Efficient Black-box Checking of Snapshot Isolation in Databases

Kaile Huang, Si Liu, Zhenge Chen,

Hengfeng Wei, David Basin, Haixiang Li, Anqun Pan

hfwei@nju.edu.cn

September 1, 2023

Transaction and Isolation Level

A transaction is a *group* of operations that are executed atomically.

$x_1 \leftarrow \mathsf{R}(acct_1)$
$x_2 \leftarrow \mathsf{R}(acct_2)$
if $x_1 + x_2 > 100$
  $x_1 \leftarrow x_1 - 100$
  $\mathsf{W}(acct_1, x_1)$

$x_1 \leftarrow \mathsf{R}(acct_1)$
$x_2 \leftarrow \mathsf{R}(acct_2)$
if $x_1 + x_2 > 100$
  $x_2 \leftarrow x_2 - 100$
  $\mathsf{W}(acct_2, x_2)$

$x_1 \leftarrow \mathsf{R}(acct_1)$
$x_2 \leftarrow \mathsf{R}(acct_2)$

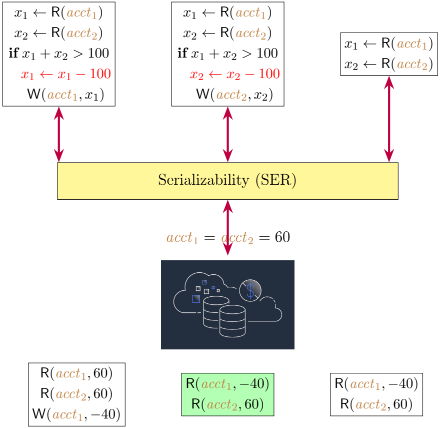
Isolation Level (aka Transactional Consistency Model)

$acct_1 = acct_2 = 60$

The isolation levels specify how concurrent transactions are isolated from each other.

Serializability (SER)

All transactions appear to be executed in some total order.



However, implementing serializability is too expensive.

Snapshot Isolation (SI)

$x_1 \leftarrow \mathsf{R}(acct_1)$
$x_2 \leftarrow \mathsf{R}(acct_2)$
if $x_1 + x_2 > 100$
$x_1 \leftarrow x_1 - 100$
$\mathsf{W}(acct_1, x_1)$

$x_1 \leftarrow \mathsf{R}(acct_1)$
$x_2 \leftarrow \mathsf{R}(acct_2)$
if $x_1 + x_2 > 100$
$x_2 \leftarrow x_2 - 100$
$\mathsf{W}(acct_2, x_2)$

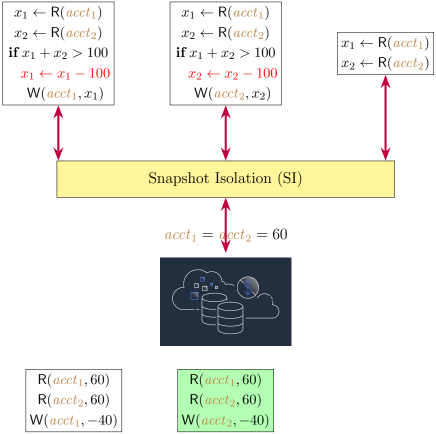
$x_1 \leftarrow \mathsf{R}(acct_1)$
$x_2 \leftarrow \mathsf{R}(acct_2)$

Snapshot Isolation (SI)

$acct_1 = acct_2 = 60$

Snapshot Read: Each transaction reads data from a *snapshot* as of the time the transaction started.

Snapshot Isolation (SI)



Snapshot Read: Each transaction reads data from a *snapshot* as of the time the transaction started.

Snapshot Isolation (SI)

$x_1 \leftarrow \mathsf{R}(acct_1)$
$x_2 \leftarrow \mathsf{R}(acct_2)$
if $x_1 + x_2 > 100$
$x_1 \leftarrow x_1 - 100$
$\mathsf{W}(acct_1, x_1)$

$x_1 \leftarrow \mathsf{R}(acct_1)$
$x_2 \leftarrow \mathsf{R}(acct_2)$
if $x_1 + x_2 > 100$
$x_2 \leftarrow x_2 - 100$
$\mathsf{W}(acct_2, x_2)$

$x_1 \leftarrow \mathsf{R}(acct_1)$
$x_2 \leftarrow \mathsf{R}(acct_2)$

Snapshot Isolation (SI)

