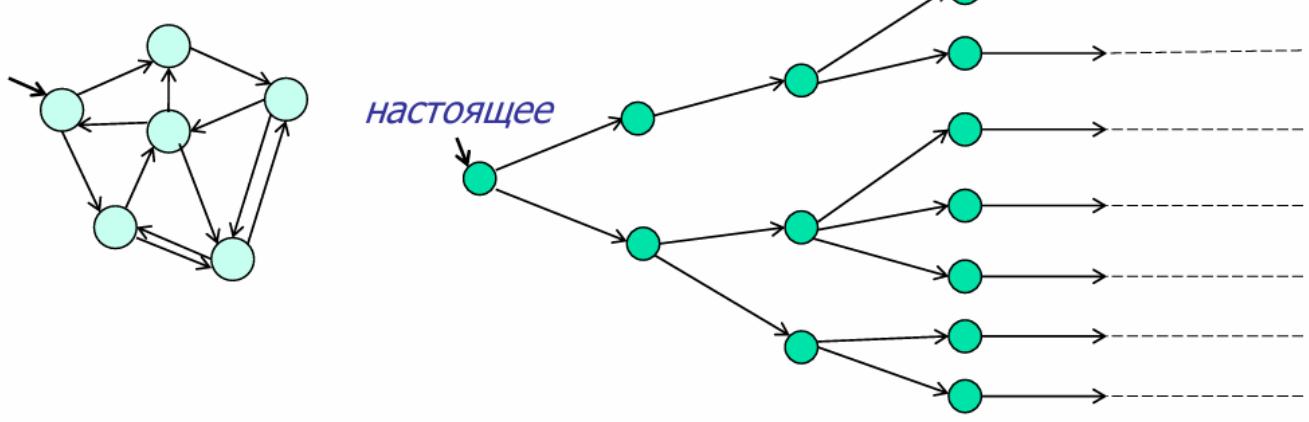
- Отношение между F и G следует из закона ДеМоргана:

$$\begin{aligned} Fp &\equiv p \vee Xp \vee XXp \vee XXXp \vee \dots \equiv \\ &\equiv \neg(\neg p \wedge \neg Xp \wedge \neg XXp \wedge \neg XXXp \wedge \dots) \equiv \\ &\equiv \neg(\neg p \wedge X \neg p \wedge XX \neg p \wedge XXX \neg p \wedge \dots) \equiv \\ &\equiv \neg G \neg p \end{aligned}$$

$$A \vee B \equiv \neg(\neg A \wedge \neg B)$$

Kripke 结构的运算

Kripke 结构是无限循环的。要怎么用有限的方式计算?



Развертка структуры Кripке может быть использована как модель ветвящегося времени
Kripke 结构可以展开成分支时间模型。

分支时间模型 就是将 Kripke 结构的状态和转移关系展开，形成树状结构，称为 **状态空间展开** (State Space Unfolding) 。

Развертка структуры Кripке Kripke 结构的展开

分支时间模型对应的是 **CTL (分支时序逻辑)**，它的核心特征是：

- 可以在多条路径上检查命题是否成立。
- CTL 公式使用 **路径量词** 和 **时序操作符** 来描述分支时间的性质：

◦ **路径量词**:

- A (For All) : 所有路径上。
- E (Exists) : 存在至少一条路径。

◦ **时序操作符**:

- X: 下一个状态。
- F: 未来某个时刻。
- G: 所有时刻。
- U: 直到某个条件满足。

例如：

- AG(p): 在所有路径上，所有未来状态都满足 P。
- EF(q): 存在至少一条路径，在未来某个时刻满足 q。

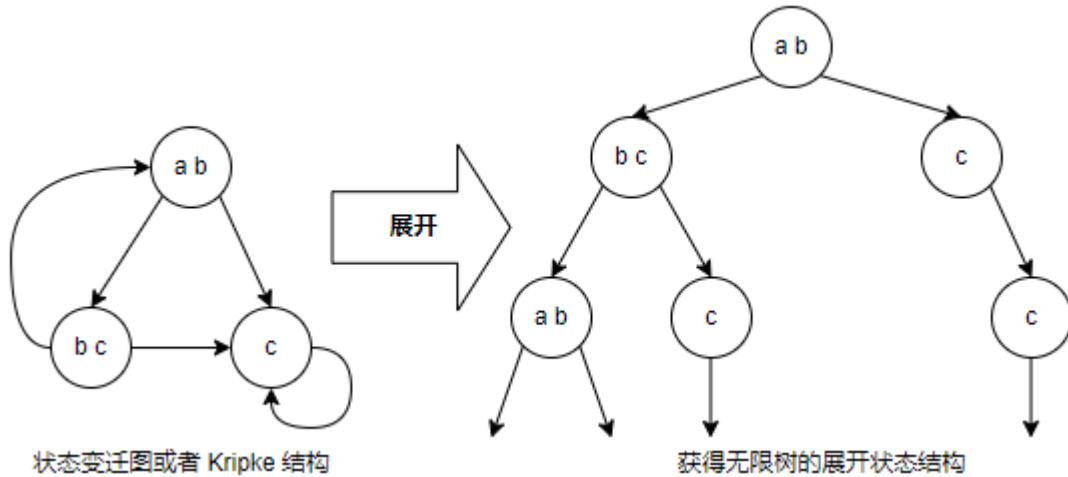
CTL

- **定义**: CTL 是一种分支时序逻辑，用于描述树状时间模型中的状态和路径的属性。
- **特点**:
 - 每个时序操作符必须带有**路径量词** (A 表示所有路径, E 表示存在路径) 。
 - 路径量词和时序操作符不能分离，必须成对出现。
- **公式结构**:
 - CTL 公式的基本形式：**路径量词 + 时序操作符**。
 - 示例： `AXp, EFq, AGp, E[pUq]`

CTL*

CTL* // Тимпоральные логики ветвящегося времени // Computational Tree Logic

CTL* 公式描述计算树 (Computation Tree) 的性质，计算树由 Kripke 结构生成。首先在 Kripke 结构中指定一个状态为初始状态，接着将这个结构展开成以此状态作为根的无限树。计算树展示了从初始状态开始的所有可能的执行路径。

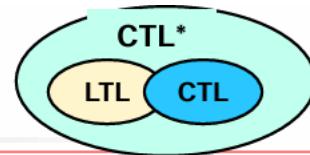


CTL 和 CTL* 的区别

- **定义:** CTL* 是 CTL 和 LTL (线性时序逻辑) 的统一扩展，允许更复杂的组合表达。
- **特点:**
 - CTL* 的路径量词 (AAA、EEE) 和时序操作符 (X,F,G,UX, F, G, UX,F,G,U) 可以分离，可以灵活地组合。
 - CTL* 允许在路径上编写任意的 LTL 公式，同时也支持分支逻辑的表达能力。
- **公式结构:**
 - CTL* 公式的基本形式：路径量词 + LTL 公式。
 - 示例：A(GFp), E(G(p ∨ Fq)), G(Fp)

特性	CTL	CTL*
路径量词与操作符	必须成对出现 (例如: AX, EF, AGAX, EF, AGAX, EF, AG)。	可以分离，路径量词与 LTL 公式自由组合。
公式的表达能力	更受限制，公式较为简单。	更强大，支持嵌套和任意组合的公式。
示例	AG(p → AFq)	G(Fp ∨ A(Gq))

LTL и CTL – подклассы CTL*



Анализ выполнения формул CTL* сложен \Rightarrow нужны подклассы

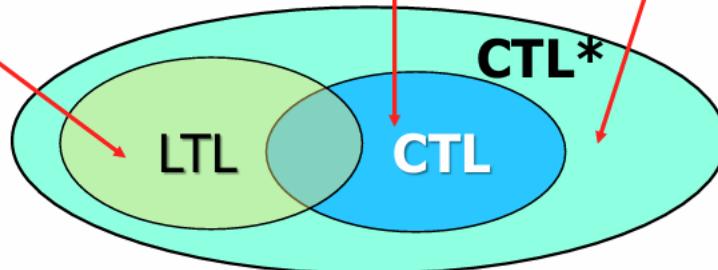
В LTL – все формулы пути должны выполняться для всех вычислений, поэтому, фактически, они **НЕЯВНО** предваряются квантором пути A

В CTL **каждый** темпоральный оператор предваряется квантором пути

Формулы LTL:
 $\text{AG}(p \Rightarrow F q)$
 $\text{A}(\neg a \vee G b \wedge (a U \neg c))$
 $\text{A}(a U \neg b)$

Формулы CTL:
 $\text{AG}(p \& \neg EF(q \Rightarrow r))$
 $EF(a \& E(a U \neg c))$
 $A(a U \neg b)$

Формулы CTL*:
 $E(\neg p \& X A F q)$
 $EX(a \& AX(b U c))$
 $A(a U \neg (F b))$



Ю.Г.Карпов

70

可以认为 CTL 是 CTL* 的子集

CTL* 允许的运算符

Операции логики
высказываний

1. \neg – Отрицание
 2. \vee – Дизъюнкция
 3. \wedge – Конъюнкция
 4. \Rightarrow – Импликация
- ...

Четыре темпоральных
оператора

1. X – “neXt time”
2. U – “Until”
3. F – “in the Future”
4. G – “Globally”

Два квантора
пути

1. E – “Exists”
2. A – “Always”

Базис CTL* = { \neg, \vee, U, X, E }

E и **A** определяют путь (кванторы пути)

X,U,F,G характеризуют темпоральные свойства вычислений

Семантика CTL* на структурах Кripке // CTL* 的语法

Семантика CTL* задается на структуре Кripке M . В состояниях выполняются формулы состояний, на путях выполняются формулы пути.

- s – состояние, σ – вычисление (путь)
- Семантика формул состояний φ
 - $M, s \models p$ iff $p \in L(s)$ (p принадлежит множеству пометок состояния s)
 - $M, s \models \neg\varphi$ iff $\neg(M, s \models \varphi)$
 - $M, s \models \varphi_1 \wedge \varphi_2$ iff $M, s \models \varphi_1 \wedge M, s \models \varphi_2$
 - $M, s \models \varphi_1 \vee \varphi_2$ iff $M, s \models \varphi_1 \vee M, s \models \varphi_2$
 - $M, s \models E\psi$ iff $(\exists \sigma : \sigma_0 = s). M, \sigma \models \psi$
 - $M, s \models A\psi$ iff $(\forall \sigma : \sigma_0 = s). M, \sigma \models \psi$

- $M, s \models p$: p 在 s 状态下为真
- $M, s \models \neg p$: p 在 s 状态下为假
- $M, s \models p \vee q$: 在 s 状态下, q 或 p 为真
- $M, s \models Ep$: 从状态 s 开始(包含 s), 存在一条路径(时刻)使得 p 满足
- $M, s \models Ap$: 从状态 s 开始(包含 s), 所有路径(时刻)都满足 p

Семантика формул пути ψ

- $M, \sigma \models \varphi$ iff $M, \sigma_0 \models \varphi$
- $M, \sigma \models \neg\psi$ iff $\neg(M, \sigma \models \psi)$
- $M, \sigma \models \psi_1 \wedge \psi_2$ iff $M, \sigma \models \psi_1 \wedge M, \sigma \models \psi_2$
- $M, \sigma \models \psi_1 \vee \psi_2$ iff $M, \sigma \models \psi_1 \vee M, \sigma \models \psi_2$
- $M, \sigma \models X\psi$ iff $M, \sigma_1 \models \psi$
- $M, \sigma \models F\psi$ iff $(\exists i : i \geq 0). M, \sigma_i \models \psi$
- $M, \sigma \models G\psi$ iff $(\forall i : i \geq 0). M, \sigma_i \models \psi$
- $M, \sigma \models \psi_1 U \psi_2$ iff $(\exists k : k \geq 0). [M, \sigma_k \models \psi_2 \wedge (\forall j : 0 \leq j < k). M, \sigma_j \models \psi_1]$

- $M, \sigma \models \varphi$: 路径的初始状态 σ_0 满足状态公式 φ 。
 - $M, \sigma \models \neg\psi$: 路径上公式 ψ 不满足。
 - $M, \sigma \models \psi_1 \wedge \psi_2$: 路径上同时满足公式 ψ_1 和 ψ_2 。
 - $M, \sigma \models \psi_1 \vee \psi_2$: 路径上满足 ψ_1 或 ψ_2 。
 - $M, \sigma \models X\psi$: 路径的下一个状态满足公式 ψ 。
 - $M, \sigma \models F\psi$: 路径上存在某个时刻满足公式 ψ 。
 - $M, \sigma \models G\psi$: 路径上所有时刻都满足公式 ψ 。
 - $M, \sigma \models \psi_1 U \psi_2$: 存在某个时刻 k , 路径上满足 ψ_2 , 并且在 k 之前的所有时刻都满足 ψ_1 。
-

CTL

В CTL каждый темпоральный оператор предварен квантором пути. ВСЕ ФОРМУЛЫ CTL –формулы СОСТОЯНИЙ.

在 CTL 中, 每个时间运算符前面都有一个路径量词。

所有 CTL 公式都是 状态 公式。

CTL 语法

共有4个运算符 **X, F, G, U**, 2个路径量词 **E, A**。组合使用, 所以一共有 8 个 CTL 操作符:

AX, EX,

AF, EF,

AG, EG,

AU, EU

合法公式: `E[cub]`, `A[pU(rEFq)]`, `ExpEXq`, `EGAFp`, ...

CTL 不合法的公式有: `EGFp`, `A[xp ∨ xxr]`, ...

