# 高效的快照隔离检测算法与工具 | VLDB 2023入选论文解读

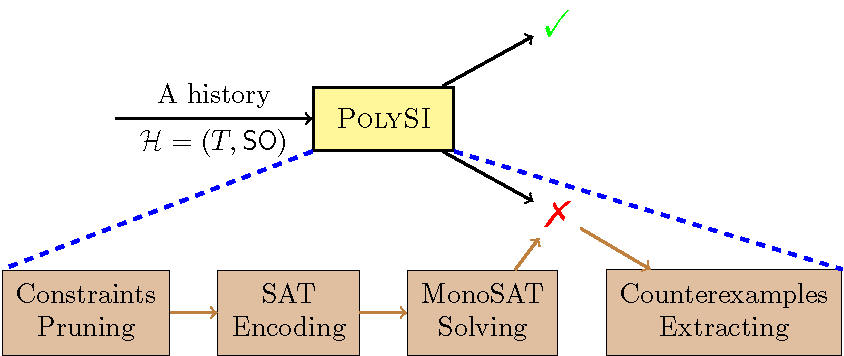
VLDB（International Conference on Very Large Data Bases）是数据库领域顶级国际会议之一。腾讯xxx团队的最新研究成果入选VLDB 2023 Research Full Paper（研究类长文），入选论文题目为“Efficient Black-box Checking of Snapshot Isolation in Databases”。

在数据库事务中，快照隔离（Snapshot Isolation；SI）是一种已被广泛使用的弱隔离级别，它既避免了可串行化带来的性能损失，又能防止多种不希望出现的数据异常。然而，近期的研究指出，一些声称提供快照隔离级别保证的数据库会产生违反快照隔离的数据异常。在本工作中，我们**设计并实现了快照隔离检测器PolySI。PolySI 能够高效地判定给定数据库的执行历史是否满足快照隔离，并在检测到数据异常时提供易于理解的反例**。PolySI的**性能优于目前已知的最好的黑盒快照隔离检查器**，并且可以扩展到包含百万级别事务数量的大规模数据库执行历史上。

## 工作原理

### 总体结构

我们首先提出了基于Polygraph结构的快照隔离刻画定理，这也是PolySI的理论基础，保证了PolySI的正确性与完备性。基于上述刻画定理，PolySI将快照隔离检测问题转化为Polygraph结构上的约束求解问题，并将其编码为命题逻辑公式，利用MonoSAT求解器进行求解。为了提高性能，PolySI采用了一种紧凑型的约束编码方案，并利用领域知识在编码前对约束进行剪枝。PolySI的结构图如下所示。



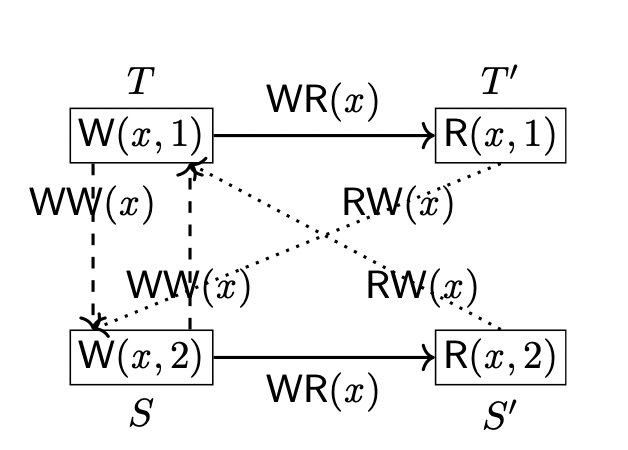
### 基于Polygraph结构的SI刻画定理

在执行历史中，事务之间存在四种依赖关系。

* SO（会话）序：在同一个会话（session）上，所有的事务串行执行。
* WR（读写）序：假设事务B读取了事务A 写入的某个值，则存在从A到B的WR依赖关系。
* WW（写写）序：假设在某个数据项上，事务B覆盖了事务A的写操作，则存在从A到B的WW依赖关系。
* RW（写读）序：假设在某个数据项上，事务B覆盖了事务A的写操作，而事务C读到了事务A写入的值，则存在从C到B的RW依赖关系。