$acct_1 = acct_2 = 60$

Write Skew ✓

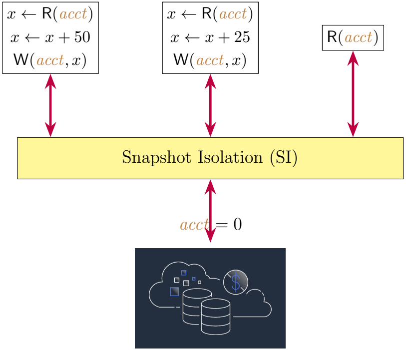
$\mathsf{R}(acct_1, 60)$
$\mathsf{R}(acct_2, 60)$
$\mathsf{W}(acct_1, -40)$

$\mathsf{R}(acct_1, 60)$
$\mathsf{R}(acct_2, 60)$
$\mathsf{W}(acct_2, -40)$

$\mathsf{R}(acct_1, -40)$
$\mathsf{R}(acct_2, -40)$

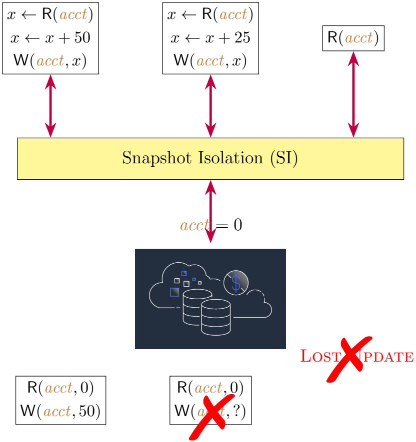
Snapshot Read: Each transaction reads data from a *snapshot* as of the time the transaction started.

Snapshot Isolation (SI)



Snapshot Write: Concurrent transactions *cannot* write to the same key. One of them must be aborted.

Snapshot Isolation (SI)

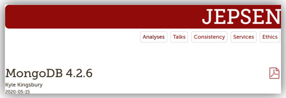


Snapshot Write: Concurrent transactions *cannot* write to the same key. One of them must be aborted.

Database systems and Snapshot Isolation

Many database systems implement snapshot isolation.

Database systems may fail to provide snapshot isolation correctly.

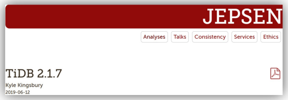

JEPSEN

Analyses | Talks | Consistency | Services | Ethics

MongoDB 4.2.6

Kyle Kingsbury

2020-05-15


JEPSEN

Analyses | Talks | Consistency | Services | Ethics

TiDB 2.1.7

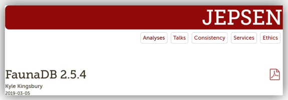
Kyle Kingsbury

2019-06-12


Elle: Inferring Isolation Anomalies from Experimental Observations

Kyle Kingsbury
Jepsen
aphyr@jepsen.io

Peter Alvaro
UC Santa Cruz
palvaro@ucsc.edu

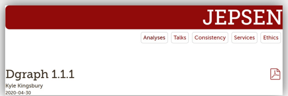

JEPSEN

Analyses | Talks | Consistency | Services | Ethics

FaunaDB 2.5.4

Kyle Kingsbury

2019-03-05


JEPSEN

Analyses | Talks | Consistency | Services | Ethics

Dgraph 1.1.1

Kyle Kingsbury

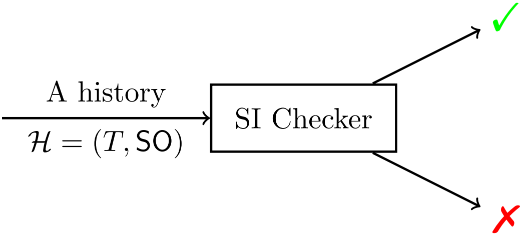
2020-04-30

The SI Checking Problem

Definition (The SI Checking Problem)

The SI checking problem is the decision problem of determing whether a *history* $\mathcal{H} = (T, \mathsf{SO})$[a] of a database system satisfies SI?

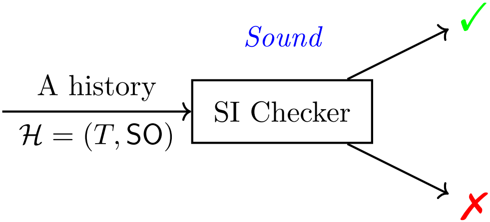
[a] We take the common "UniqueValue" assumption on histories: for each key, every write to the key assigns a unique value.