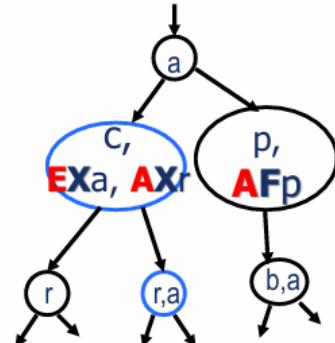
Синтаксис (грамматика):
 $\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \text{AX}\varphi \mid \text{EX}\varphi \mid \text{AF}\varphi \mid \text{EF}\varphi \mid \text{AG}\varphi \mid \text{EG}\varphi \mid \text{A}[\varphi_1 \mathbf{U} \varphi_2] \mid \text{E}[\varphi_1 \mathbf{U} \varphi_2]$

AX φ – формула φ выполняется во всех *следующих* состояниях (непосредственных потомках текущего состояния)

EX φ - формула φ выполняется хотя бы в одном *следующем* состоянии

AF φ (*неизбежно* φ) - на всех путях из текущего состояния формула φ *когда-нибудь* выполнится

EF φ (*возможно* φ) - из текущего состояния существует путь, на котором формула φ *когда-нибудь* выполнится

**Синтаксис** (грамматика):
 $\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \text{AX}\varphi \mid \text{EX}\varphi \mid \text{AF}\varphi \mid \text{EF}\varphi \mid \text{AG}\varphi \mid \text{EG}\varphi \mid \text{A}[\varphi_1 \mathbf{U} \varphi_2] \mid \text{E}[\varphi_1 \mathbf{U} \varphi_2]$

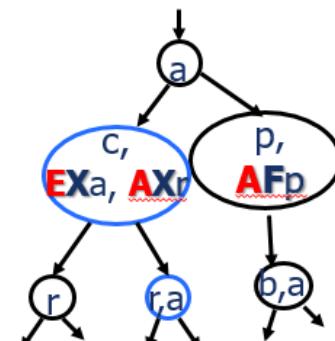
E $[\varphi_1 \mathbf{U} \varphi_2]$ **Все формулы CTL – это формулы состояний!!**

AG φ - на всех путях из текущего состояния во всех состояниях этих путей формула φ выполняется

EG φ - существует путь из текущего состояния, во всех состояниях которого формула φ выполняется

A($\varphi_1 \mathbf{U} \varphi_2$) - на всех путях из текущего состояния когда-нибудь выполнится формула φ_2 , а до этого во всех состояниях выполняется формула φ_1

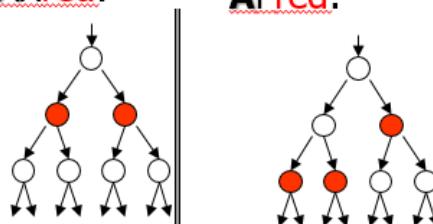
E($\varphi_1 \mathbf{U} \varphi_2$) – из текущего состояния существует путь, на котором когда-нибудь выполнится φ_2 , а до этого во всех состояниях этого пути выполняется φ_1



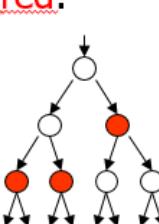
Синтаксис (грамматика):

$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid AX_\varphi \mid EX_\varphi \mid AF_\varphi \mid EF_\varphi \mid AG_\varphi \mid EG_\varphi \mid A[\varphi_1 \cup \varphi_2] \mid E[\varphi_1 \cup \varphi_2]$

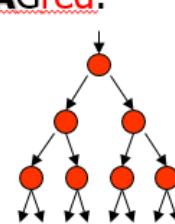
AXred:



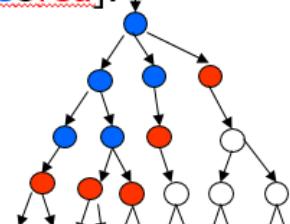
AFred:



AGred:



A [blueUred]:

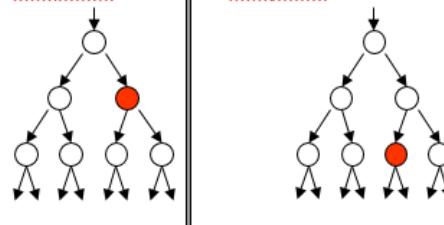


$AF_\varphi = \varphi \vee AX\varphi$

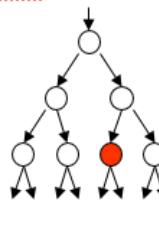
$AG_\varphi = \varphi \wedge AX\varphi$

$A[\varphi \cup \psi] = \varphi \vee \psi \wedge A[\varphi \cup \psi]$

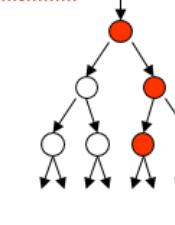
EXred:



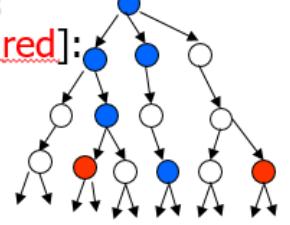
EFred:



EGred:



E [blueUred]:



$EF_\varphi = \varphi \vee EX\varphi$

$EG_\varphi = \varphi \wedge EX\varphi$

$E[\varphi \cup \psi] = \varphi \vee \psi \wedge E[\varphi \cup \psi]$

Ю.Г.Карпов

16

形式化定义CTL:

Формальная семантика CTL

$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid AX_\varphi \mid EX_\varphi \mid AF_\varphi \mid EF_\varphi \mid AG_\varphi \mid EG_\varphi \mid A[\varphi_1 \cup \varphi_2] \mid E[\varphi_1 \cup \varphi_2]$

$s \models p \equiv p \in L(s);$

$s \models \neg\varphi \equiv s \not\models \varphi;$

$s \models \varphi_1 \vee \varphi_2 \equiv s \models \varphi_1 \vee s \models \varphi_2;$

$s \models AX\varphi \equiv (\forall s_1: s \rightarrow s_1) s_1 \models \varphi;$

$s \models EX\varphi \equiv (\exists s_1: s \rightarrow s_1) s_1 \models \varphi;$

$s \models AG\varphi \equiv \forall (s_0 \rightarrow s_1 \rightarrow \dots) (s=s_0) (\forall i : 0 \leq i) s_i \models \varphi;$

$s \models EG\varphi \equiv \exists (s_0 \rightarrow s_1 \rightarrow \dots) (s=s_0) (\forall i : 0 \leq i) s_i \models \varphi;$

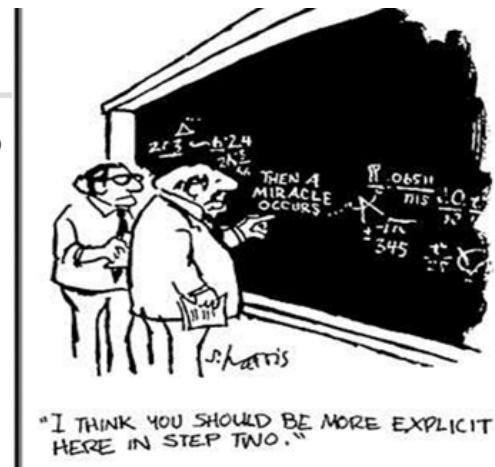
$s \models AF\varphi \equiv \forall (s_0 \rightarrow s_1 \rightarrow \dots) (s=s_0) (\exists i : 0 \leq i) s_i \models \varphi;$

$s \models EF\varphi \equiv \exists (s_0 \rightarrow s_1 \rightarrow \dots) (s=s_0) (\exists i : 0 \leq i) s_i \models \varphi;$

$s \models A(\varphi_1 \cup \varphi_2) \equiv \forall (s_0 \rightarrow s_1 \rightarrow \dots) (s=s_0) (\exists j : 0 \leq j) (s_j \models \varphi_2 \wedge (\forall k : 0 \leq k < j) s_k \models \varphi_1);$

$s \models E(\varphi_1 \cup \varphi_2) \equiv \exists (s_0 \rightarrow s_1 \rightarrow \dots) (s=s_0) (\exists j : 0 \leq j) (s_j \models \varphi_2 \wedge (\forall k : 0 \leq k < j) s_k \models \varphi_1).$

Ю.Г.Карпов



23

其他表示方法:

在模型验证中，表达式 $M, s \models A\psi$ 表示在模型 M 的状态 s 下，“**总是**”满足公式 ψ ，也就是说，沿着从 s 出发的 **所有路径 (All)**，公式 ψ 都成立。

具体解释如下：

- **M**: 这是一个 Kripke 结构（或状态转换系统），用来表示模型的所有状态及其状态间的转移关系
- **s**: 模型 M 中的一个特定状态
- **A**: 这是 CTL（计算树逻辑）中的路径量词“所有（All）”。它表示从当前状态出发的所有可能路径。
- **ψ** : 这是一个逻辑公式，可以描述状态的某种性质或一系列性质

因此，整个公式 $M, s \models A\psi$ 表示：

在模型 M 的状态 s 中，沿着所有可能的路径，公式 ψ 都成立。换句话说，从状态 s 出发，不管未来的路径怎么发展，公式 ψ 在每条路径上都是成立的

CTL 公式例子

- **AG p**: 在所有路径上的所有状态中， p 都为真 (p 在所有未来都成立)。
- **EF p**: 存在一条路径，在该路径的某个未来状态， p 为真 (p 在某个未来状态成立)。

CTL 的一个关键特性是它的表达力不仅考虑了单一未来的线性时间，还能够表达不同路径的分支未来。例如，“所有可能的未来中某个状态必须满足条件”是 CTL 能表达的一个经典性质。

MC for CTL

Алгоритм Model checking для проверки выполнимости формул CTL

Наша задача – рассмотреть алгоритм Model checking для логики CTL

目标：验证 CTL（分支时序逻辑）公式在 Kripke 结构 M 上的可满足性。

任务：确定给定的 Kripke 结构 $M=(S,R,L)$ 和状态 s_0 是否满足公式 φ ，即：

$$M, s_0 \models \varphi$$

CTL 公式的特点：

- 路径量词：A, E
- 时序操作词：X, F, G, U
- CTL 公式必须满足路径两次与时序操作符配对的规则，比如
 - $AXp, EFq, AGp, E[pUq]$

方法：基于 Kripke 结构的验证方法 【必考！Exam 第一题！】

形式化表达问题：

- 输入:
 - Kripke 结构 $M = (S, R, L)$:
 - S : 状态集合。
 - $R \subseteq S \times S$: 状态之间的转移关系。
 - L : 标签函数, 将原子命题映射到状态。
 - CTL 公式 φ 。
- 输出:
 - 判断 $M, s \models \varphi$: 某状态 s 是否满足公式 φ 。

"Необходимо разработать алгоритм проверки выполнимости произвольной темпоральной формулы на любой структуре Крипке."

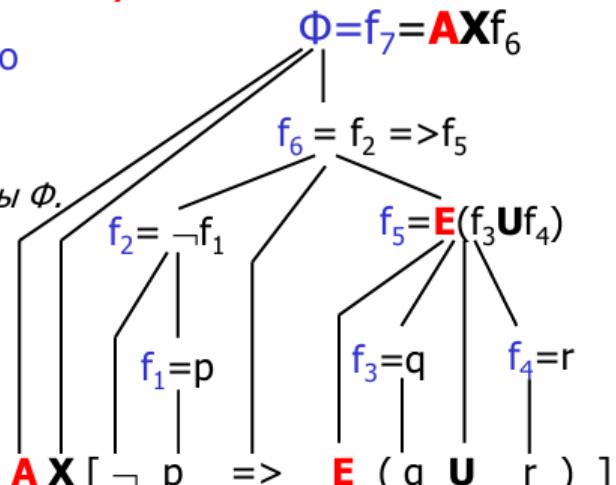
Алгоритм *маркировки* для CTL формул

Формул CTL – бесконечное число !! Как проверить для данной структуры Крипке выполнимость произвольной формулы, например
 $\Phi = \text{AX} [\neg p \Rightarrow \text{E} (q \text{U} r)]$

Общая идея – структурная индукция:

Формул – бесконечное число, но число типов подформул конечно! Они перечислены в ГРАММАТИКЕ языка

- Строим синтаксическое дерево формулы Φ ,
- Последовательно помечаем (маркируем) все состояния структуры Крипке теми подформулами формулы Φ , которые истинны в этих состояниях.
- По окончании алгоритма, если начальное состояние структуры Крипке M помечено Φ , то Φ выполняется на M .



Необходимо разработать алгоритмы маркировки состояний структуры Крипке для каждой возможной подформулы формулы Φ

Ю.Г.Карпов

22

Общая идея – структурная индукции

总体思想是结构归纳法

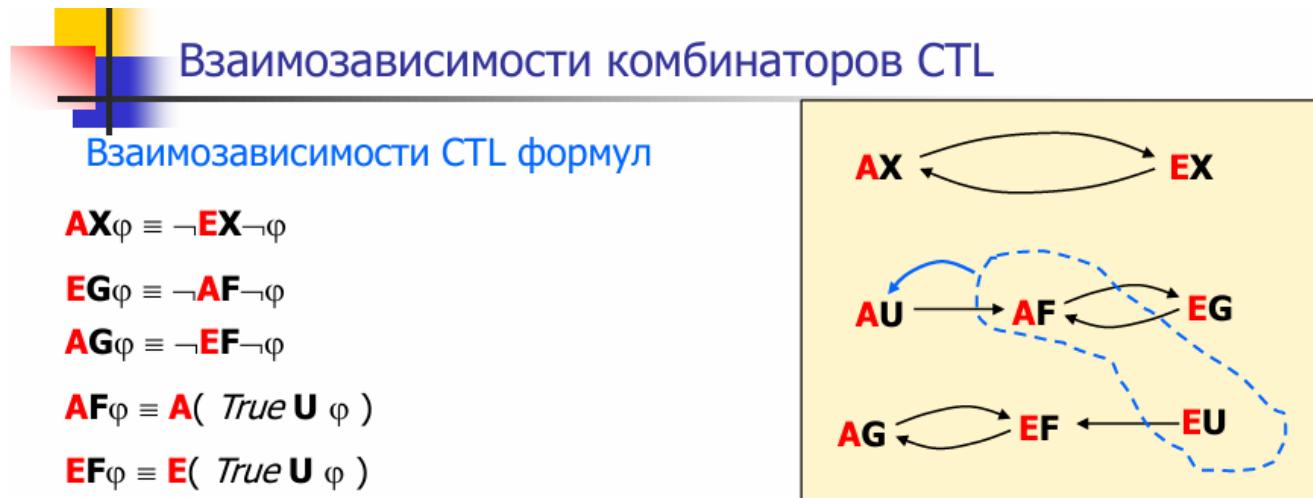
Формул – бесконечное число, но число типов подформул конечно! Они перечислены в ГРАММАТИКЕ языка
 公式有无限多个, 但子公式的类型数量是有限的! 它们列在该语言的语法中

步骤:

1. 为公式F构建语法树
2. 我们依次用在这些状态下为真的公式 Φ 的子公式来标记 (标记) 克里普克结构的所有状态。

3. 在算法结束时，如果Kripke结构M的初始状态标记为 Φ ，则在M上能满足公式 Φ 。

CTL 之间的相互关系：



可能的 CTL 基础：

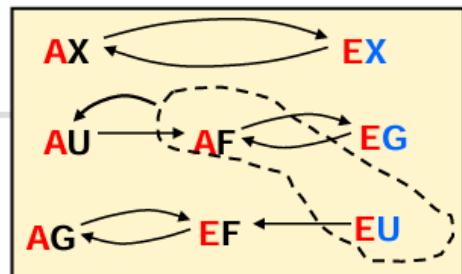
总共存在 6 个 CTL 基础：

{ EX, AF, EU },
{ AX, AU, EU },
{ EX, EG, EU } 和等等。

对于 Model checking 算法，只需要构建一个基础的标记化算法，
只针对 CTL-组合子中的任何一个基础。

这样的算法对于 EX, AX, EF, AF, EU, AU 很简单。

对于 EG 和 AG，算法稍微复杂一些，但更有效率。

**Синтаксис** (грамматика):
 $\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi_1 U \varphi_2] \mid E[\varphi_1 U \varphi_2]$


■ Базисы CTL:

■ EX, EG, EU

- $AX\varphi \equiv \neg EX\neg\varphi,$
- $EF\varphi = E[TU\varphi],$
- $AG\varphi \equiv \neg EF\neg\varphi,$
- $AF\varphi \equiv \neg EG\neg\varphi,$
- $A[\varphi U \psi] \equiv \neg E [\psi U (\neg \varphi \wedge \neg \psi)] \wedge AF\psi$

Выражение других пар через базис

■ EX, AU, EU

- $AF\varphi \equiv A[TU\varphi],$
- $EF\varphi \equiv E[TU\varphi],$
- ...