WW序无法直接从执行历史中确定，因此需要尝试不同的WW可能性，这也是测试复杂度的主要来源。以事务为节点、四种依赖关系为有向边，可以构造一个有向图，称之为“依赖图”。2018年，Cerone与Gotsman提出了基于依赖图的SI刻画定理：一个执行历史满足快照隔离，当且仅当存在它对应的某个依赖图，该依赖图是无环的或者它的每个环中都包含至少两条连续的RW边。

为了更紧凑地刻画一个执行历史所对应的所有可能的依赖图，我们引入了Polygraph的概念。对于一个执行历史，WW序是不确定的。比如，事务T和T’写了同一个数据项*x*，但是谁先谁后并不确定。我们用Polygraph建模这种WW不确定性以及由此带来的RW序关系。例如，在下图中，T->T’, S->S’是两个确定的WR依赖关系，而约束 ⟨either = {(𝑇, 𝑆, WW), (𝑇’, 𝑆, RW)}, or = {(𝑆, 𝑇, WW), (𝑆’, 𝑇, RW)}⟩ 则刻画了T与S之间两种可能的WW依赖关系。



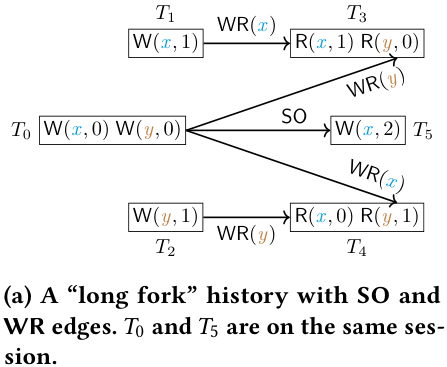
### 约束剪枝与约束编码

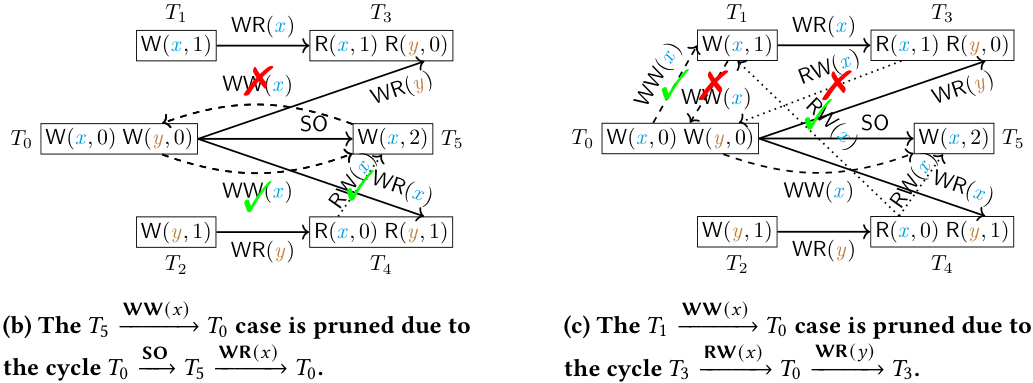
由于一个执行历史所对应的可能的依赖图数量庞大，我们选择使用MonoSAT求解其中的WW约束。基于MonoSAT，PolySI测试算法如下：

* 根据输入的执行历史，构建Polygraph。
* 对Polygraph进行约束剪枝。
* 利用上述的SI刻画定理（与额外的边转换规则），对Polygraph进行SAT编码。
* 使用MonoSAT判断SAT编码表示的有向图是否是无环的。如果是，则该执行历史满足SI；否则，MonoSAT返回一个最小的反例。

