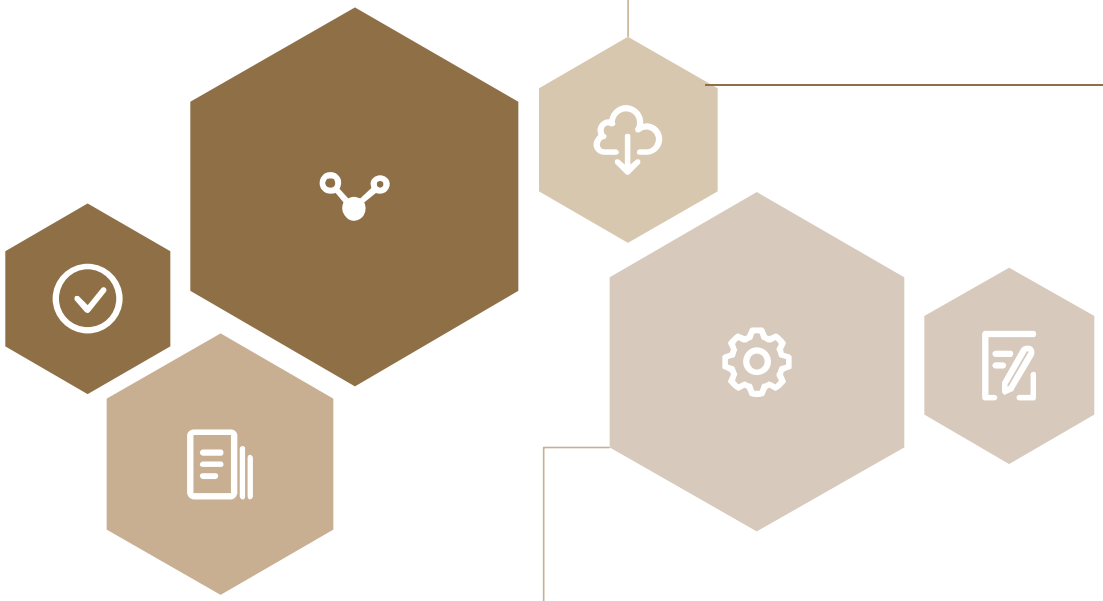
# A Survey of Recent Trends in Testing Concurrent Software Systems

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING , 2018

reporter：曾斯炎 S1810W0760

# 目录
# CONTENTS

# 背景与意义



并发系统难于测试

- 并发故障多取决于各个执行流动作的交错
- 并发故障具有不可再现性

- 过去十五年并发系统迅速普及
- 并发系统种类和数量爆炸式增长

- 伴随着并发系统的增长，并发系统测试技术在近十年也发展迅猛
- 目前缺乏对并发系统测试技术的全面分类、分析和比较。

# 背景与意义——并发系统

包含许多可以“同时”执行并且相互作用的执行流

- 在重叠的时间帧内执行的流 → 比如在多处理器并行和多节点分布式架构上执行的并发程序
- 在非重叠的时间帧内执行的流 → 比如在单核架构上执行的并发程序

# 背景与意义——并发系统分类

- 以执行流间的交互方式为依据对并系统进行分类：
  - 共享内存
  - 消息传递
- 原子性操作建模：
  - 对公共资源的读（read）、写（write）操作。
  - 对接受（receive）和发送（send）两个操作。

# 背景与意义——并发前提

- 前提—— 顺序一致性 Sequential Consistency

保证并发系统中的所有执行流都遵循相同的指令顺序。

- 单个执行流视角
- 整个程序视角

| Execution1 | Execution2 |
| --- | --- |
| x=1 ; | y = 1; |
| r1 = y; | r2 = x; |

| case1 | case2 | case3 |
| --- | --- | --- |
| x = 1; | y = 1; | x = 1; |
| r1 = y; | r2 = x; | y = 1; |
| y = 1; | x = 1; | r1 = y; |
| r2 = x; | r1 = y; | r2 = x; |
| result :<br>r1==0<br>r2 == 1 | result :<br>r1 == 1<br>r2 == 0 | result :<br>r1 == 1<br>r2 == 1 |