Удобно рассматривать как базис только из кванторов существования

Синтаксис

(грамматика):

 $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid EG\varphi \mid E[\varphi U \varphi]$

Этот базис требует только простых алгоритмов

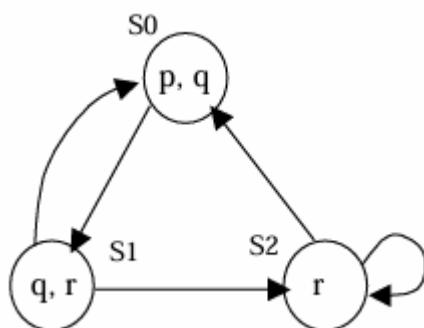
Ю.Г.Карпов

27

tbd_verification: Модуль 4. MC для LTL_1, p34页 中有一个例子:

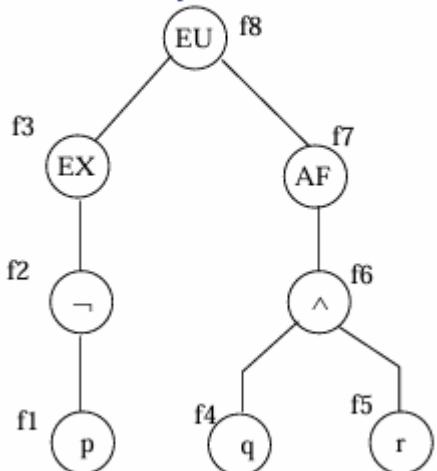
现有一个CTL公式: $\Phi = E [EX \text{ not } p \cup AF (q \wedge r)]$

和一个模型 (自动机) :

检查在模型的哪些状态下满足公式 Φ ?先说结论, 再说实操。结论是递归分解成最小的CTL公式, 然后自底向上检查每个状态满足哪些CTL公式。最后的关键是看公式 Φ 是否落在 s_0 上。如果落在了 s_0 则说明模型能完成 Φ 公式。

以这个例子探讨一下这类题型的做法: 实操三步走。

Синтаксическое дерево:



1) 公式转化成语法树:

Ю.Г.Карпов

2) 列出语法树中所有出现的公式:

$$f_1 = p$$

$$f_2 = \neg f_1$$

$$f_3 = \text{EX } f_2$$

$$f_4 = q$$

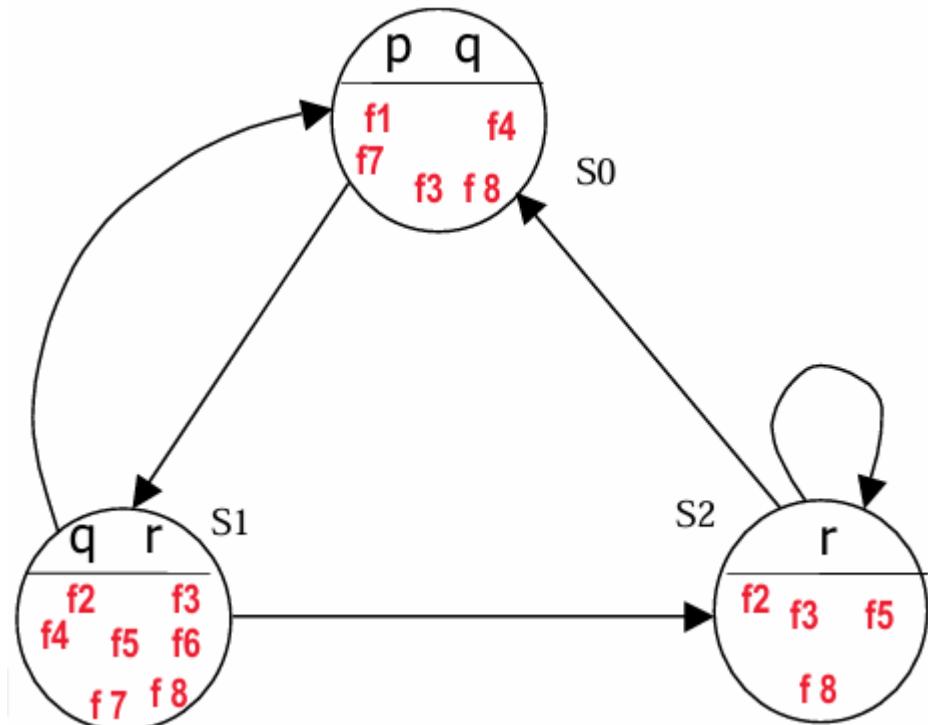
$$f_5 = r$$

$$f_6 = f_4 \wedge f_5$$

$$f_7 = \text{AF } f_6$$

$$f_8 = \text{E} [f_3 \cup f_7]$$

3) 从 f_1 开始到 $f_n(\phi)$, 检查自动机的每个状态可以满足哪些公式:



4) 最后可以得出结论, 在模型的 s_0, s_1, s_2 状态下都满足公式 $f_8 (\phi)$

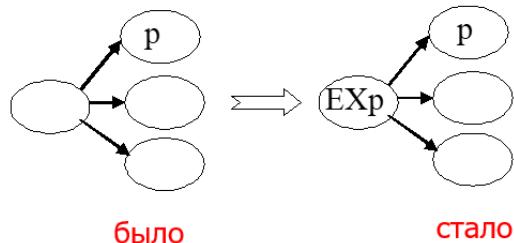
Для заданных структуре Крипке и формулы Φ логики CTL нужно уметь строить множества состояний, в которых выполняется формула Φ (и все ее подформулы).

Это является стандартным умением, которое будет проверяться у каждого студента.

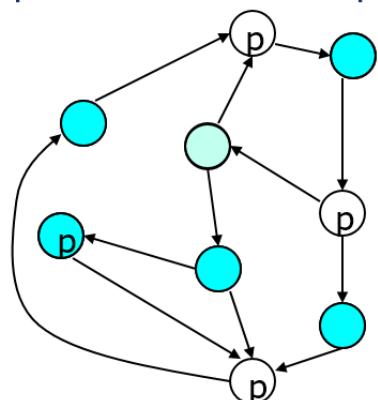
补充: 怎样检查各个公式: EX、AF、EU -----> 【重要!】

Алгоритм Model Checking для CTL (EX)

EX_p: помечаем s меткой r_{Exp} если хотя бы один преемник s помечен r



```
function SAT_EX(p) /*дает все s,
    в которых истинна EXp */
local var Y;
begin
    Y := { s | (exists s1 in SAT(p)) s → s1 };
    return Y
end
```



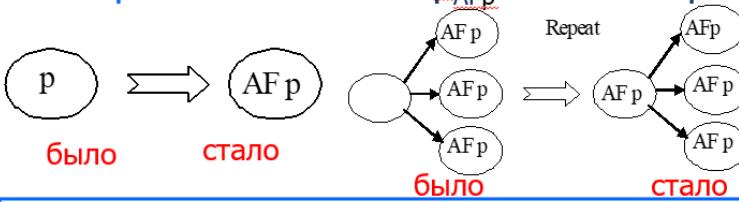
Все закрашенные
состояния попадают в
множество SAT_EX(p)

Перебираем все состояния структуры Крипке и смотрим, есть ли из текущего состояния ребро в состояние, помеченное p

Алгоритм Model Checking для CTL (AF)

$\text{AF}\varphi$: каждое состояние, если оно помечено p помечаем $\text{AF}p$;

повторяем: s помечаем $\text{AF}p$, если все преемники s помечены $\text{AF}p$



function SAT_AF(p)

/*дает все s , в которых истинна $\text{AF}p$ */

local var X, Y

begin

$X := S$;

$Y := \text{SAT}(p)$;

repeat until $X = Y$

begin

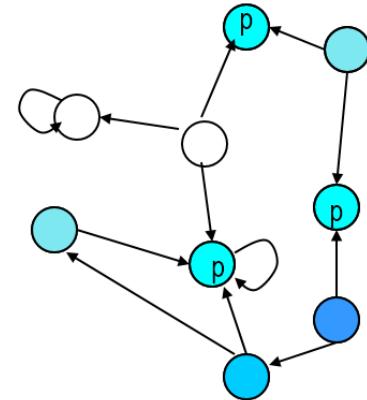
$X := Y$;

$Y := Y \cup \{ s \mid (\forall s_1: s \rightarrow s_1) s_1 \in Y \}$

end

return Y

end



Все закрашенные состояния

попадают в множество

$\text{SAT}_{\text{AF}p}$

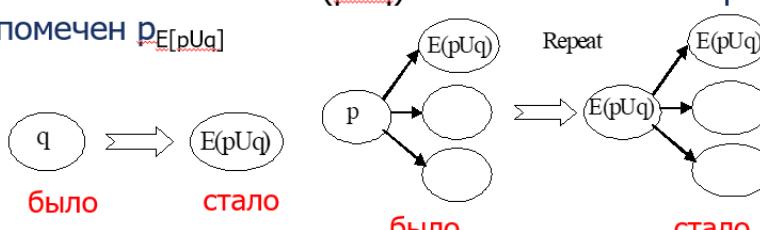
30

Алгоритм Model Checking для CTL (EU)

$E(p \cup q)$: помечаем состояние меткой $p_{E(p \cup q)}$, если оно уже помечено q ; повторяем:

помечаем s меткой $E(p \cup q)$ если оно помечено p и хотя бы один преемник s

помечен $p_{E(p \cup q)}$



function SAT_EU (p, q) /*дает все s , удовл $E(p \cup q)$ */

local var P, X, Y

begin

$P := \text{SAT}(p); X := S; Y := \text{SAT}(q);$

repeat until $X = Y$

begin

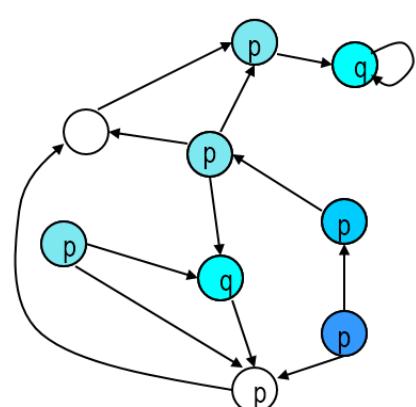
$X := Y$;

$Y := Y \cup (P \cap \{ s \mid (\exists s_1 \in Y) s \rightarrow s_1 \})$

end

return Y

end



Все закрашенные состояния попадают в
множество $\text{SAT}_{E(p \cup q)}$

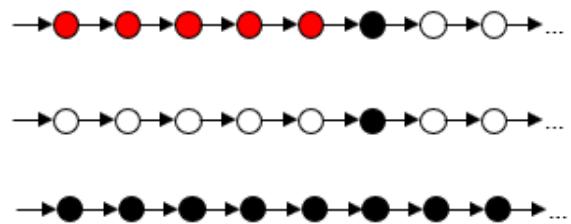
31

△ MC for LTL

Model checking для формул LTL

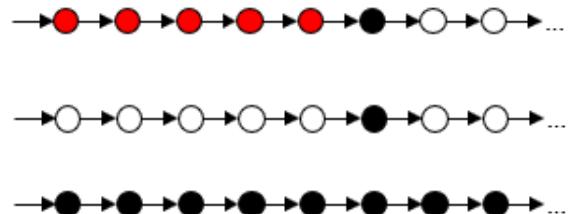
可以通过X来定义LTL的时间操作符 (U, F, G) :

- $pUq \equiv q \vee p \wedge Xq \vee p \wedge Xp \wedge XXq \vee \dots$
- $Fq \equiv q \vee Xq \vee XXq \vee \dots$
- $Gq \equiv q \wedge Xq \wedge XXq \wedge \dots$



Рекурсивное определение операторов LTL

- $pUq \equiv q \vee p \wedge X(pUq)$
- $Fq \equiv q \vee XFq$
- $Gq \equiv q \wedge XGq$



Ю.Г.Карпов

34

F与G之间的关系:

$$Fp = \neg G \neg p$$

$$Gp = \neg F \neg p$$

遵循“德摩根定律”

LTL 运算符的语义

假设存在一条路径 $\sigma=s_0 s_1 s_2 \dots$

那么 $\sigma_i | = \phi$ 表示: 在状态 s_i 中命题 ϕ 为真

$\sigma = s_0 s_1 s_2 \dots$; $\sigma_i \models \phi$ означает: в состоянии s_i вычисления σ истинно ϕ

Базовые операторы \vee, \neg, U, X

$\sigma_i \models p \quad \text{iff} \quad$ в состоянии s_i истинно атомарное утверждение p

$\sigma_i \models \neg \phi \quad \text{iff} \quad \sigma_i \not\models \phi$

$\sigma_i \models \phi \vee \psi \quad \text{iff} \quad \sigma_i \models \phi \text{ или } \sigma_i \models \psi$