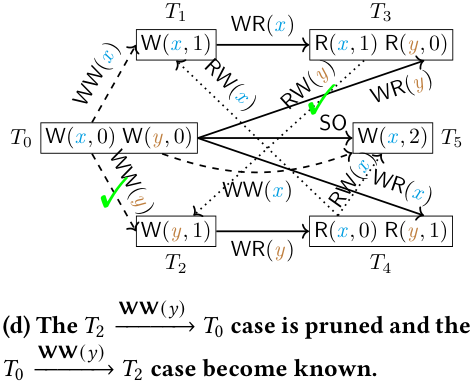
其中，第二步的剪枝策略对于提高PolySI的效率至关重要。剪枝的基本思想是检查Polygraph中的每个约束，如果某个约束的其中一种可能性必定不可能发生（会导致违反SI的环结构），则可以删除该约束，直接选择它的另一种可能性；如果两种可能性都必定不可能发生，则说明对应的执行历史不满足SI。

下面我们通过一个具体的例子介绍PolySI使用的剪枝策略与编码方案。图(a) 展示了一个包含6个事务的执行历史，并给出了SO与WR边。

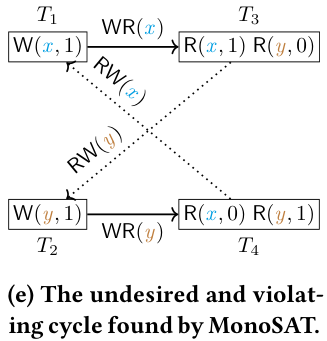




图(b) 中的约束 {T0 ->[WW] T5 / T5 ->[WW] T0} 表示了T0与T5之间两种可能的WW序。由于从T0到T5有一条路径，T5 ->[WW] T0会导致一个环，因此该约束可确定为从T0到T5的WW顺序。图(c) 考虑T0与T1之间两种可能的WW序，其中从T1到T0的WW顺序会造成从T3到T0再到T3的环，因此该约束也可被删除，并确定为T0到T1的WW顺序。



同理，如图(d) 所示，T0与T2之间的约束也可确定为从T0到T2的WW顺序。在约束剪枝步骤之后，T1与T5之间的WW序仍不确定。如前所述，PolySI使用Polygraph图中的约束刻画这种不确定性：⟨either = {(𝑇1, 𝑇5, WW), (𝑇3, 𝑇5, RW)}, or = {(𝑇5, 𝑇1, WW)}⟩。该约束将被编码为符合MonoSAT格式要求的命题逻辑公式。