# 背景与意义——并发系统测试方法

- 并发系统测试难点：
  - 并发故障通常由执行流意外交错产生，可能非常难以重现

- 并发系统测试方法主要目标：
  - 生成测用例
  - 选择交错的执行流
  - 生成测试结果

# 调查方法与研究趋势

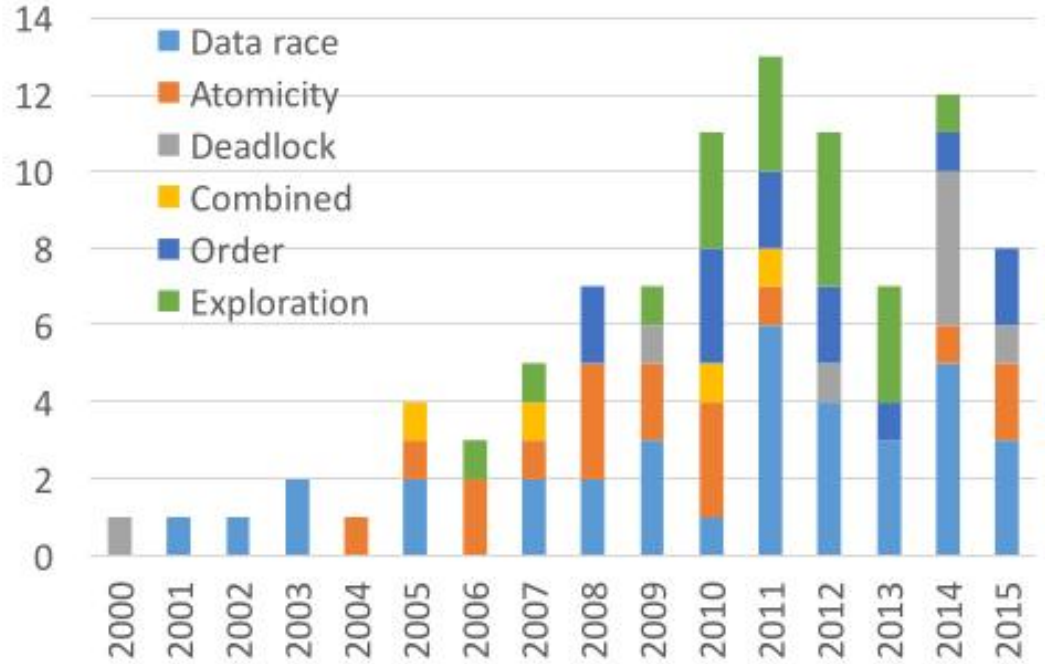
- 搜索了主要的科学出版商的在线资料库
  - IEEE Explore
  - ACM Digital Library
  - Springer Online Library
  - Elsevier Online Library
- 搜索了一些流行的在线搜索引擎
  - Google Scholar
  - Microsoft Academic Search