$\sigma_i \models X \phi \quad \text{iff} \quad \sigma_{i+1} \models \phi$

$\sigma_i \models \phi U \psi \quad \text{iff} \quad (\exists j \geq i) \sigma_j \models \psi \text{ и } (\forall k: i \leq k < j) \sigma_k \models \phi$

Выводимые операторы Fp, Gp

$\sigma_i \models G \phi \quad \text{iff} \quad (\forall j \geq i) \sigma_j \models \phi$

$\sigma_i \models F \phi \quad \text{iff} \quad (\exists j \geq i) \sigma_j \models \phi$

定理: Формула LTL выполняется на структуре Kripke M , если она выполняется на любом пути (вычислении), начинающемся в начальном состоянии M

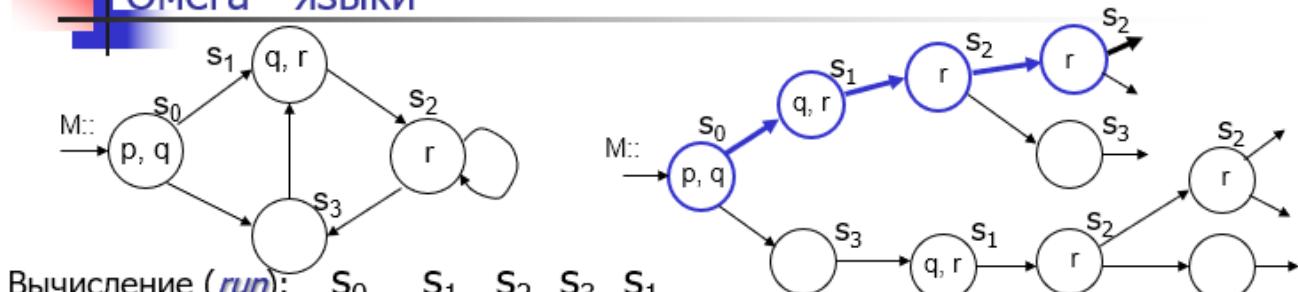
LTL 公式在 Kripke 结构 M 上成立，当且仅当它在从 M 的 **初始状态** 开始的所有路径（计算）上都成立。

- 从 Kripke 结构 M 的初始状态 s_0 出发，可能存在多条路径（状态序列），每条路径代表系统的一种可能的执行或行为。
- 公式在 **结构 M** 上成立，意味着公式必须在 **所有从初始状态出发的路径** 上都为真。
换句话说，不能有一条路径使公式不成立。

与其验证所有路径都符合LTL公式，不如找到一条路径不符合LTL公式。如果找到了则LTL不在模型上成立，如果找不到则LTL在模型上成立。

Важны НЕ вычисления, а траектории.

Омега - языки



Формальный язык (изучали раньше) – множество цепочек над конечным словарем. Число цепочек **БЕСКОНЕЧНО**. Все цепочки **КОНЕЧНЫ**.

Например: Язык Паскаль – конечные цепочки над конечным словарем (лексем). Каждая программа на Паскале – цепочка (лексем).

Каждая цепочка (программа) конечна. Число цепочек бесконечно.

ω -языки – бесконечное число бесконечных цепочек.

Каждое вычисление реагирующей программы – своя ω -цепочка.

Каждая ω -цепочка бесконечна. Число цепочек бесконечно.

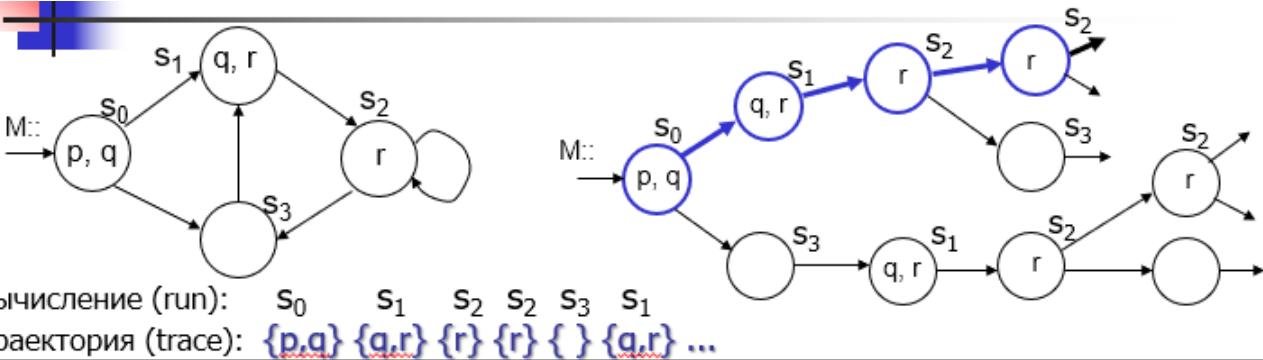
Ю.Г.Карпов

11

形式化语言生成的字符串是长度是有限的。可以通过有限自动机（如 DFA、NFA）进行验证字符串是否符合词法。

Omega语言生成的字符串长度是无限的！它的字符串只能通过通过 Büchi 自动机 或 ω -自动机 来验证和识别。

属性的描述并不是针对“状态”($s_0 s_1 s_2 s_3 s_1 s_2 \dots$), 而是针对“轨迹”($\{p,q\} \{q,r\} \{r\} \{r\} \{\} \{q,r\} \dots$)。



Свойства поведения формулируются **НЕ относительно вычислений** $S_0 S_1 S_2 S_3 S_1 S_2 \dots$ (**последовательностей состояний системы**), а **относительно траекторий** (**последовательностей подмножеств атомарных предикатов системы**).

Возможные траектории М:

- | | |
|---|--|
| $\{p,q\} \{q,r\} \{r\} \{r\} \{r\} \dots$ | Свойство $q \cup (Gr)$ выполняется |
| $\{p, q\} \{q, r\} \{r\} \{\} \{q, r\} \{r\} \dots$ | Свойство $q \cup (Gr)$ НЕ выполняется |
| $\{p,q\} \{ \} \{q,r\} \{ r \} \{ r \} \{ r \} \dots$ | Свойство $q \cup (Gr)$ НЕ выполняется |

Траектории бесконечны, и самих траекторий бесконечное число.

Можно считать эти цепочки цепочками ω -языка, задаваемого структурой Крипке.

Ю.Г.Карпов

12

要判断一个LTL公式是否在所有路径上满足属性，重点是找出所有“轨迹”，然后判断每条“轨迹”是否符合属性。

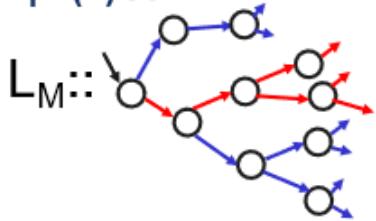
轨迹的数量是无穷的。

Model Checking для LTL.

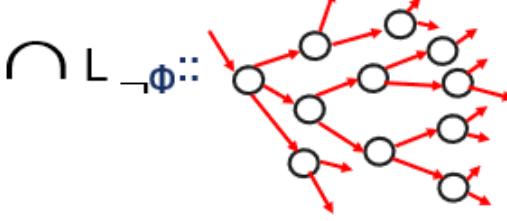
Идея – проверка пересечения языков

Все поведения структуры М Все некорректные поведения

$M \models (?) A\Phi$



\cap



Пересечение дает возможные в М некорректные поведения



ПУСТО???

- $M \models (?) A\Phi$, где Φ – формула LTL, а квантор пути **A** говорит о том, что мы проверяем, что Φ выполняется на **ВСЕХ** траекториях структуры Крипке.
- Единственный **контрпример** достаточен, чтобы опровергнуть $M \models A\Phi$.

ИДЕЯ: Проверить, совпадают ли какие-нибудь траектории M с траекториями, удовлетворяющими $\neg\Phi$

Конечным способом с помощью автомата опишем все цепочки, которые **НЕ удовлетворяют Φ , и проверим, пусто ли пересечение **ВСЕХ** этих цепочек с траекториями (языком) M .**

Ю.Г.Карпов

Верификация. Model checking

13

模型M的轨迹集合A在上图左，不满足公式 ϕ 的轨迹集合B在上图右。我们要把两个图的轨迹相交，看看有没有交集。如果有交集则说明模型中有不符合 ϕ 的轨迹的轨迹存在，则 $M \models \phi$ 不成立。

那怎么计算集合A、B呢？

方法就是使用Buchi自动机。Buchi自动机可以证明无穷序列。([[Lecture 6 Verification LTL 2]] slide24)

LTL 模型检测的基本思想 // Алгоритм Model Checking для формул LTL -----> 【重要！】

1. Структура Крипке M -> автомат Бюхи BM:

- Kripke 结构 M:

Kripke 结构表示系统的状态集合及状态间的转移关系。

- Büchi 自动机 BM:

从 Kripke 结构 M 可以构建一个 Büchi 自动机 BM，其接受所有可能的 轨迹（无限状态序列），这些轨迹表示 Kripke 结构中的状态路径。

2. Формула Φ LTL -> автомат Бюхи $B\neg\Phi$:

- LTL 公式 Φ :

要验证的线性时序逻辑 (LTL) 公式。

- $B\neg\Phi$:

将 LTL 公式 Φ 转换为一个 Büchi 自动机，但这个自动机接受 不满足公式 Φ 的所有路径 (НЕCORPEKTHIE 轨迹)。

- 重点:

$B\neg\Phi$ 接受的是 不满足公式 Φ 的轨迹。

3. Строится автомат $BM \otimes B\neg\Phi$:

- 将 BM_M 和 $B\neg\Phi$ 进行 同步组合 (交互)，得到新的 Büchi 自动机 $BM \otimes B\neg\Phi$ 。

- 作用:

- 这个自动机接受 Kripke 结构 m 上的轨迹，且这些轨迹在 LTL 公式 Φ 下是 不满足条件的 (错误路径)。
- 等价于计算 $BM \cap B\neg\Phi$ 。

4. Проверяется, допускает ли автомат $BM \otimes B\neg\Phi$ непустой язык:

- 检查自动机

$BM \otimes B\neg\Phi$ 是否 接受非空语言:

- 如果存在一条路径被接受 (即进入 循环 并包含一个 接受状态)，说明存在一个 不满足公式 Φ 的路径。

- 两种结果:

- 非空语言: 公式 Φ 不满足 Kripke 结构 MM。
- 空语言: 公式 Φ 满足 Kripke 结构 MM。

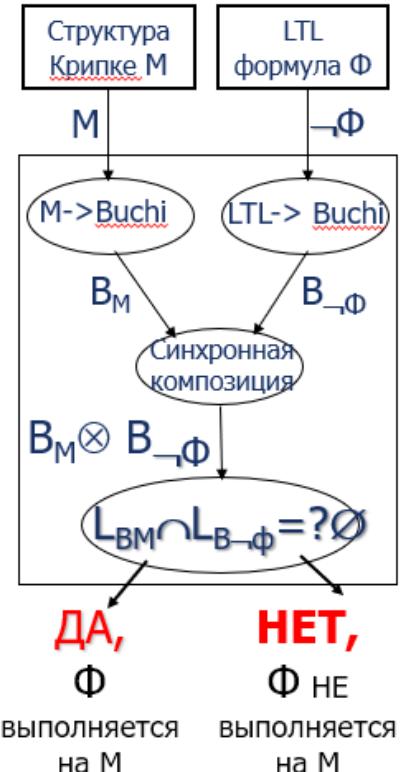
5. LTL формула Φ выполняется для M, если и только если $BM \otimes B\neg\Phi$ допускает пустой язык.

- 结论:

- 如果组合自动机 $BM \otimes B\neg\Phi$ 接受的语言为空 (即不存在错误路径)，则 LTL 公式 Φ 在 Kripke 结构 MM 上成立。
- 反之，如果语言非空，则公式不成立。

Алгоритм Model Checking для формул LTL

- Структура Кripке $M \rightarrow$ автомат Бюхи B_M .
Автомат B_M допускает все возможные траектории структуры M (цепочки подмножеств атомарных предикатов структуры Кripке M).
- Формула Φ LTL \rightarrow автомат Бюхи $B_{\neg\Phi}$.
Этот автомат допускает множество вычислений, которые **НЕ** удовлетворяют формуле Φ . Т.е. автомат $B_{\neg\Phi}$ допускает все **НЕКОРРЕКТНЫЕ** траектории.
- Строится автомат $B_M \otimes B_{\neg\Phi}$. Этот автомат допускает пересечение языков, допускаемых автоматом Бюхи M и автоматом Бюхи, допускающим все **НЕКОРРЕКТНЫЕ** траектории.
- Проверяется, допускает ли автомат $B_M \otimes B_{\neg\Phi}$ **НЕпустой** язык, т.е. достижимо ли в цикле хотя бы одно финальное состояние.
- LTL формула Φ выполняется для M , если и только если $B_M \otimes B_{\neg\Phi}$ допускает пустой язык.



Ю.Г.Карпов

Верификация. Model checking

26

△ Автоматы Бюхи Büchi 自动机

- Автоматы Бюхи, их формальное определение, примеры.
- Построение **по структуре Кripке M** такого автомата Бюхи B_M , который допускает все возможные вычисления структуры M .
- Автомат Бюхи и формулы LTL. Алгоритм построения по формуле Φ автомата Бюхи B_Φ
- Синхронная композиция двух автоматов Бюхи.
- Алгоритм проверки пустоты языка, допускаемого автоматом Бюхи.
- Buchi自动机的形式化定义和示例
- 根据Kripke结构（模型M）构建Buchi自动机
- 根据LTL公式构建Buchi自动机的算法
- 两个自动机的同步合成
- 检查Buchi自动机允许的语言的非空性的算法。

现在来认识一下Buchi自动机。

先来看正则表达式：

$$E ::= \emptyset | \varepsilon | A | E + E' | E \cdot E' | E^*$$

这里的 E^* 表示 E 可以重复有限次。但如果写成了： E^ω 则代表无限次重复。

如果语言带上了 omega，则记为： L^ω ，被称为omega语言。

NBA 的定义：

A **non-deterministic Büchi automaton**, later referred to just as a **Büchi automaton**, has a transition function which may have multiple outputs, leading to many possible paths for the same input; it accepts an infinite input if and only if some possible path is accepting.