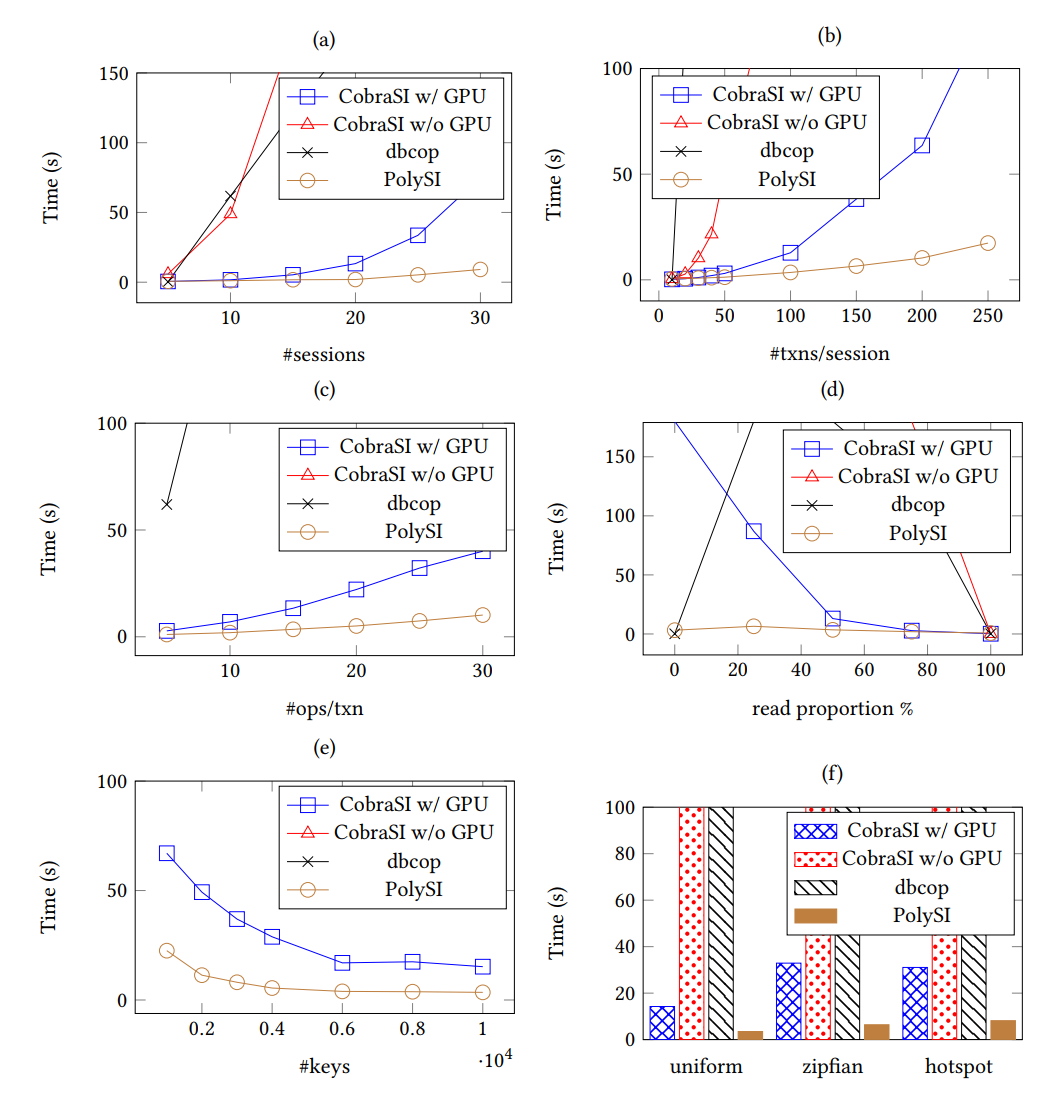


最后，图(e) 展示了违反SI要求的一个环结构，这也是MonoSAT的输出结果。

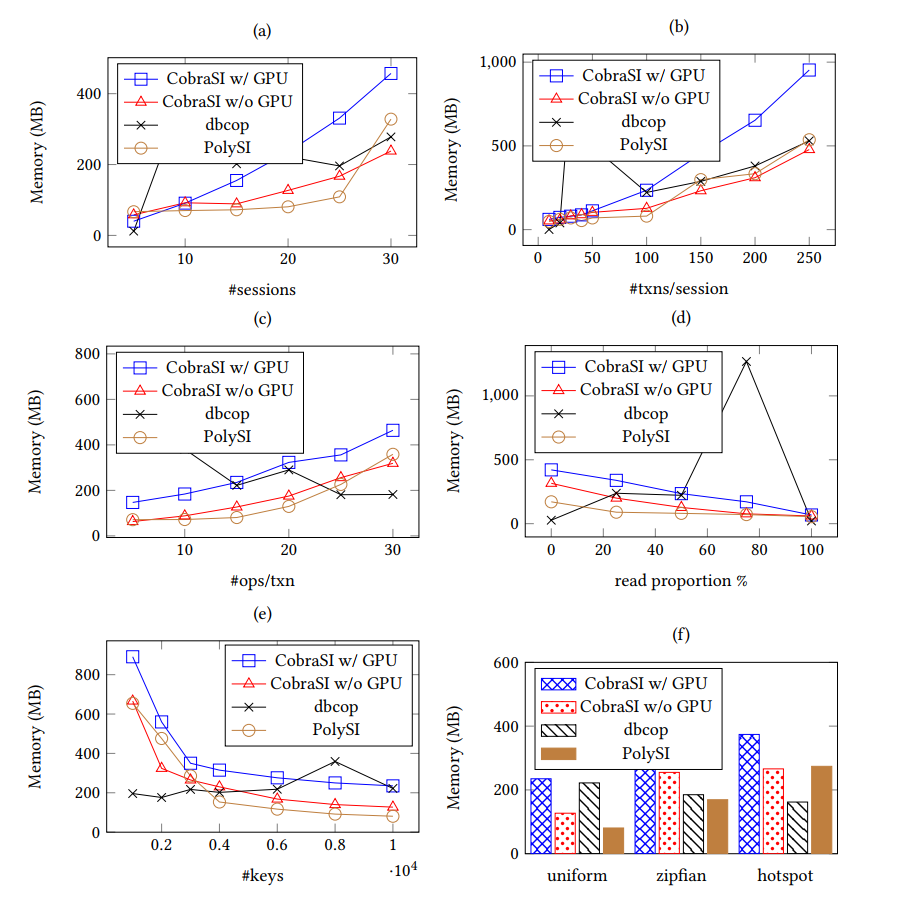
## 实验评估

### 性能实验

为了说明PolySI的高效性，我们从多个方面对PolySI和同类工具的效率进行了对比实验。这里对比的4个工具，分别为：PolySI、dbcop、不带GPU优化的 Cobra、带GPU优化的 Cobra。为了对比的公平性，使用 Cobra 进行测试时，我们先将测试SI的输入归约为测试SER 的输入，再使用 Cobra 进行测试，我们将该工具称为 CobraSI。