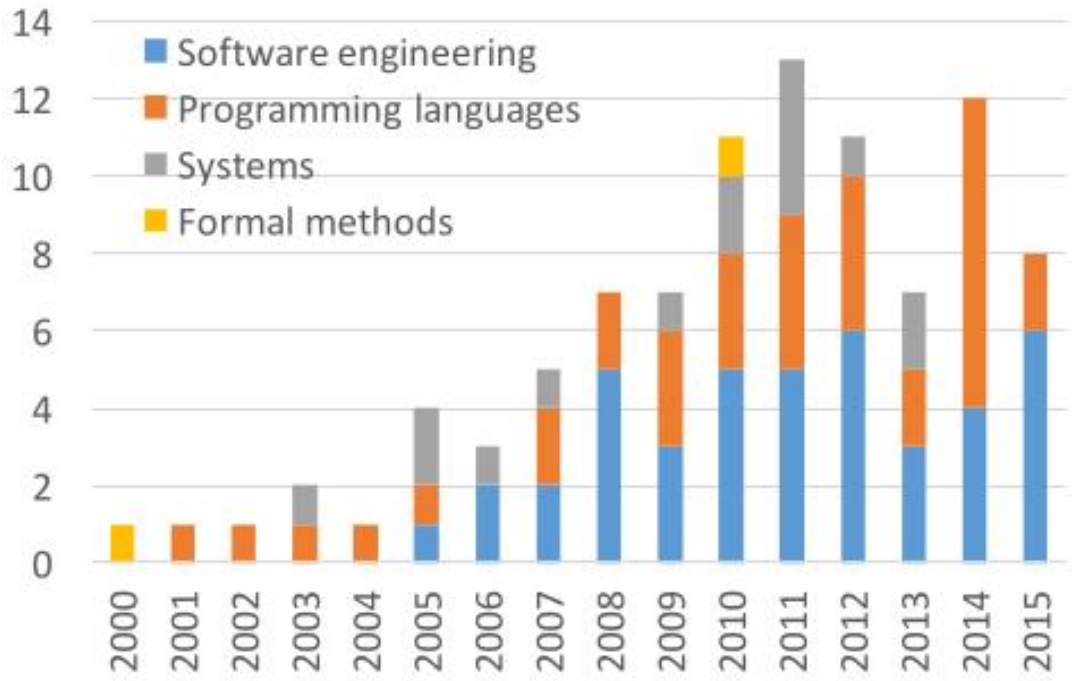
✓ “测试+并发”　✓ “测试+多线程”　✓ “测试+并行”　✓ “测试+分布式”