SO : *session order* among the set T of transactions

The SI Checking Problem

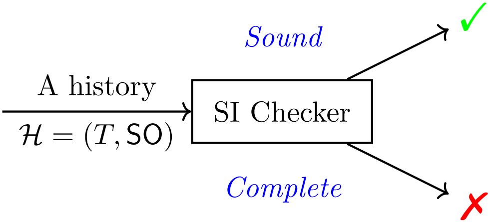
Sound: If the checker says ✗, then the history does *not* satisfy SI.



The SI Checking Problem

Sound: If the checker says ✗, then the history does *not* satisfy SI.

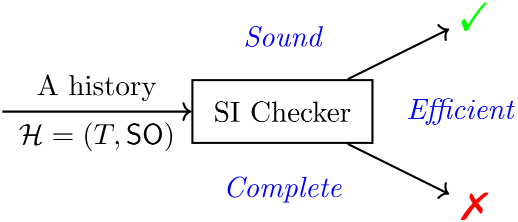
Complete: If the checker says ✓, then the history *satisfies* SI.



The SI Checking Problem

Sound: If the checker says ✗, then the history does *not* satisfy SI.

Complete: If the checker says ✓, then the history *satisfies* SI.

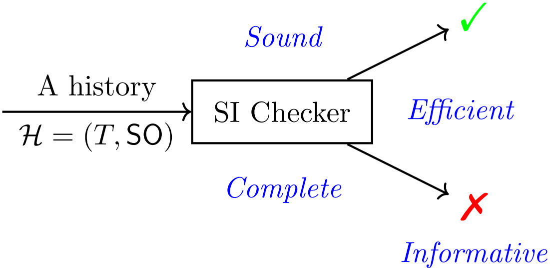


Efficient: The checker should *scale* up to large workloads.

The SI Checking Problem

Sound: If the checker says ✗, then the history does *not* satisfy SI.

Complete: If the checker says ✓, then the history *satisfies* SI.



Efficient: The checker should *scale* up to large workloads.

Informative: The checker should provide understandable *counterexamples* if it finds violations.

Related Work

dbcop [Biswas and Enea, 2019] checker for SI

not practically efficient;

not informative, returning only "False" upon violations

Cobra [Tan et al., 2020] state-of-the-art checker for SER

The SI checking problem is *harder*.

Related Work

Elle [Kingsbury and Alvaro, 2020] checker for various isolation levels

Work perfectly on traceable and recoverable histories; but may be incomplete on the key-value datatype

[a] Could Elle tell the difference between snapshot isolation and strong-session-snapshot-isolation? https://github.com/jepsen-io/elle/issues/17

[b] Elle may miss two types of transaction anomalies. https://github.com/jepsen-io/elle/issues/21

Related Work

Elle [Kingsbury and Alvaro, 2020] checker for various isolation levels

Work perfectly on traceable and recoverable histories; but may be incomplete on the key-value datatype

SI checking based on the Adya-style notions [Adya, 1999] relies on the start/commit timestamps of transactions.

[a] Could Elle tell the difference between snapshot isolation and strong-session-snapshot-isolation? https://github.com/jepsen-io/elle/issues/17

[b] Elle may miss two types of transaction anomalies. https://github.com/jepsen-io/elle/issues/21

Related Work

Elle [Kingsbury and Alvaro, 2020] checker for various isolation levels

Work perfectly on traceable and recoverable histories; but may be incomplete on the key-value datatype

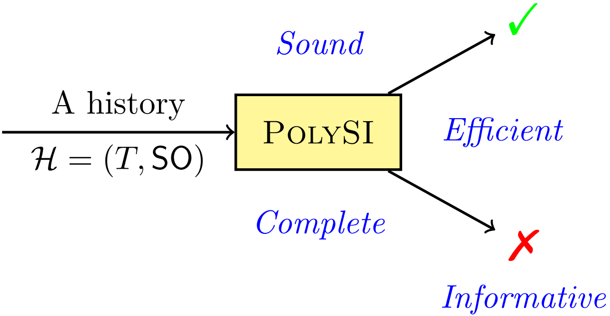
SI checking based on the Adya-style notions [Adya, 1999] relies on the start/commit timestamps of transactions.

