TLA+ and a Concurrent Hashmap Specification

Åsmund Aqissiaq Arild Kløvstad

Universitetet i Bergen

March 17, 2022

Table of Contents

Outline

1. What was the project?
2. The data structure
3. TLA+
4. The specification
5. Results and lessons learned

The Project

- 10 pt optional Bachelor Thesis at UiT
- Spring semester 2020
- Supervised by Håvard D. Johansen
- Specifying and modelchecking a hash table in TLA+

Problem Statement

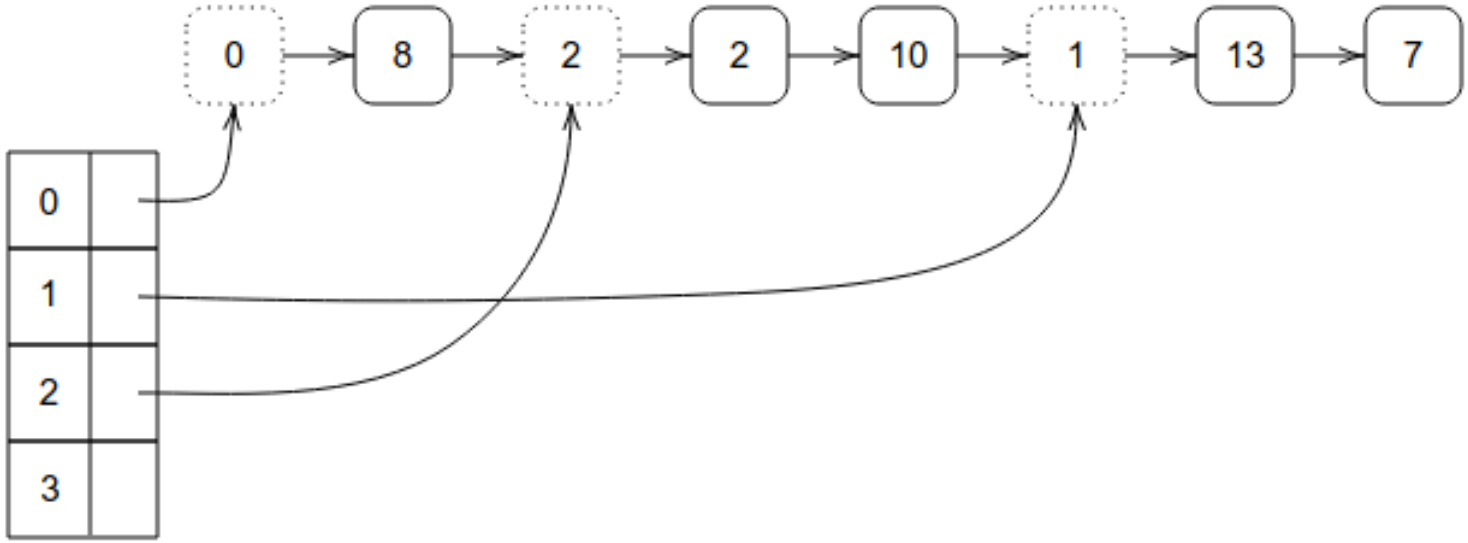
- Formal methods
- Concurrent and distributed systems
- A novel hashmap [1]

Shalev et. al.'s Hash Table can be formally verified for concurrent settings

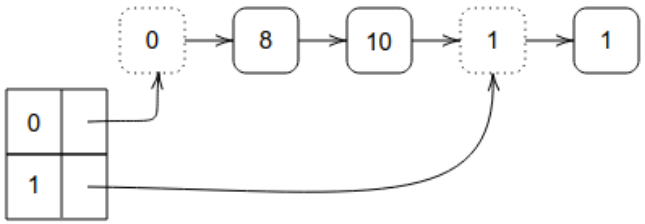
Table of Contents

The Idea

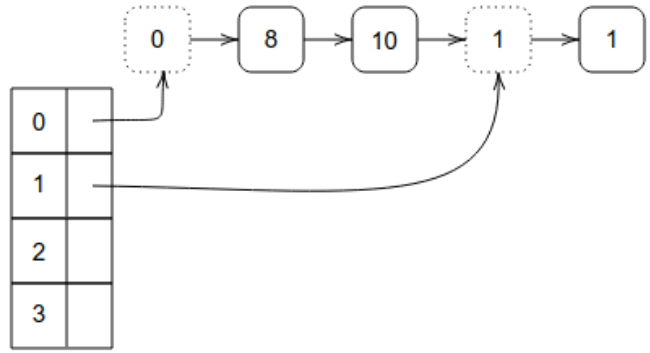
- Resizing is hard in concurrent settings
- Solution: move the buckets instead
- Split-ordering (sort by reverse binary representation)
- Dummy nodes