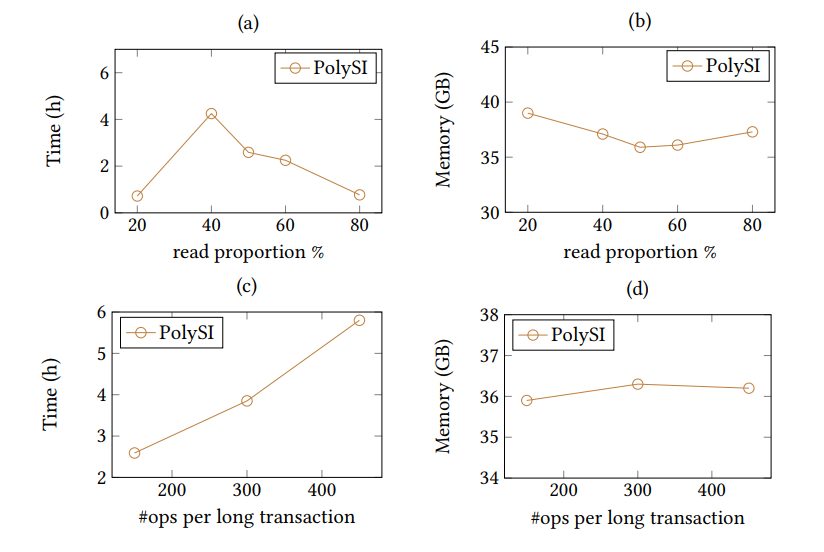


图(a)、图(b)、图(c)说明，在测试规模变大时，各工具的求解时间呈指数规律增长，但是PolySI 求解时间随问题规模的增长是最缓慢的。图(d) 展示的是读写操作的比例对求解时间的影响。该组实验表明，PolySI 是效率最高的，并且几乎不受读写操作比例的影响。图(e) 考察在规模不变的情况下增加键 (key) 的数量对求解时间的影响。实验表明，在这种情况下，PolySI的性能依旧是最好的。图(f) 则表明，在 Uniform, Zipfian 和 Hotspot 三类常见操作访问模式上，PolySI 的性能都是最优的。