SI checking based on the [Cerone and Gotsman, 2018] notions may miss SI violations.[a] [b]

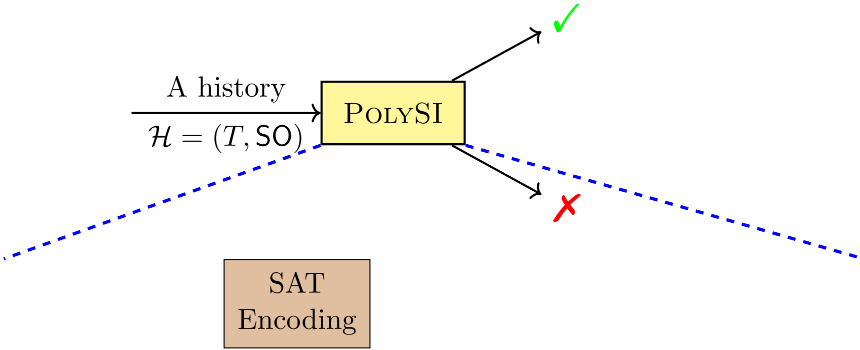
[a] Could Elle tell the difference between snapshot isolation and strong-session-snapshot-isolation? https://github.com/jepsen-io/elle/issues/17

[b] Elle may miss two types of transaction anomalies. https://github.com/jepsen-io/elle/issues/21

Contribution: the PolySI Checker

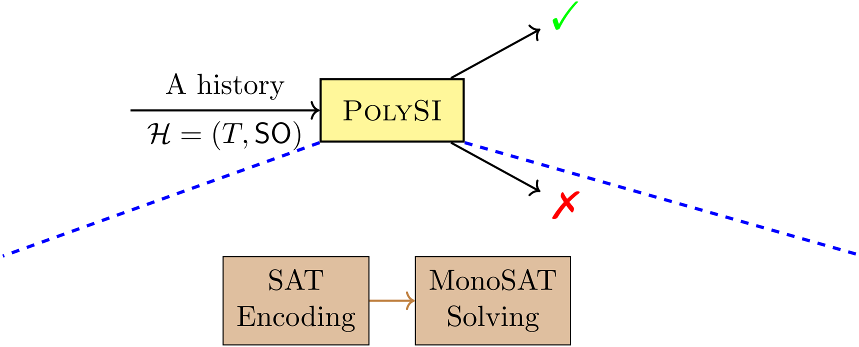


Contribution: the PolySI Checker



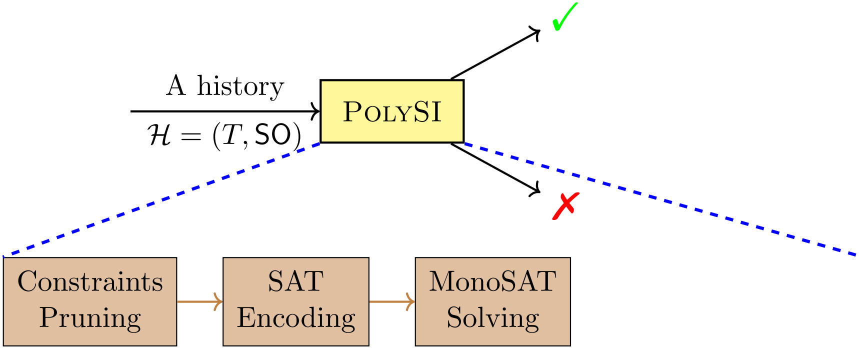
Sound & Complete: a novel *polygraph* based characterization of SI

Contribution: the PolySI Checker

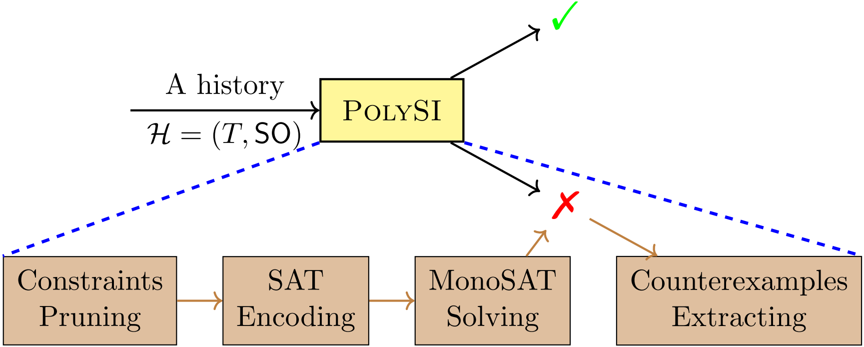


Efficient: utilizing MonoSAT solver optimized for graph problems

Contribution: the PolySI Checker



Efficient: pruning the constraints on *polygraph* before encoding



Informative: extract counterexamples from the UNSAT core

PolySI: Polygraph based Characterization of SI

Before this, we first review the *dependency graph* based characterization of SI [Cerone and Gotsman, 2018].