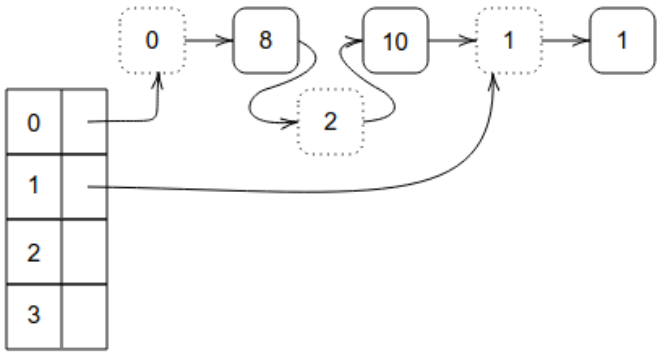
Insertion with Splitting



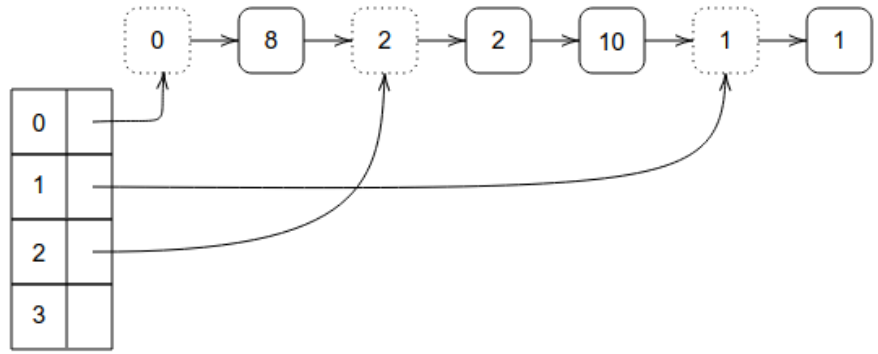
(a) Initial state



(b) Table is expanded



(c) Dummy node for new bucket inserted



(d) Table entry points to bucket node

Table of Contents

The Basics

- **T**emporal **L**ogic of **A**ctions
- Modalities: always ($\Box$) and eventually ($\Diamond$)
- **actions** are logical predicates on **variables**
- The basic specification "Init and always Next":

$$Spec \triangleq Init \wedge \Box[Next]_{\langle iter, revIter \rangle}$$

- Implementation is implication. A implements B exactly when $A \implies B$

Syntax Quirks

- Primed variables

$$Next \triangleq iter' = iter + 1 \land \text{UNCHANGED } revIter$$

- Functions

$$list \triangleq [0..42 \mapsto String]$$

- EXCEPT

$$list' = [list \text{ EXCEPT } ![4] = \text{"}hei\text{"}]$$

Table of Contents

Overview

Four specifications:

1. Generic hash table
2. Non-concurrent split-order implementing 1.
3. Concurrent split-order
4. (Concurrent with operation ID's)

Generic Hash Table

MODULE *hashmap*

This module describes a hashmap to be used for testing with Shalev et al.'s split-ordered list implementation of the data structure

EXTENDS *Integers*

CONSTANTS *NULL*, *PossibleKeys*, *PossibleValues*

VARIABLES *keys*, *map*

Initial state has empty map and no keys

$$HashmapInit \triangleq \land keys = \{\} \land map = [k \in PossibleKeys \mapsto NULL]$$

Insert changes exactly one mapping of the hashmap and adds one key to the set of keys

$$Insert \triangleq \exists\, k \in PossibleKeys : \exists\, v \in PossibleValues : \land keys' = keys \cup \{k\} \land map' = [map \text{ EXCEPT } ![k] = v]$$

Remove sets exactly one mapping to NULL

$$Remove \triangleq \exists\, k \in PossibleKeys : \land keys' = keys \setminus \{k\} \land map' = [map \text{ EXCEPT } ![k] = NULL]$$

Figure: Generic hash table specification

Non-concurrent

$$
\begin{aligned}
&SOInsert \triangleq \ \land\ \exists\, k \in PossibleKeys : \\
&\quad \exists\, v \in PossibleValues : \\
&\qquad BucketInsert(k,\ v)
\end{aligned}
$$

$$
\begin{aligned}
&BucketInsert(k,\ v) \triangleq \\
&\quad \text{Either a bucket needs to be initialized} \\
&\qquad \lor\ \land\ buckets[k\%size] = NULL \\
&\qquad\quad \land\ BucketInit(k\%size) \\
&\qquad\quad \land\ ListInsert(SORegularKey(k),\ v) \\
&\qquad\quad \land\ AuxKeys' = AuxKeys \cup \{k\} \\
&\qquad\quad \text{Or the bucket is already initialized} \\
&\qquad \lor\ \land\ buckets[k\%size] \neq NULL \\
&\qquad\quad \land\ ListInsert(SORegularKey(k),\ v) \\
&\qquad\quad \land\ AuxKeys' = AuxKeys \cup \{k\} \\
&\qquad\quad \land\ \textsc{unchanged}\ \langle buckets \rangle
\end{aligned}
$$

$$
\begin{aligned}
&ListInsert(k,\ v) \triangleq \textsc{if}\ list[k] = NULL \\
&\quad \textsc{then}\ list' = [list\ \textsc{except}\ ![k] = v] \land count' = count + 1 \\
&\quad \textsc{else}\ \ \textsc{unchanged}\ \langle list,\ count \rangle
\end{aligned}
$$