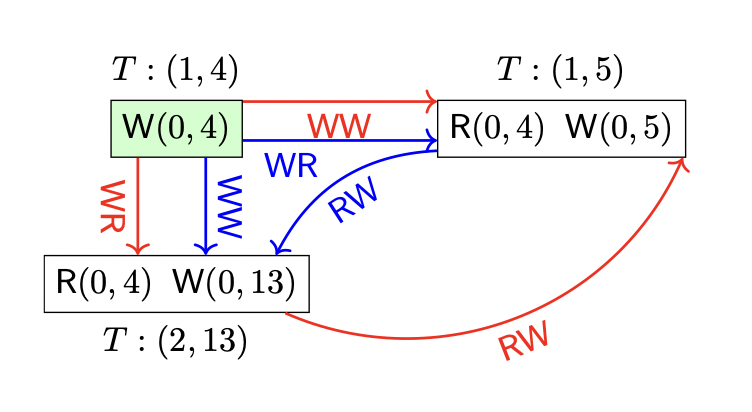
上图表明，在内存开销方面，PolySI也是优于同类测试工具的。

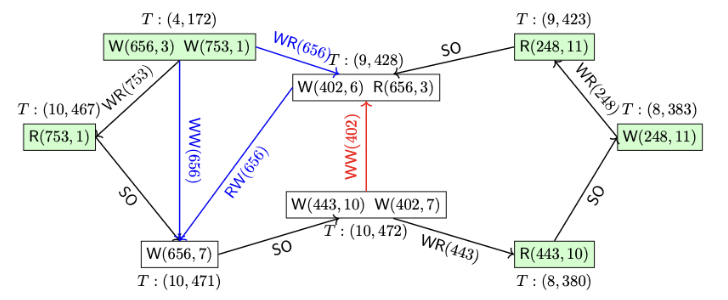
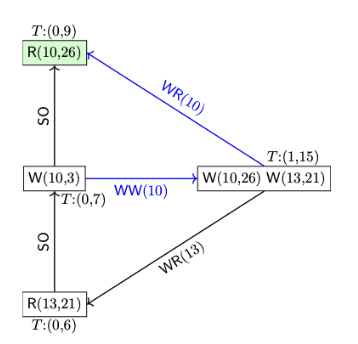
最后，我们通过实验验证了PolySI在超大规模负载下的优秀性能。下图展示了在百万级事务、十亿级键数量负载下，PolySI可以在6个小时内完成检测，而其它工具都无法处理这种规模的负载。