NBA的形式化定义：

Formally, a **deterministic Büchi automaton** is a tuple $A = (Q, \Sigma, \delta, q_0, F)$ that consists of the following components:

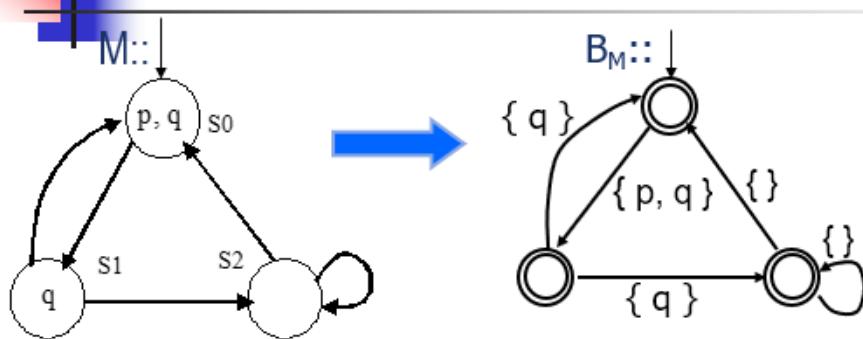
- Q is a **finite set**. The elements of Q are called the *states* of A .
- Σ is a finite set called the *alphabet* of A .
- $\delta: Q \times \Sigma \rightarrow Q$ is a function, called the *transition function* of A .
- q_0 is an element of Q , called the *initial state* of A .
- $F \subseteq Q$ is the *acceptance condition*. A accepts exactly those runs in which at least one of the infinitely often occurring states is in F .

In a (**non-deterministic**) **Büchi automaton**, the transition function δ is replaced with a transition relation Δ that returns a set of states, and the single initial state q_0 is replaced by a set I of initial states. Generally, the term Büchi automaton without qualifier refers to non-deterministic Büchi automata.

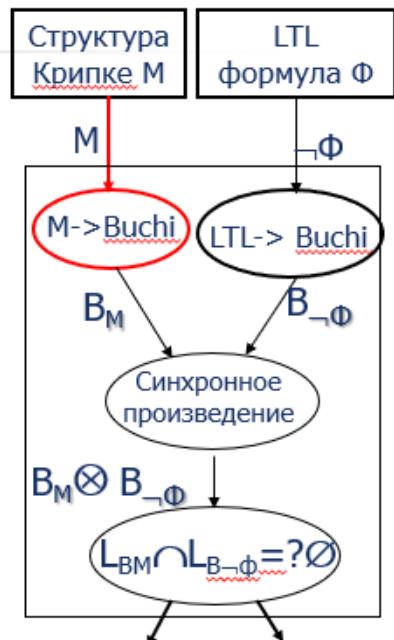
根据 Kripke 结构构建Buchi自动机： (p33)

// Построение по структуре Крипке M такого автомата Бюхи BM, который допускает все возможные вычисления структуры M

Структура Крипке \Rightarrow автомат Бюхи



- Входными символами (событиями) автомата Бюхи в нашей теории являются **подмножества** атомарных предикатов (то, что может стоять в состояниях структуры Крипке).
- Каждое состояние структуры Крипке $M \rightarrow$ принимающее состояние автомата Бюхи B_M .



Каждая выходная стрелка из состояния s автомата B_M помечается множеством $L(s)$ – атомами, истинными в s .

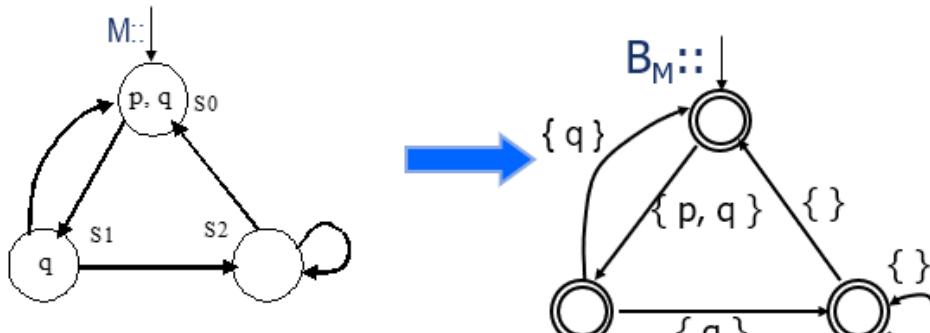
Структуру Крипке M можно рассматривать как автомат Бюхи B_M , принимающий язык L_M над словарем 2^{AP} . Т.е. входной словарь автомата Бюхи – множество подмножеств AP структуры Крипке. Все состояния B_M – принимающие!

Ю.Г.Карпов

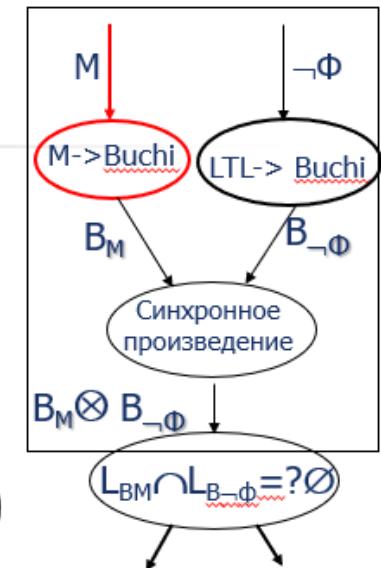
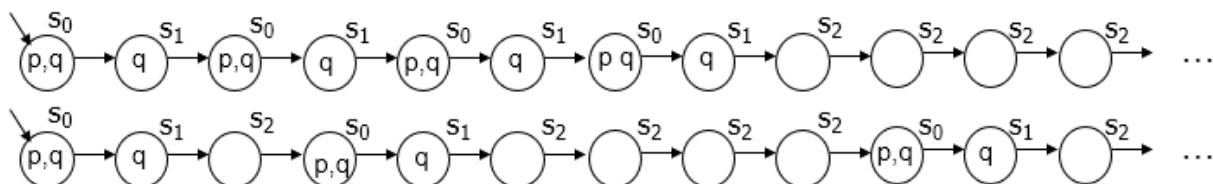
Верификация. Model checking

33

Структура Крипке \Rightarrow автомат Бюхи



Вычисления M :



Примеры цепочек языка, допускаемого B_M :

{p,q} {q} {p,q} {q} {p,q} {q} (p,q) {q} {} {} {} ...
 {p,q} {q} {} {p,q} {q} {} {} {} {p,q} {q} {} {} ...

Ю.Г.Карпов

Верификация. Model checking

34

从别的PPT中读到的：



阅读这篇PPT: presentation_salama

详细说说, 怎样一步一步把一个Kripke结构转化成一个Buchi自动机?
要详细说说Buchi自动机 $B = \langle Q, \Sigma, \Delta, Q_0, F \rangle$ 中各个集合都是怎样生成的。



如何将Kripke结构转化为Büchi自动机

将 Kripke 结构 转化为 Büchi 自动机 的目的是表示系统的无限路径, 尤其是对 LTL 公式 进行验证时所需的模型。以下是详细的步骤和构建规则:

Büchi自动机的定义

Büchi自动机 B 是一个五元组:

$$B = \langle Q, \Sigma, \Delta, Q_0, F \rangle$$

其中:

1. Q : 状态集合 — 包含所有可能的状态。
 2. Σ : 输入字母表 — 对应Kripke结构中的原子命题的幂集 $P(AP)$ 。
 3. Δ : 转移关系 — 从一个状态到另一个状态的路径, 由Kripke结构的转移关系决定。
 4. Q_0 : 初始状态集合 — Kripke结构的初始状态。
 5. F : 接受状态集合 — 所有状态都被标记为接受状态。
-

步骤: 将Kripke结构转化为Büchi自动机

假设我们有一个Kripke结构 M :

$$M = \langle S, S_0, R, L \rangle$$

其中：

- S : 状态集合。
- S_0 : 初始状态集合。
- $R \subseteq S \times S$: 状态之间的转移关系。
- $L : S \rightarrow 2^{AP}$: 标签函数，将状态映射到一组原子命题 AP 。

1. 构建Büchi自动机的状态集合 Q

- 取 $Q = S \cup \{i\}$, 即Kripke结构的所有状态, 再添加一个虚拟的初始状态 i 。
- 这个虚拟初始状态 i 用于引入到Kripke结构初始状态的转移。

2. 构建输入字母表 Σ

- 输入字母表是所有可能的原子命题组合的幂集：

$$\Sigma = P(AP)$$

例如, 如果 $AP = \{P, Q\}$, 那么 $\Sigma = \{\emptyset, \{P\}, \{Q\}, \{P, Q\}\}$ 。

3. 构建转移关系 Δ

- 转移关系从Kripke结构的转移关系 R 推导而来：
 - 若 $(s, s') \in R$, 则在Büchi自动机中存在一条从 s 到 s' 的转移:

$$(s, L(s'), s') \in \Delta$$

其中 $L(s')$ 是目标状态 s' 的标签集合。

- 添加从虚拟初始状态 i 到Kripke结构中所有初始状态 S_0 的转移：

$$\forall s_0 \in S_0 : (i, L(s_0), s_0) \in \Delta$$

4. 指定初始状态集合 Q_0

- 初始状态集合是虚拟的初始状态 i :

$$Q_0 = \{i\}$$

5. 指定接受状态集合 F

- 所有状态都被标记为接受状态：

$$F = Q$$

这是因为Kripke结构中的无限路径会自动满足Büchi自动机的接受条件（无限次访问某个状态）。

示例：

同步两个Buchi自动机【todo】

// Синхронная композиция двух автоматов Бюхи

两个Buchi自动机的交叉乘积 (cross-product)

交叉乘积 (cross-product) 是将两个 Büchi 自动机 B_1 和 B_2 合并为一个新的 Büchi 自动机 B , 其语言是 B_1 和 B_2 语言的交集, 即:

$$L(B) = L(B_1) \cap L(B_2)$$

交叉乘积的构造通过组合两个自动机的状态、转移关系和接受条件来完成。

△ 根据LTL公式构建Buchi自动机的算法

// Автомат Бюхи и формулы LTL. Алгоритм построения по формуле Φ автомата Бюхи $\text{B}\Phi$

Существует несколько алгоритмов построения автомата Бюхи по заданной формуле LTL, Они называются *LTL2Buchi*.

Сложность автомата Бюхи экспоненциальна относительно сложности формулы Φ

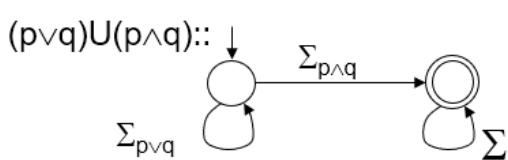
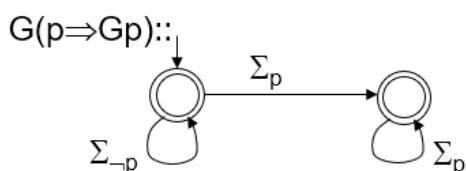
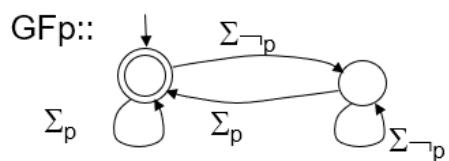
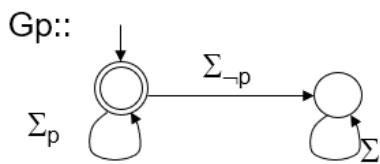
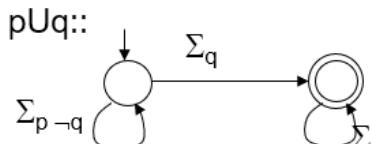
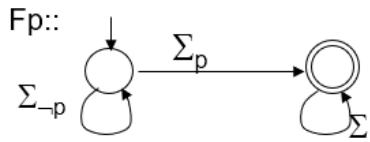
网站: <http://www.lsv.fr/~gastin/ltl2ba/index.php>

例子: 根据LTL公式构造的Buchi自动机 (好像更多的是根据经验, 或者用网站生成)

Примеры автоматов Бюхи, допускающие цепочки, удовлетворяющие LTL формулам

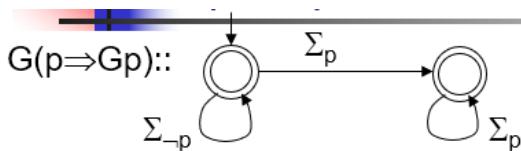
$AP = \{p\}; \Sigma = \{\emptyset, \{p\}\}; \Sigma_{\neg p} = \{\emptyset\}; \Sigma_p = \{\{p\}\}$

$AP = \{p, q\}; \Sigma = \{\emptyset, \{p\}, \{q\}, \{p, q\}\}; \Sigma_{\neg p} = \{\emptyset, \{q\}\}; \Sigma_p = \{\{p\}, \{p, q\}\}; \Sigma_{p \neg q} = \{\{p\}\};$



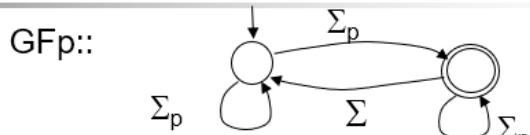
Ю.Г.Карпов

41



LTL formula: $G(p \Rightarrow Gp)$

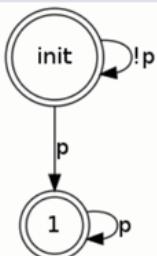
- Use Spin syntax
- Use Spin 4.3.0
- Display an image of the generalised Büchi automaton
- Display an image of the Büchi automaton
- echo a never claim for Spin
- Use verbose mode
- Display time and size statistics
- Enable on-the-fly automata simplification
- Enable a posteriori automata simplification
- Enable strongly connected components simplification
- Consider second set in $f_j \rightarrow f_j$ (internal use)



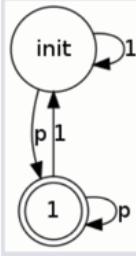
LTL formula: GFp

- Use Spin syntax
- Use Spin 4.3.0
- Display an image of the generalised Büchi automaton
- Display an image of the Büchi automaton
- echo a never claim for Spin
- Use verbose mode
- Display time and size statistics
- Enable on-the-fly automata simplification
- Enable a posteriori automata simplification
- Enable strongly connected components simplification
- Consider second set in $f_j \rightarrow f_j$ (internal use)