Figure: The insert operation

Concurrent Operations I

$$
\begin{array}{l}
SONext \triangleq \\
\quad \lor \ \land \exists\, k \in PossibleKeys : \\
\qquad\quad \exists\, v \in PossibleValues : \\
\qquad\qquad Insert(k,\ v) \\
\qquad \land BagCardinality(activeOps) < MaxActiveOps \\
\qquad \land \text{UNCHANGED } \langle buckets,\ count,\ list,\ size \rangle \\
\quad \lor \ \land \exists\, k \in PossibleKeys : \\
\qquad\quad Delete(k) \\
\qquad \land BagCardinality(activeOps) < MaxActiveOps \\
\qquad \land \text{UNCHANGED } \langle buckets,\ count,\ list,\ size \rangle \\
\quad \lor Insert1 \\
\quad \lor Insert2 \\
\quad \lor Insert3 \\
\quad \lor Insert4 \\
\quad \lor Delete1 \\
\quad \lor Delete2 \\
\quad \lor Delete3 \\
\quad \lor BucketInit1 \\
\quad \lor BucketInit2 \\
\quad \lor BucketInit3
\end{array}
$$

Next step

Concurrent Operations II

$$
\begin{array}{l}
Insert1 \triangleq \\
\quad \text{Start a } bucket\_init \text{ if necessary} \\
\quad \exists\, op \in BagToSet(activeOps) : \\
\qquad \land\ op.type = \text{“insert”} \\
\qquad \land\ op.step = 1 \\
\qquad \land\ \text{IF } buckets[op.key\%size] = NULL \\
\qquad\quad \text{Nextstep and “begin } bucket\_init\text{” both modify the state of } activeOps \text{ and need to be combined} \\
\qquad\quad \text{THEN } activeOps' = (activeOps \ominus SetToBag(\{op\})) \\
\qquad\qquad\qquad\qquad \oplus SetToBag(\{[op \text{ EXCEPT } ![\text{“step”}] = op.step + 1]\}) \\
\qquad\qquad\qquad\qquad \oplus SetToBag(\{[type \mapsto \text{“bucket\_init”},\ step \mapsto 1,\ bucket \mapsto op.key\%size]\}) \\
\qquad\quad \text{ELSE } NextStep(op) \\
\qquad \land\ \text{UNCHANGED } \langle list,\ buckets,\ size,\ count \rangle
\end{array}
$$

$$
\begin{array}{l}
Insert2 \triangleq \\
\quad \text{If key is already in list, end operation. Else insert in list} \\
\quad \exists\ op \in BagToSet(activeOps) : \\
\qquad \land\ op.type = \text{“insert”} \\
\qquad \land\ op.step = 2 \\
\qquad \land\ \ \text{IF } list[SORegularKey(op.key)] = NULL \\
\qquad\qquad\quad \text{THEN } list' = [list \text{ EXCEPT } ![SORegularKey(op.key)] = op.value] \land NextStep(op) \\
\qquad\qquad\quad \text{ELSE } End(op) \land \text{UNCHANGED } list \\
\qquad \land\ \text{UNCHANGED } \langle buckets,\ size,\ count \rangle
\end{array}
$$

Concurrent Operations III

$$
\begin{aligned}
&Insert3 \triangleq \\
&\quad \text{Increment count} \\
&\quad \exists\ op \in BagToSet(activeOps) : \\
&\qquad \land\ op.type = \text{“insert”} \\
&\qquad \land\ op.step = 3 \\
&\qquad \land\ count' = count + 1 \\
&\qquad \land\ NextStep(op) \\
&\qquad \land\ \text{UNCHANGED}\ \langle list,\ buckets,\ size \rangle
\end{aligned}
$$

$$
\begin{aligned}
&Insert4 \triangleq \\
&\quad \text{Change size if loadfactor is exceeded and end insert} \\
&\quad \exists\ op \in BagToSet(activeOps) : \\
&\quad \land\ op.type = \text{“insert”} \\
&\quad \land\ op.step = 4 \\
&\quad \land\ \text{IF}\ count \div size > LoadFactor\ \text{THEN}\ size' = Min(2 * size,\ MaxSize)\ \text{ELSE}\ \text{UNCHANGED}\ size \\
&\quad \land\ End(op) \\
&\quad \land\ \text{UNCHANGED}\ \langle list,\ buckets,\ count \rangle
\end{aligned}
$$