### 有效性实验

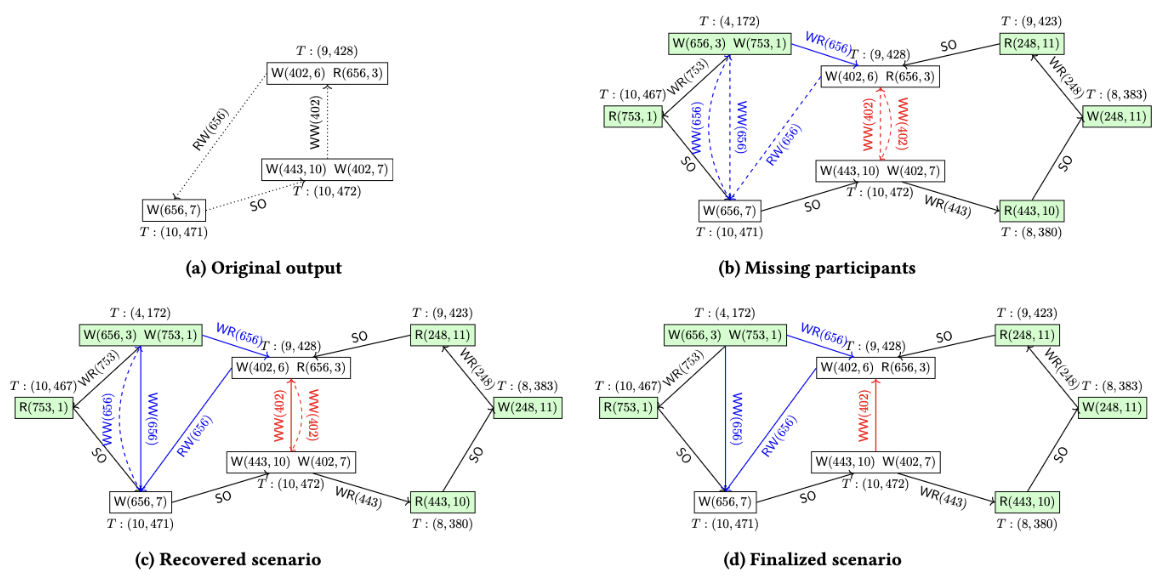
实验结果表明，PolySI可以成功复现已知的快照隔离数据异常，也发现了三个数据库中未知的快照隔离异常并给出了易于理解的反例。如下图，第一个bug是MariaDB中存在的“更新丢失”（lost update）数据异常，两个并发事务T:(1,5) 和 T:(2,13) 都成功更新了key ‘0’的值，导致其中一个更新被覆盖。该bug已经得到了MariaDB官方的确认。第二个bug是DGraph中存在的“因果悖逆”（causal violation）数据异常，三个事务T:(4,172) -> T:(10,471) -> T:(9,428) 之间存在因果序，但是T:(9,428) 却读到了T:(4,172) 而不是T:(10,471) 对同一个键的修改。该bug帮助了DGraph 官方开发新版本，并在最新版本中修复了该bug。第三个bug 是YugabyteDB中存在的“因果悖逆”数据异常，三个事务T:(1,15)->T:(0,7)->T:(0,9) 之间存在着因果序，但是 T:(0,9) 读到了 T:(1,15) 的内容，却没有读到 T:(0,7) 对同一个键的修改内容。



### 反例抽取

当输入的执行历史不满足SI时，MonoSAT会输出一个环。这个环是bug的一部分，但是通常情况下，仅凭这个环很难理解bug的全貌。例如，下图(a) 展示的是PolySI给出的DGraph中bug的原始输出，它缺失了很多关键信息，比如为什么T(10,472) 发生在T(9,428) 之前？



对于图(a) 中的 T(9,428) ->[RW] T(656,7) 边，我们可以恢复它所在的约束，即图(b) 中的蓝色虚线部分。对于蓝色虚线部分的另一个方向，也就是T(10,471) ->[WW] T(4,172)，我们可以确定它所在的最小环，也就是图(b)中左边的三个事务构成的环。同样地，图(b) 中的红色部分也是一个约束，我们恢复了它的另一个方向所在的最小环，即图中右边的五个事务构成的环。在这个恢复环的过程中，我们确保找到的是最小环，这样就可以得到最小反例。

对于图中的RW边，它是通过同一个约束中对应的WW边和某条WR边推导得出的，为了理解其成因，我们还需要还原其对应的WR边。在图(c) 中，我们恢复图中T(9,428) ->[RW] T(656,7) 边对应的WR边（蓝色部分的实线）。在MonoSAT输出的环所在的依赖图中，我们还原出来的约束中的另一个方向的边实际上是不存在的，因此我们需要将它们删除，得到图(d)。

## 总结

本工作提出了一种高效的快照隔离检测算法，并实现了相应的检测工具PolySI。目前，PolySI将MonoSAT当作黑盒使用。我们正在考虑如何将SAT求解器深度整合到算法过程中，设计并实现针对事务一致性的理论与专用求解器，进一步提升检测效率。