Büchi automaton



Büchi automaton

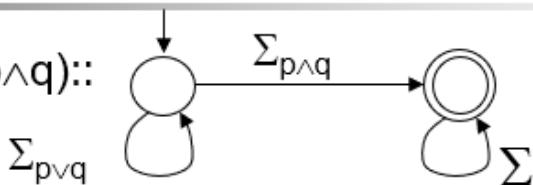


Ю.Г.Карпов

42

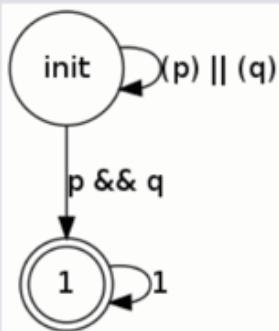
$(p \vee q) \cup (p \wedge q) ::$

LTL formula: $(p \parallel q) \cup (p \& \& q)$



- Use Spin syntax
- Use Spin 4.3.0
- Display an image of the generalised Büchi automaton
- Display an image of the Büchi automaton
- echo a never claim for Spin
- Use verbose mode
- Display time and size statistics
- Enable on-the-fly automata simplification
- Enable a posteriori automata simplification
- Enable strongly connected components simplification
- Consider second set in $f_j \rightarrow f_j$ (internal use)

Büchi automaton



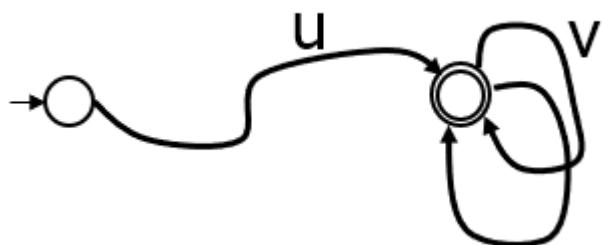
■ И

Ю.Г.Карпов

检查 Büchi 自动机允许的语言的非空性

// Проблема пустоты автомата Бюхи

omega字符串 σ 能被Buchi自动机A接受，当且仅当终止状态A在输入字符串时无限多次发生。



如果从自动机的初始状态出发，存在一条到接受状态的路径，且该状态有自循环，那么说明Buchi自动机接受的语言非空。

Контрпример: цепочка, переводящая автомат в цикл принимающих состояний

反例：将自动机转移到一个接收状态循环的字

检查 Büchi 自动机语言非空性的算法：

1. Находим все Сильно Связные Компоненты графа переходов.
2. Оставляем те ССК, в которых есть хотя бы одно допускающее состояние.
3. Проверяем, есть ли путь из S_0 хотя бы в одну оставшуюся ССК.

Каждая цепочка из начального состояния в ССК с принимающими состояниями определяет **контрпример -- вычисление, на котором не выполняется Φ**

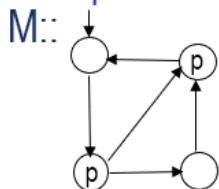
1. 找到所有强连通 (ССК, Сильно связные компоненты) 【即回路! 环!】
2. 只保留包含了可接受终态的强连通
3. 检查是否存在从 s_0 出发的包含可接受终态的强连通存在

如果不存在则说明 Büchi 自动机的语言为空。否则，每一个从初始状态到达接受终态的串都是自动机的语言。（即：一个不满足 Φ 性质的反例！！）

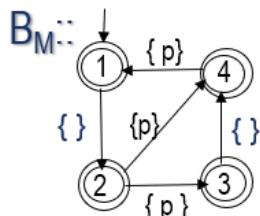
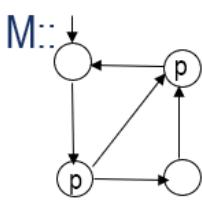
例题：

Пример: Model Checking для LTL

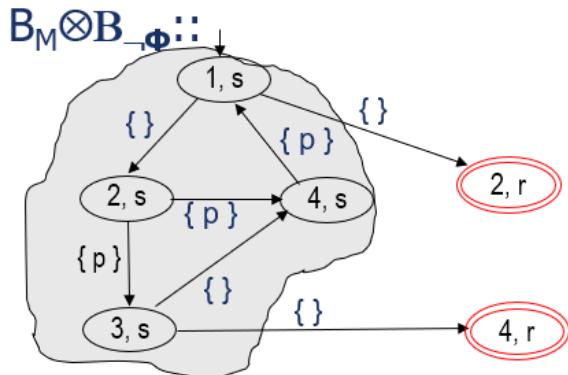
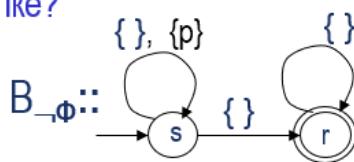
Удовлетворяет ли формуле $\Phi = GFp$ структура Кripке?



Если какое-либо вычисление **не удовлетворяет** $\Phi = GFp$, то оно должно удовлетворять $\neg\Phi = \neg GFp = FG \neg p$



$$\neg\Phi = FG \{\}$$



ССК без финальных состояний

Поскольку нет достижимой ССК, включающей **принимающее состояние**, язык $L_{B_M \otimes B_{\neg\Phi}}$ пуст. Следовательно, на структуре Кripке формула GFp выполняется

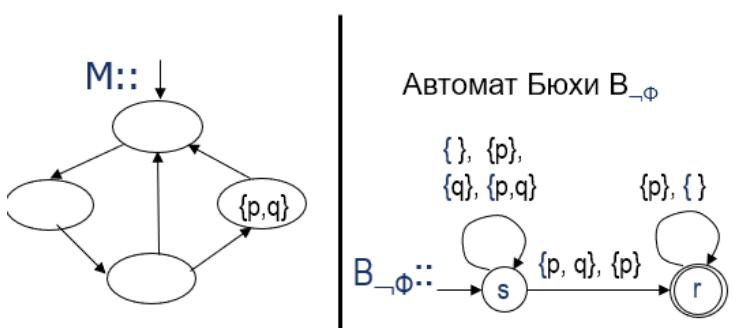


Model Checking для LTL: Пример

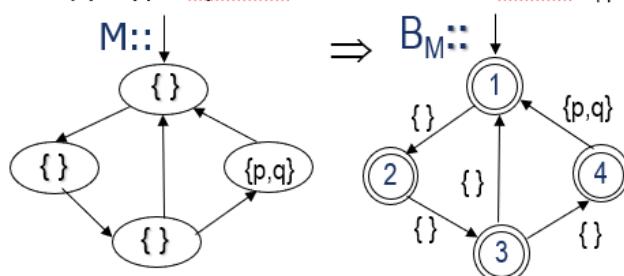


Выполняется ли формула $\Phi = G(p \Rightarrow X F q)$ на структуре Кripке M?

$$\neg\Phi = \neg G(p \Rightarrow X F q) = F(p \wedge X G \neg q)$$



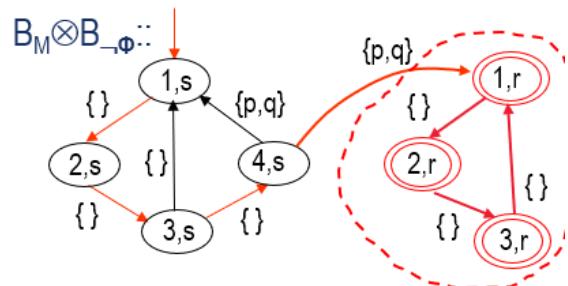
Структура Кripке M \Rightarrow автомат Бюхи B_M



Ю.Г.Карпов

Верификация. Model checking

Композиция автоматов Бюхи:



Есть цикл, включающий принимающее состояние \Rightarrow
язык $L_{B_M \otimes B_{-\Phi}}$ непуст.
M не удовлетворяет Φ .
Контрпример: 1234(123) $^\omega$

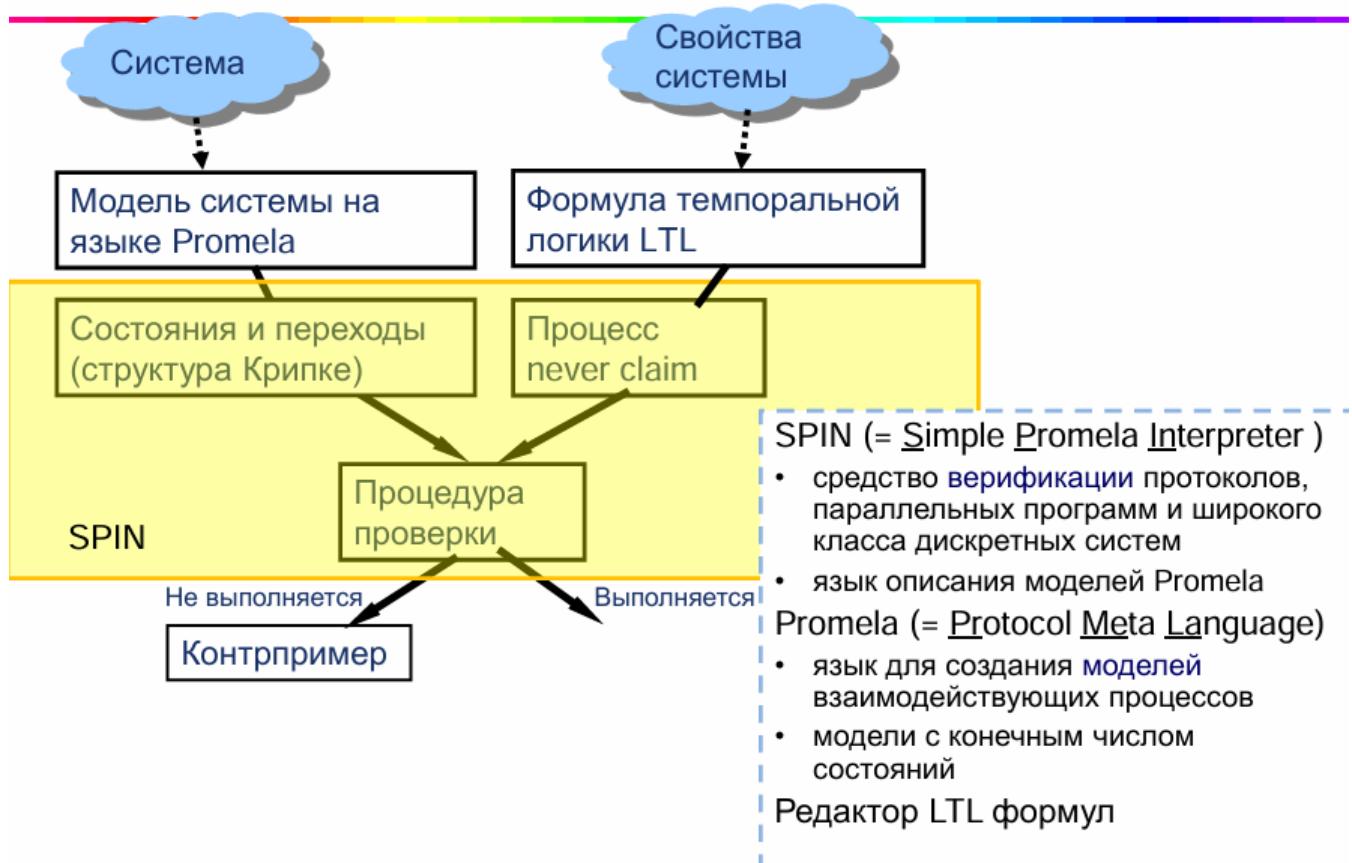
49

Spin可以充当模型-LTL公式检测器。具体来说，只要用Promela语言描述系统模型，然后再描述ltl公式，将这两个描述作为输入输入到spin，spin会自动检查模型是否满足ltl公式。（方法也是基于Buchi自动机）

Spin的使用 tbd

什么是SPIN?

Что такое SPIN?



Promela语言:

Modeling by Kripke structure

通过Kripke结构进行建模

要解决的问题：Как преобразовать программу в структуру Крипке？

Символьная верификация CTL 符号模型检测CTL

Применение ТЛ

- Примеры спецификации требований на LTL *todo* 【重要！考题！】

- Примеры спецификации требований на CTL

Примеры проверяемых свойств (CTL)

■ $\text{at_start} \wedge \varphi \Rightarrow \text{AG}(\text{at_finish} \Rightarrow \psi)$

- частичная корректность: если в начале программы (at_start) выполняется предусловие φ , то на любом вычислении (AG) если программа завершается (at_finish), то выполняется постусловие ψ

■ AGAF Restart

- при любом функционировании системы (на любом пути) из любого состояния системы всегда обязательно вернемся в состояние рестарта

■ AG EF Restart

- при любом функционировании системы (на любом пути) из любого состояния системы существует путь, по которому можно перейти в состояние рестарта

■ E[p U A [q U r]]

- существует путь, на котором p выполняется до тех пор, пока событие q станет выполняться до выполнения r на всех путях

Ю.Г.Карпов

Верификация

13

- Применение шаблонов спецификации требований

Dwyer M.B., Avrunin G.S., Corbett J.C. Property Specification Patterns for Finite-state Verification // Proceedings of the 2nd workshop on Formal Methods in Software Practice. – 1998

Предлагается система шаблонов, которая помогает практическому программисту точно специфицировать требования

Система шаблонов состоит из Типа и Области действия требований

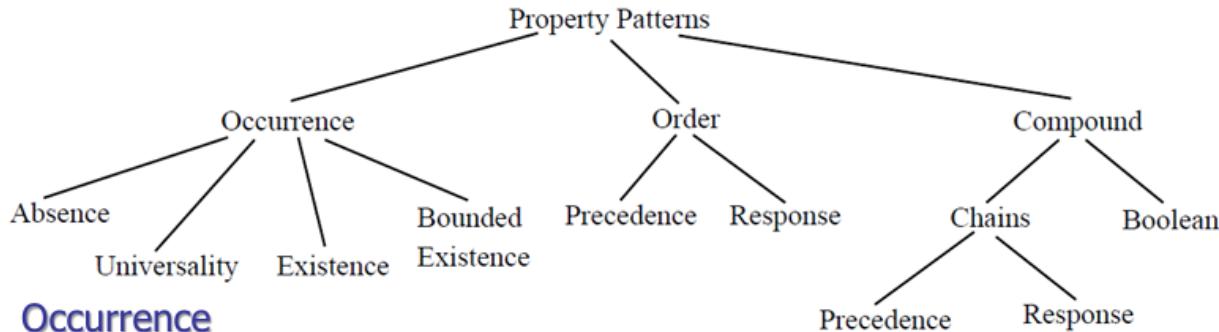
Эта работа дает спецификацию на LTL и CTL каждого типа требования для каждой области действия.

