# Automated Synthesis of Comprehensive Distributed Consistency Model Litmus Test Suites*

Xue Jiang
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
xxx@smail.nju.edu.cn

Hengfeng Wei*
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
hfwei@nju.edu.cn

Yu Huang
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
yuhuang@nju.edu.cn

. . .

. . .

## ABSTRACT

**PVLDB Reference Format:**
Xue Jiang, Hengfeng Wei*, Yu Huang, . . . , and . . . . Automated Synthesis of Comprehensive Distributed Consistency Model Litmus Test Suites. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

## 1 INTRODUCTION

***Motivations.*** Distributed consistency models are quite tricky to understand. Even worse, in the literature, there are often several variants of a specific consistency model. For example, Jiang et al. presents six variants of causal consistency in [**?** ] for non-transactional databases. Crooks presents a hierarchy of eight variants of snapshot isolation [**?** ] for transactional databases. It is difficult, even for experts,

- **Consistency Checking:** to check whether a given history satisfies some consistency model or not;
- **Test-case Generating:** to come up with histories under some constraints (e.g., on size) that satisfy or refuse some consistency model; and
- **Model Comparing:** to tell the differences between two consistency models by presenting distinguishing histories that satisfy one consistency model but refuse the other.

All the three tasks are concerned about histories, which are easier for human to understand.

programs vs. litmus tests vs. histories vs. abstract executions

TODO: 1) how to control the execution of a given history; 2) random tests vs litmus tests; 3) how to divide histories into equivalent classes (reduce reduction)

***Challenges.***

- How to formally express consistency models?
- generating too many (redundant) histories

***Our Contributions.***

- Alloy∗ model of both non-transactional and transactional consistency models
- Two case studies

## 2 PRELIMINARIES

### 2.1 The Alloy and Alloy* Modelling Languages

"Alloy [**?** ] provides a domain-specific language for defining relational models: models consisting of a set of atoms (or, for our purposes, nodes in a graph), and relations (edges in a graph). The relational approach makes it a natural fit for describing consistency models. Alloy feeds its models into the Kodkod relational model finder, which in turn uses off-the-shelf SAT [] solvers to search for instances of the given model. This flow provides a convenient front end suited for asking questions about memory models and/or about particular litmus tests. The basic Alloy syntax is summarized in Table 3." However, Alloy is based on a first-order relational logic. The for-all and exists-some quantifiers over relations cannot be described. To this end, Alloy* [**?** ] is provided to support higher-order and bounded constraint solver based on the Alloy analyzer.

### 2.2 The (VIS, AR) Specification Framework

*2.2.1 Non-transactional Consistency Models.*

*2.2.2 Transactional Consistency Models.*

## 3 OVERVIEW

## 4 APPROACH

## 5 CASE STUDIES

### 5.1 Causal Consistency Variants

*5.1.1 The Alloy* Model.* We first define the $(vis, ar, V)$ framework using Alloy. At the top of the model, several type signatures ("sigs") are defined. According to Definition **??**, an event $E$ contains an operation op and a return value rval. Additionally, the signature $E$ defines sorelation that originates form itself to other events on the same session, session relation that indicates where $E$ performs on and ve relation that depicts the events in $V(e)$ (Definition **??**). Since we focus on read/write key-value stores in this paper, there are two

*Corresponding author. Hengfeng Wei is also with Software Institute at Nanjing University.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

kinds of operations, i.e., Read and Write. The Read and Write operations reads from and writes to a key of type Key.

Below the sig definitions are the basic constraints ("facts"). For example, so is a total order over all events on the same session. Any valid instance of the model specified in the $(vis, ar, V)$ framework is required to satisfy all of the listed facts to be considered well-formed, regardless of whether it is considered valid according to the model. Then we describe the return value consistency RVal(reg) (Definition **??**) by the predicates "ReadLastVisibleWrite" and "VeIsReasonable".

Based on the $(vis, ar, V)$ framework in Figure **??**, we formalize the recipes for $vis$, $ar$ and $V$ in Table **??** and six causal consistency variants specified in the $(vis, ar, V)$ framework. The predicate for each causal consistency variant specifies the properties of $vis$, $ar$ and $V$ that must hold true for an execution which satisfies the model.

#### 5.1.2 *Consistency Checking.*

#### 5.1.3 *Test-case Generating.*

#### 5.1.4 *Model Comparing.*

### 5.2 Snapshot Isolation Variants

#### 5.2.1 *The Alloy* Model.*

#### 5.2.2 *Consistency Checking.*

#### 5.2.3 *Test-case Generating.*

#### 5.2.4 *Model Comparing.*

## 6 RELATED WORK

***Consistency Checking.*** MemSAT [**?** ]: "Given an axiomatic specification of a memory model and a multi-threaded test program containing assertions, MemSAT outputs a trace of the program in which both the assertions and the memory model axioms are satisfied, if one can be found. If it cannot find a trace, it outputs a minimal subset of the memory model and program constraints that are unsatisfiable."

***Test-case Generating.***

***Model Comparing.*** [**?** ]: "systematically comparing hardware memory models specified using both operational and axiomatic styles. When the models differs, the tool finds a minimal "litmus test" program that demonstrates the difference."

## 7 CONCLUSION