Testing the Non-concurrent Spec

$$SOSpec \triangleq SOInit \wedge \Box[SONext]_{\langle keys, AuxKeys, list, buckets, size, count, map \rangle}$$

INSTANCE hashmap
THEOREM SOSpec $\Longrightarrow$ *HashmapSpec*

Testing the Specifications I

Claimed invariants:

1. If a bucket is initialized, then it points to a dummy node in the list

$$
\begin{array}{l}
BucketsInitialized \triangleq \\
\quad \forall\, i \in 0\,..\,(size-1) : \\
\qquad \lor\ buckets[i] = NULL \\
\qquad \lor\ buckets[i] = SODummyKey(i) \land list[SODummyKey(i)] = i
\end{array}
$$

2. If k is not in the map, then `delete(k)` fails. Otherwise k is removed from the map

Testing the Specifications II

An insert with key in map will not reach step 3

$$
\begin{array}{l}
InsertFails \triangleq \\
\quad \forall\, op \in OperationStates : \\
\quad\quad \text{IF } op.type = \text{“insert”} \\
\quad\quad\ \text{THEN} \\
\quad\quad\quad \wedge\, op \in BagToSet(activeOps) \\
\quad\quad\quad \wedge\, op.step = 1 \\
\quad\quad\quad\ \wedge\, list[SORegularKey(op.key)] \neq NULL \\
\quad\quad\ \Rightarrow \Box\Diamond(\neg([op \text{ EXCEPT } ![\text{“step”}] = 3] \in BagToSet(activeOps))) \\
\quad\quad\ \text{ELSE TRUE}
\end{array}
$$

3. If k is in the map, then `insert(k)` fails. Otherwise k is added to the map

Table of Contents

Non-concurrent

Keys	Values	Diameter	Distinct States	Time (hh:mm:ss)
2	4	10	2,523	00:00:01
2	8	10	23,763	00:00:08
2	16	10	279,075	00:01:02
4	2	17	39,827	00:00:09
4	4	17	1,790,067	00:03:44
4	8	17	147,266,723	07:14:45

Table: Model checking results for SplitOrder

Concurrent

Invariant	Keys	Values	Conc. Ops	Diameter	Distinct States	Time (mm:ss)
InsertSucceeds	2	2	2	38	10,083	00:02
	2	4	2	38	66,901	00:13
	2	4	3	45	1,351,453	01:50
	4	2	2	62	1,627,390	02:15
DeleteSucceeds	2	2	2	38	10,083	00:01
	2	4	2	38	66,901	00:07
	2	4	3	45	1,351,453	00:31
	4	2	2	62	1,627,390	00:45
BucketsInitialized	2	2	2	38	10,083	00:02
	2	4	2	38	66,901	00:16
	2	4	8	38	624,645	00:16
	2	4	16	38	7,371,157	01:55
	4	2	2	62	1,627,390	00:22
	4	3	2	62	8,368,282	01:39
	4	4	2	62	29,973,646	06:10
	4	5	2	62	85,419,916	18:41
	4	2	3	75	61,353,460	13:59

Failures

Invariant	Keys	Values	Conc. Ops	Diameter	Distinct States	Time (hh:mm:ss)	Error
Non-Concurrent	4	16		17	84,587,043	02:54:11	3
InsertSucceeds	4	3	2	27	1,426,888	00:33:30	2
DeleteSucceds	4	3	2	39	6,067,795	00:30:30	1
BucketsInitialized	4	2	4	57	120,175,837	00:26:49	4

Table: Failed model checks

Code	Error
1	GC Overhead limit exceeded
2	Out of heap space
3	No space left on device
4	Unknown

Table: Error Codes

Table of Contents

Designing Specifications

1. Decide on purpose early and precisely
2. Choose step size early and deliberately
3. Test assumed invariants and try to break them

The Use of Model Checking

- Poor scaling
- Computationally expensive
- Useful
  1. as a way to produce minimal examples of errors, and
  2. as a development tool for algorithms/protocols
- Increased confidence in algorithms, though arguably no more than a tested implementation

References

Tables and figures from my bachelor thesis [2]

Ori Shalev and Nir Shavit. "Split-Ordered Lists: Lock-Free Extensible Hash Tables". In: *J. ACM* 53.3 (2006), pp. 379–405. ISSN: 0004-5411. DOI: 10.1145/1147954.1147958. URL: https://doi.org/10.1145/1147954.1147958.

Åsmund Aqissiaq Arild Kløvstad. "Formal Verification of a Lock-free Split-order Hashmap". In: (2020).