提出了一个模板系统，可以帮助实际程序员准确地指定需求。

模板系统由需求类型和范围组成

这项工作为每个范围的每个需求类型的 LTL 和 CTL 提供了规范。

Структурирование шаблонов – типы требований



■ Occurrence

- **Absence** Событие не встречается
- **Universality** Событие встречается
- **Existence** Событие должно наступить когда-нибудь
- **Bounded Existence** Событие должно наступить k раз (at least k; at most k)

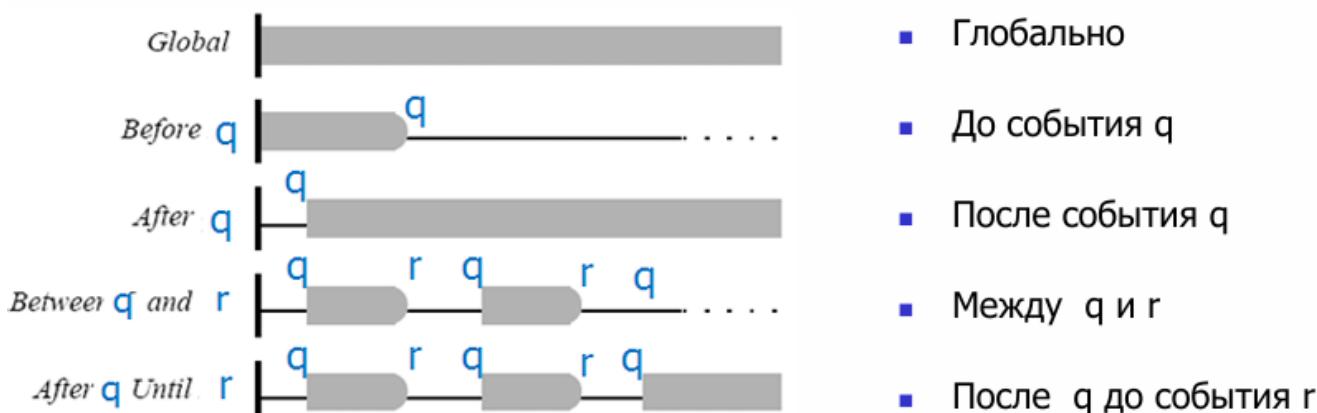
■ Ordering

- **Precedence** Событие p всегда предшествует событию q
- **Response** Событие p всегда следует за событием q (причинная связь)

■ Compound

- **Chain Precedence** Одна цепочка событий всегда предшествует другой
- **Chain Response** Одна цепочка событий всегда следует за другой

Свойство p должно выполняться:

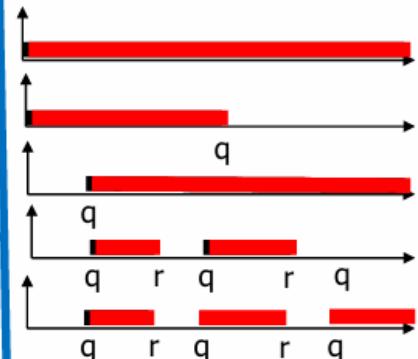


关于“出现(Absence)”的模板：

событие p не выполняется

LTL

- Глобально $G \neg p$
- До q $F q \Rightarrow \neg p \cup q$
- После q $G(q \Rightarrow G \neg p)$
- Между q и r $G((q \wedge X F r) \Rightarrow (\neg p \wedge X(\neg p \cup r)))$
- После q до r $G(q \Rightarrow (\neg p \wedge X(\neg p \cup (r \vee G(\neg r \wedge \neg p)))))$



CTL

- Глобально $AG \neg p$
- До q $A(\neg p \cup (q \vee AG \neg q))$
- После q $AG(q \Rightarrow AG \neg p)$
- Между q и r $AG(q \Rightarrow A(\neg p \cup (r \vee AG \neg r)))$
- После q до r $AG(q \Rightarrow \neg E(\neg r \cup (p \wedge \neg r)))$

Ю.Г.Карпов

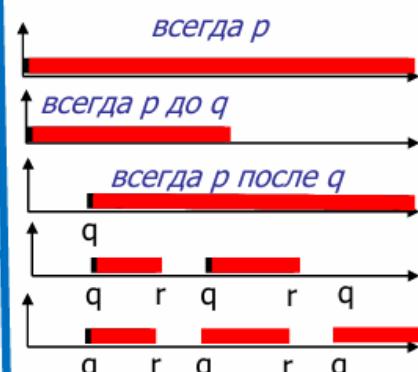
Верификация

29

Требование всеобщности выполнения события p (universality)

LTL

- Глобально $G p$
- До q $F q \Rightarrow p \cup q$
- После q $G(q \Rightarrow G p)$
- Между q и r $G((q \wedge X F r) \Rightarrow (p \wedge X(p \cup r)))$
- После q до r $G(q \Rightarrow (p \wedge X(p \cup (r \vee G(\neg r \wedge p))))))$



CTL

- Глобально $AG p$
- До q $A[(p \vee AG \neg q) W q]$
- После q $AG(q \Rightarrow AG p)$
- Между q и r $AG(q \wedge \neg r \Rightarrow A(p \vee AG \neg r) W r))$
- После q до r $AG(q \wedge \neg r \Rightarrow A(p W r))$

Ю.Г.Карпов

Верификация

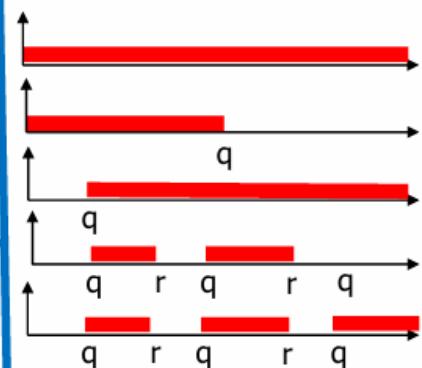
30

Шаблон "существование": событие p должно выполниться, по крайней мере, один раз (Existence)

LTL

- Глобально $\mathbf{F}p$
- До q $\neg q \mathbf{W} (\mathbf{p} \wedge \neg q)$
- После q $G \neg q \vee \neg q \mathbf{U} (q \wedge \mathbf{X} F p)$
- Между q и r $G (q \wedge \neg r \Rightarrow (\neg r \mathbf{W} (\mathbf{p} \wedge \neg r)))$
- После q до r $G (q \wedge \neg r \Rightarrow (\neg r \mathbf{U} (\mathbf{p} \wedge \neg r)))$

Выполнение p :



CTL

- Глобально $\mathbf{AF}p$
- До q $A(\neg q \mathbf{W} (\mathbf{p} \wedge \neg q))$
- После q $A(\neg q \mathbf{W} (q \wedge \mathbf{AF} p))$
- Между q и r $AG (q \wedge \neg r \Rightarrow A(\neg r \mathbf{W} (\mathbf{p} \wedge \neg r)))$
- После q до r $AG (q \wedge \neg r \Rightarrow A(\neg r \mathbf{U} (\mathbf{p} \wedge \neg r)))$

Ю.Г.Карпов

Верификация

31

- Формулировка ОБЩИХ требований к программным системам логического управления, отражающих их ПРИРОДУ как систем ПАРАЛЛЕЛЬНЫХ ВЗАИМОДЕЙСТВУЮЩИХ ПРОЦЕССОВ
 - Достигимость (reachability) – конкретное состояние может быть достигнуто: $\mathbf{EF}\phi$
 - Безопасность (safety)- нечто плохое никогда не произойдет : $\mathbf{AG}\neg\phi$
 - Свобода от дедлоков (deadlock freeness) – проверка блокировок:
 - Живость, живучесть (liveness) - нечто хорошее обязательно произойдет (something good happens): $\mathbf{AG}(\mathbf{req} \rightarrow \mathbf{AF}\mathbf{resp})$ в CTL, $\mathbf{G}(\mathbf{req} \rightarrow \mathbf{F}\mathbf{resp})$ в LTL
 - Справедливость (fairness) – дополнительные ограничения: $\mathbf{GF} \text{ succeed}$ 强公平性、弱公平性...

题型总结

Hoare逻辑(弗洛伊德法)

判断程序的正确性(wp, sp)

Bap1, 1

Bap7, 2

使用 wp 判断程序的正确性

使用 sp 判断程序的正确性

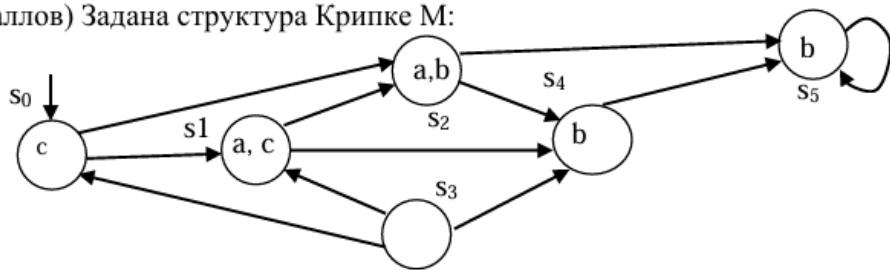
Крипке

给定 Крипке + CTL 公式，判定在哪些状态下公式能满足？（必考）

递归标记算法

Var1, 1

1. (15 баллов) Задана структура Крипке M:



和公式 $E(c \text{UAX} b)$ 逻辑 CTL。在哪些状态下结构 M 这个公式满足？

Крипке + LTL + автомат Бюхи

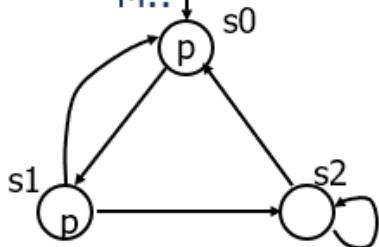
将公式转化为Buchi自动机，然后验证路径是否接受

Var2, 2

解法： (L6,S37)

Пример: проверка выполнения требования в структуре Крипке

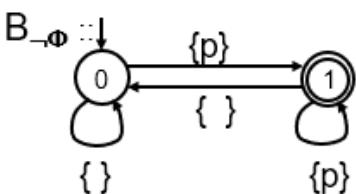
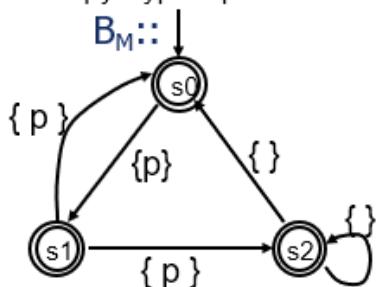
Структура Крипке $M::$



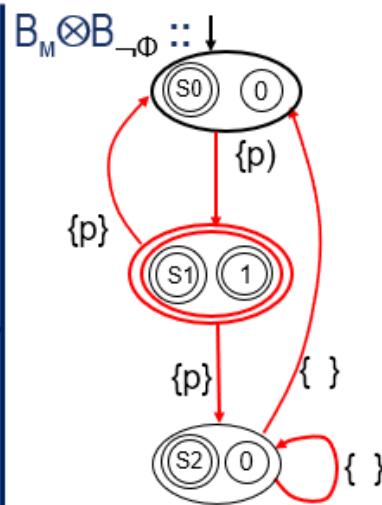
Проверим, выполняется ли на M требование $\Phi = FG\neg p$

Отрицание формулы Φ :
 $\neg\Phi = \neg(FG\neg p) = GFp$

Автомат B_M получен из структуры Крипке M



Автомат Бюхи B допускает все цепочки, удовлетворяющие свойству GFp



Автомат $B_M \otimes B_{-\Phi}$ допускает цепочку
 $\{p\} \{p\} \{p\} \{p\} \{p\} \dots$

Поскольку в автомате $B_M \otimes B_{-\Phi}$ достижим цикл с принимающим состоянием, то в структуре M формула Φ не выполняется. Контрпример: $s_0 s_1 s_0 s_1 s_0 s_1 \dots$

Ю.Г.Карпов

Верификация. Model checking

37

LTL

► 用LTL公式表达系统属性(Спецификации требований на LTL)

考试题:

Бап1,2;

2. (15 баллов) Выразите формулой логики LTL следующие свойства поведений:

- а) Если произойдет событие p , то после него событие q произойдет только после события r .
- б) Атомарные предикаты p и q в одном состоянии не встречаются, и если выполнится p , то после этого p не будет выполняться, пока не выполнится q , и наоборот.
- в) Каждое событие $alarm$ происходит только если до него когда-то случалось событие $fault$.

解题方法1：熟练掌握 G, F, X, U 的语义

Утверждение q будет истинным:

- всегда в будущем: Gq
- хотя бы раз в будущем: Fq
- никогда в будущем: $\neg Fq$
- бесконечно много раз в будущем: GFq
- с какого-то момента постоянно: FGq

Рассмотрим, как с помощью этих операторов формально записать некоторые утверждения.

- События p и q никогда не произойдут одновременно:

$$G\neg(p \wedge q)$$

- Любое посланное сообщение когда-нибудь в будущем будет получено:

$$G(\text{Послано}_m \Rightarrow F \text{ Получено}_m)$$

- Джейн вышла замуж и родила ребенка:

$$P(\text{Джейн_выходит_замуж} \wedge F \text{ Джейн_рожает_ребенка})$$

- Джейн родила ребенка и вышла замуж:

$$P(\text{Джейн_рожает_ребенка} \wedge F \text{ Джейн_выходит_замуж})$$

- Ленин жил, Ленин жив, Ленин будет жить:

$$PG \text{ Ленин_жив}$$

- Пока ключ зажигания не вставлен, машина не поедет:

$$G(\neg P \text{ Зажигание} \Rightarrow \neg \text{Старт})$$

В темпоральную логику можно ввести еще два оператора: *NextTime* (**X**) и *Until* (**U**).

Оператор *NextTime*: утверждение Xq истинно в момент времени t , если q истинно в следующий момент $t+1$:

- Джон убил, и ему стало страшно:

$$P(\text{Джон_убивает} \wedge XG \text{Джону_страшно}).$$

- Если я видел ее раньше, то я ее узнаю при встрече:

$$G(P \text{ Увидел} \Rightarrow G(V \text{ встретил} \Rightarrow X \text{ Узнал})).$$

- Джон умер и его похоронили:

$$P(\text{Джон_умирает} \wedge XF \text{Джона_хоронят}).$$

□ Мы не друзья, пока ты не извинишься:

$$(\neg \text{Мы_друзья}) \mathbf{U} \text{Ты_извиняешься}$$

□ Лифт никогда не пройдет мимо этажа, вызов от которого поступил, но еще не обслужен:

$$\mathbf{G}(\text{Вызов}(n) \Rightarrow (\neg \text{Лифт}(n)) \mathbf{U} \text{Обслуживается}(n))$$

Через оператор *Until* легко выражается оператор **F**:

$$\mathbf{F}q = \text{true} \mathbf{U} q$$

а следовательно, и оператор **G**:

$$\mathbf{G}q = \neg(\mathbf{F} \neg q) = \neg(\text{true} \mathbf{U} \neg q).$$

п68

Пример 2.2

Рассмотрим примеры записи некоторых свойств бесконечных вычислений с помощью формул LTL.