# 调查方法与研究趋势

- 另外，手动分析与过滤掉一些讨论范围之外（纯粹的硬件测试）的文章

- 选择了过去15年发表的94篇论文，并确定了其侧重的并发属性：

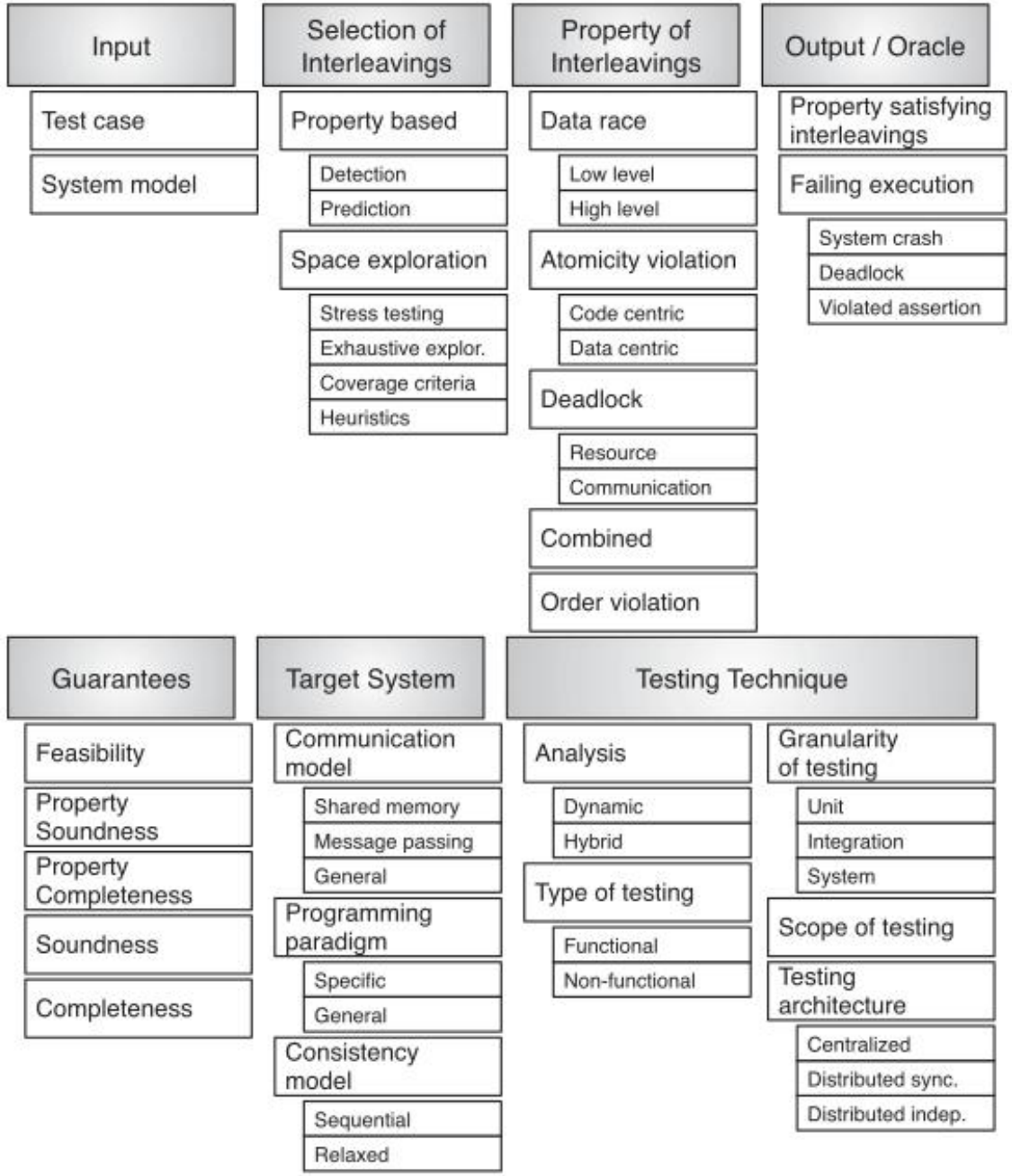


# 分类模型

- 根据七个指标来将并发系统测试技术进行分类。

- 输入
- 输出
- 执行流交错方案的选择
- 属性交错
- 保证性
- 目标系统
- 测试技术架构种类



# 分类模型

## ➢ 输入

- 在执行测试之前需要预准备一组测试用例
- 将测试用例的生成工作包含到测试工作中，即测试技术会自动生成测试用例



## ➢ 输出

- 直接输出对感兴趣属性（比如死锁，数据竞争等）的执行流的交错组合方案

# 分类模型

## ➢ 选择执行流的交错

选择执行流的交错是许多软件测试方法的主要目标

- 基于属性的技术

  主要利用感兴趣的属性来选择执行流的交错。

- 穷尽探索技术
  - 压力测试技术
  - 穷举技术

# 分类模型

- 经典测试属性——数据竞争、原子性侵犯、死锁

## 数据竞争

来自两个不同执行流的操作试图访问同一个数据项，并且至少一个操作是写操作，且没有对该数据项的访问采取同步机制，那么就可能发生数据竞争。

| executionf_1 | executionf_2 |
| --- | --- |
| begin f_1 | begin f_2 |
| x=’AAAAAAAA | x=’BBBBBBBB’ |
| end f_1 | end f_2 |

# 分类模型

- 原子性侵犯

当假定以原子方式执行的执行流的操作序列与来自其他执行流的操作交错时，就会发生原子性侵犯。

| executionf_1 | executionf_2 |
| --- | --- |
| begin f_1 | begin f_2 |
| if (x>0) | x=0 |
| x = x-1 | end f_2 |
| end f_1 | |

# 分类模型

- 死锁

同步机制无限期地阻止某些执行流程继续执行时，就会发生死锁。

| executionf_1 | executionf_2 |
| --- | --- |
| begin f_1 | begin f_2 |
| acquire(L1) | acquire(L2) |
| acquire(L2) | acquire(L1) |
| ... | ... |
| release(L2) | release(L1) |
| release(L1) | release(L2) |
| end f_1 | end f_2 |

中断 →

← 获取L1的锁失败

- 比如上例中，假如执行流f_1首先获取L1的锁，在获取L2的锁之前发生中断。
- 这时执行流f_2获取L2的锁之后，想要去获取L1的锁失败
- 而f_1想要获取L2的锁也失败，因此两个执行流循环等待，造成死锁。