- a) $\mathbf{G}(q \Rightarrow \mathbf{X}\mathbf{G}\neg q)$ — q встретится в будущем не более одного раза;
- б) $\mathbf{F}q \wedge \mathbf{G}(q \Rightarrow \mathbf{X}\mathbf{G}\neg q)$ — q встретится в будущем точно один раз;
- в) $p \Rightarrow \mathbf{F}q$ — на p , наступившем в начальном состоянии, когда-нибудь в будущем будет реакция q ;
- г) $\mathbf{G}(p \Rightarrow \mathbf{F}q)$ — на любое встретившееся в вычислении p всегда в будущем будет реакция q ;
- д) $\mathbf{G}(p \Rightarrow p \mathbf{U} q)$ — всегда если запрос p будет подан, реакция q на него обязательно будет получена, а до ее получения запрос p не сбросится;
- е) $\mathbf{F}q \Rightarrow (\neg p) \mathbf{U} q$ — если событие q наступит, то до его наступления событие p не наступит.

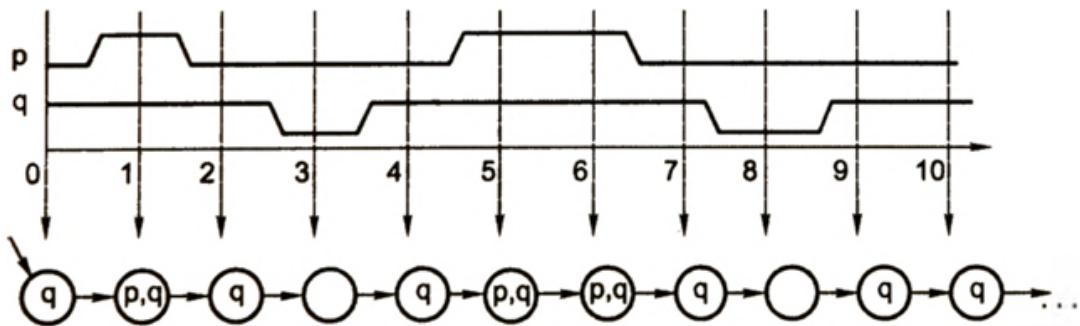


Рис. 2.10. Пример временной диаграммы и ее модели-вычисления

Пример 2.3

Свойства поведения дискретных логических схем тоже могут быть описаны с помощью формул логики LTL. На рис. 2.10 приведена временная диаграмма — график значений двух сигналов на выходе электронной схемы. Состояния схемы фиксируются в каждом такте.

Если единичное значение потенциала принять за логическое значение *true*, а низкое — за логическое значение *false*, то этой временной диаграмме можно сопоставить ее модель — вычисление, в каждом состоянии которого заданы истинностные значения атомарных предикатов *p* и *q*.

Считая, что сигналы на выходе этой схемы стабилизировались, на этом вычислении (назовем его σ) можно интерпретировать (определить истинностное значение) любую формулу LTL с атомарными предикатами *p* и *q*. В частности, для этой схемы выполняются следующие свойства:

- $\sigma \models \mathbf{F}\mathbf{G}\neg p$: на вычислении σ когда-нибудь в будущем сигнал *p* сбросится и останется в этом сброшенном состоянии;
- $\sigma \models \mathbf{F}\mathbf{G}q$: когда-нибудь в будущем сигнал *q* установится и останется в этом состоянии постоянно;
- $\sigma \models \mathbf{G}(p \Rightarrow q)$: в любом состоянии если установлен сигнал *p*, то и сигнал *q* установлен;
- $\sigma \models q \mathbf{U}(\neg p \wedge \neg q)$: оба сигнала, *p* и *q*, когда-нибудь в будущем будут сброшены, а до этого сигнал *q* будет все время установлен.

Последнее свойство выполняется на вычислении σ , потому что оно выполняется в начальном состоянии $\sigma(0)$ этого вычисления. Действительно, оба сигнала, *p* и *q*, в состоянии $\sigma(3)$ будут сброшены, а во всех предыдущих состояниях, $\sigma(2)$ $\sigma(1)$ и $\sigma(0)$, сигнал *q* установлен.

Формулы LTL могут служить для задания множеств бесконечных цепочек (так называемых ω -языков). Например, формула

$$(a \vee b) \mathbf{U}(\mathbf{G}b)$$

задает все бесконечные цепочки из *a* и *b*, содержащие только конечное число вхождений символа *a* (подробнее этот вопрос рассматривается в гл. 4).

Существует множество других соотношений между операторами LTL. Например, общезначимые (всегда истинные) соотношения:

$\mathbf{G}p \Rightarrow \mathbf{F}p$ — "то, что всегда будет, то когда-нибудь наступит".

$p \Rightarrow \mathbf{F}p$ — "то, что есть, то когда-нибудь наступит".

$(\mathbf{G}p \vee \mathbf{G}q) \Rightarrow \mathbf{G}(p \vee q)$ — "если всегда будет истинно *p* или всегда будет истинно *q*, то всегда будет истинно *p* \vee *q*".

$\mathbf{G}(p \Rightarrow q) \Rightarrow (\mathbf{G}p \Rightarrow \mathbf{G}q)$ — "если наступление *p* всегда влечет наступление *q*, то если *p* будет истинно всегда, то и *q* будет истинно всегда".

解题方法2：找出PPT中所有Примеры спецификации требований на LTL

- PPT 《Модуль 10. Применение темпоральных логик для выражения свойств》 - s12

► 证明LTL公式的等价性 // 构造反例 证明两个LTL之间的不同

Var1,3;

3. (15 баллов). Покажите, что две LTL формулы не эквивалентны, построив вычисление, удовлетворяющее одной из формул, и не удовлетворяющее другой:

- $F(p \mathbf{U} G \neg p)$ и $p \mathbf{U}(F G \neg p)$.
- $p \Rightarrow G(p \Rightarrow Xp)$ и $Fp \wedge G(p \Rightarrow Xp)$.

(不确定) 解题方法1：分别对两个ltl构造BA，如果BA相等则说明两个ltl等价；如果不等则找出一条路径作为反例证明两个公式不等价。

(不确定) 解题方法2：熟悉LTL之间的等式转换？

Операторы \mathbf{U} , \mathbf{F} и \mathbf{G} можно определить бесконечными формулами с помощью оператора \mathbf{X} :

$$p \mathbf{U} q \equiv q \vee (p \wedge Xq) \vee (p \wedge Xp \wedge XXq) \vee \dots$$

$$\mathbf{F}q \equiv q \vee Xq \vee XXq \vee XXXq \dots$$

$$\mathbf{G}q \equiv q \wedge Xq \wedge XXq \wedge XXXq \dots$$

Можно определить эти операторы рекурсивно, сами через себя (рис. 2.11):

$$p \mathbf{U} q \equiv q \vee (p \wedge \mathbf{X}(p \mathbf{U} q))$$

$$\mathbf{F}q \equiv q \vee \mathbf{X}\mathbf{F}q$$

$$\mathbf{G}q \equiv q \wedge \mathbf{X}\mathbf{G}q$$

Эти определения ясны и без формального доказательства.

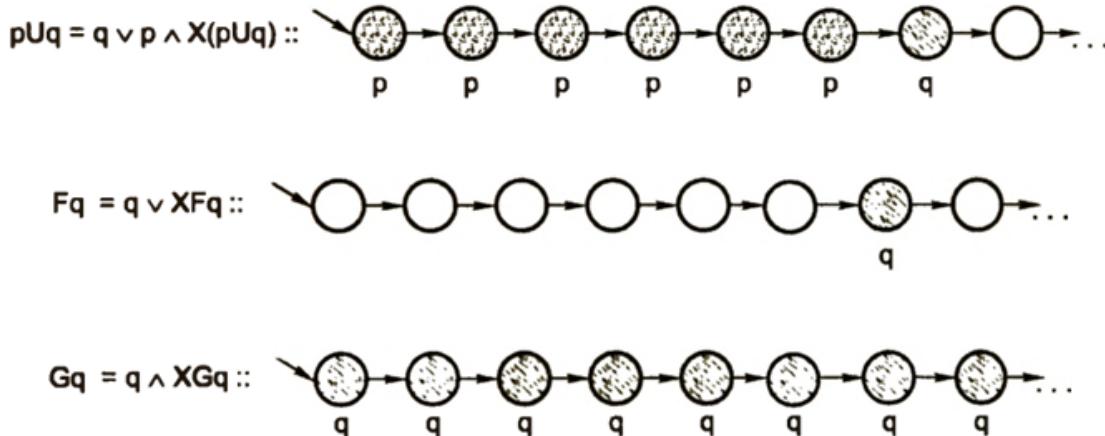


Рис. 2.11. Рекурсивное определение темпоральных операторов

Приведем несколько соотношений, которые легко могут быть доказаны на основании формального определения семантики формул LTL.

Комбинации темпоральных операторов и булевых операций:

$$\mathbf{X}(p \vee q) \equiv \mathbf{X}p \vee \mathbf{X}q$$

$$\mathbf{X}(p \wedge q) \equiv \mathbf{X}p \wedge \mathbf{X}q$$

$$\neg \mathbf{X}p \equiv \mathbf{X}\neg p$$

$$\mathbf{F}(p \vee q) \equiv \mathbf{F}p \vee \mathbf{F}q$$

$$\mathbf{G}(p \wedge q) \equiv \mathbf{G}p \wedge \mathbf{G}q$$

$$(p \wedge q) \mathbf{U}r \equiv p \mathbf{U}r \wedge q \mathbf{U}r$$

$$p \mathbf{U}(q \vee r) \equiv p \mathbf{U}q \vee p \mathbf{U}r$$

Законы поглощения (идемпотентность):

$$\mathbf{FF} p \equiv \mathbf{F}p$$

$$\mathbf{GG} p \equiv \mathbf{G}p$$

$$p \mathbf{U}(p \mathbf{U}q) \equiv p \mathbf{U}q$$

$$(p \mathbf{U}q) \mathbf{U}q \equiv p \mathbf{U}q$$

$$\mathbf{FGF} p \equiv \mathbf{GF}p$$

$$\mathbf{GFG} p \equiv \mathbf{FG}p$$

► 证明或证伪公式

Bar7,4; Bar4,4;

► 构造满足LTL公式的计算【?】

[Вариант 2,3], [Вариант 3,4],

Для каждой из формул LTL постройте несколько вычислений, удовлетворяющих ей

3. (15 баллов). Постройте несколько вычислений, удовлетворяющих формулам LTL:

- а) $p \Rightarrow XG\neg q$.
- б) $p \wedge FGr$.
- в) $F(a \wedge \neg(bUc))$.
- г) $F(p \oplus q) \Rightarrow GXq$.

⌚ Автомат Бюхи

补充资料

经典命题逻辑

我们通过如下规则定义这种真值指派A在什么时候满足特定公式：

- A满足命题变量P 当且仅当 $A(P) = \text{真}$
- A满足 $\neg\varphi$ 当且仅当A不满足 φ
- A满足 $(\varphi \wedge \psi)$ 当且仅当A满足 φ 与 ψ 二者
- A满足 $(\varphi \vee \psi)$ 当且仅当A满足 φ 和 ψ 中至少一个
- A满足 $(\varphi \rightarrow \psi)$ 当且仅当并非A满足 φ 但不满足 ψ 的情况
- A满足 $(\varphi \leftrightarrow \psi)$ 当且仅当A满足 φ 与 ψ 二者，或则不满足它们中的任何一个

逻辑等价

<https://zh.wikipedia.org/wiki/%E9%80%BB%E8%BE%91%E7%AD%89%E4%BB%B7>

等价关系	关系名称
$p \wedge T \equiv p$	Identity laws 恒等律
$p \vee F \equiv p$	
$p \vee T \equiv T$	Domination laws
$p \wedge F \equiv F$	支配律
$p \vee p \equiv p$	Idempotent laws
$p \wedge p \equiv p$	幂等律
$\neg(\neg p) \equiv p$	Double negation laws 双非律
$p \vee q \equiv q \vee p$	Commutative laws
$p \wedge q \equiv q \wedge p$	交换律
$(p \vee q) \vee r \equiv p \vee (q \vee r)$	Associative laws
$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	结合律
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	Distributive laws
$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	分配律
$\neg(p \wedge q) \equiv \neg p \vee \neg q$	De Morgan's laws
$\neg(p \vee q) \equiv \neg p \wedge \neg q$	德摩根律
$p \vee (p \wedge q) \equiv p$	Absorption laws
$p \wedge (p \vee q) \equiv p$	吸收律
$p \vee \neg p \equiv T$	Negation laws
$p \wedge \neg p \equiv F$	否定律

包含蕴含的逻辑等价、包含双蕴含的逻辑等价【重要！】

包括蕴涵的逻辑等价：

1. $p \rightarrow q \equiv \neg p \vee q$
2. $p \rightarrow q \equiv \neg q \rightarrow \neg p$
3. $p \vee q \equiv \neg p \rightarrow q$
4. $p \wedge q \equiv \neg(p \rightarrow \neg q)$
5. $\neg(p \rightarrow q) \equiv p \wedge \neg q$
6. $(p \rightarrow q) \wedge (p \rightarrow r) \equiv p \rightarrow (q \wedge r)$
7. $(p \rightarrow q) \vee (p \rightarrow r) \equiv p \rightarrow (q \vee r)$
8. $(p \rightarrow r) \wedge (q \rightarrow r) \equiv (p \vee q) \rightarrow r$
9. $(p \rightarrow r) \vee (q \rightarrow r) \equiv (p \wedge q) \rightarrow r$

包含双蕴涵（逻辑双条件）的逻辑等价：

1. $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$
2. $p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$
3. $p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
4. $\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$