# 分类模型

1

➢ 保证性：这一点性质定义了并发系统测试技术检测并发故障的准确性的标准。

- ✓ 可靠性soundness
- ✓ 完备性completeness

2

➢ 目标系统：

确定了测试技术测试的并发系统类型，如基于共享内存的或基于消息传递的。

3

➢ 测试技术的架构类型：

主要区分了测试技术采用的测试架构的类型。

# 详细调查

- 数据竞争方面：
  - lockset分析
  - happen-before分析

- Savage等人在90年代后期首次提出了用于检测数据竞争的lockset分析方法：
  - 解决了当时静态数据竞争分析的保守局限性
  - 大大提高了数据竞争检测的效率

# 详细调查

- 为了提高lockset分析的精度：

  - 2005，PPoPP —— Shacham等人

    - 将动态的lockset分析与模型检验相结合，从而更好的检测一般共享内存程序中的数据竞争。
    - 通过lockset分析来捕获锁的约束，并执行模型检查器识别的执行流的交错以暴露程序在并发过程中的故障。
    - 因为lockset分析只需要存储关于执行流程所持有的锁的信息，因此，基于lockset分析的技术不具有属性可靠性。

# 详细调查

- 为了进一步提高lockset分析的效率：

  - 2011，ICSE —— Racez

    - 通过采样减少了lockset分析的开销，从而提高了lockset分析的整体性能。
    - Racez采样的部分主要是那些需要进行同步操作和内存访问操作的执行流的集合。
    - Racez通过仅检测同步操作并将检测到的内存访问来进一步减少开销。
    - Racez不具有属性可靠性，同时它也不是属性完备的

# 详细调查

- 将lockset分析延伸到新的编程范式：

  - 2011，PPoPP —— ACCORD

    - ACCORD以基于数组的并发程序中的数据竞争为目标。
    - ACCORD中提出的模型将指定每个执行流读取和写入内存的位置。
    - ACCORD是利用开发人员的注释来提高测试结果准确性的少数方法之一。
    - ACCORD不具有属性完整性，也不具有属性可靠性。

# 详细调查

- 为了减少happen-before分析的开销:

  - 2009，PLDI—— LiteRace

    - LiteRace方法与之前介绍的Racez方法类似。
    - LiteRace根据频繁访问的代码元素（热区）具有较少的参与数据竞争这一前提。
    - LiteRace具有属性完备性，但不具有属性的可靠性。

# 详细调查

- 原子性侵犯方面：

  - 以代码为中心的测试技术
  - 以数据为中心的测试技术

- 检测执行流之间的交错是否违反原子性，就是判断其执行的结果是否等同于任何串行执行的结果。
- 以代码为中心的方法主要通过“搜索不必要的非可串行化的内存访问”来减少验证交错原子性的问题。

# 详细调查

- 以代码为中心的原子性侵犯检测技术:

  - 2014，PLDI——  DoubleChecker

    - DoubleChecker结合精确和不精确的分析来降低检测原子性侵犯的成本
    - DoubleChecker不精确的分析是通过跟踪执行流之间的所有可能的数据依赖性。
    - DoubleChecker既不具有属性完整性也不具有属性可靠性，因为在分析之前可能会错过某些同步约束。

# 详细调查

- 以数据为中心的原子性侵犯检测技术:

  - 2015，ICSE——  ReConTest

    - ReConTest在测试中选择一组执行流交错的集合，形成以优化并发程序为目标的回归测试模型
    - ReConTest会先更改程序，识别可能受程序更改影响的执行流的交错方案。
    - ReConTest执行程序的旧版本和新版本以捕获相关的内存访问的信息。

# 结论与展望

1 并发系统的大多数测试技术都以相关的执行流的交织的选择为目标，而很少有技术专注于测试用例的生成。

2 绝大多数测试方法都以共享内存系统为目标。未来测试消息传递系统仍然值得进行更深入的研究。

3 新的编程范式将测试问题从低级别内存访问冲突转移到高级别的一致性违规，并制造了利用现代编程范式语义来设计测试方法的机会。

# 谢谢您的观看