Theorem (Theorem 4.1 of [Cerone and Gotsman, 2018])

Informally, a history satisfies SI if and only if there exists a dependency graph for it that contains only cycles (if any) with at least two adjacent RW *edges.*

Dependency Graph based Characterization of SI

T_A

$\boxed{\begin{array}{c}\mathsf{R}(acct, 0) \\ \mathsf{W}(acct, 50)\end{array}}$

T_0

$\boxed{\mathsf{W}(acct, 0)}$

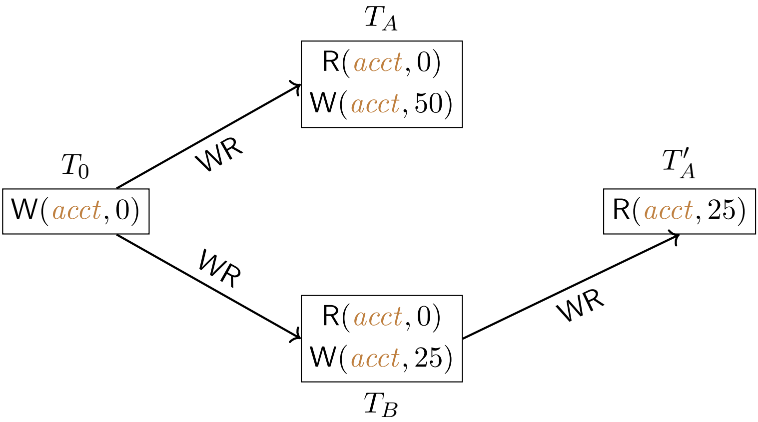
T'_A

$\boxed{\mathsf{R}(acct, 25)}$

$\boxed{\begin{array}{c}\mathsf{R}(acct, 0) \\ \mathsf{W}(acct, 25)\end{array}}$

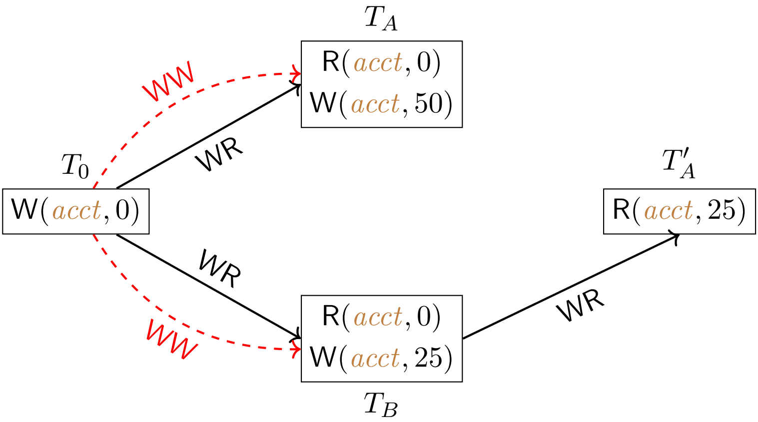
T_B

Dependency Graph based Characterization of SI



WR: “write-read” dependency capturing the “read-from” relation

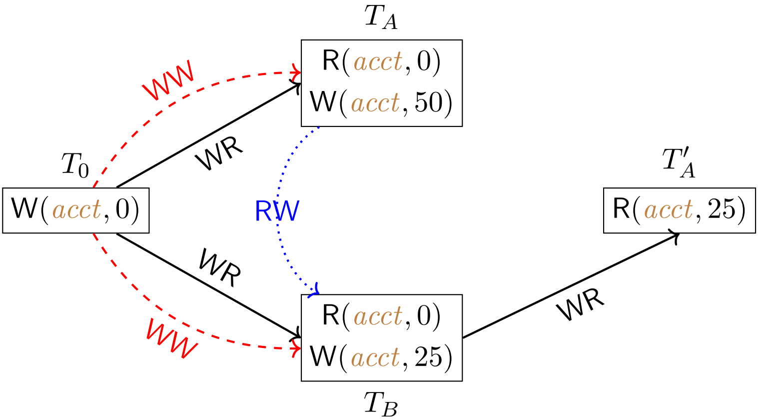
Dependency Graph based Characterization of SI



WW: "write-write" dependency capturing the version order on *acct*

Dependency Graph based Characterization of SI

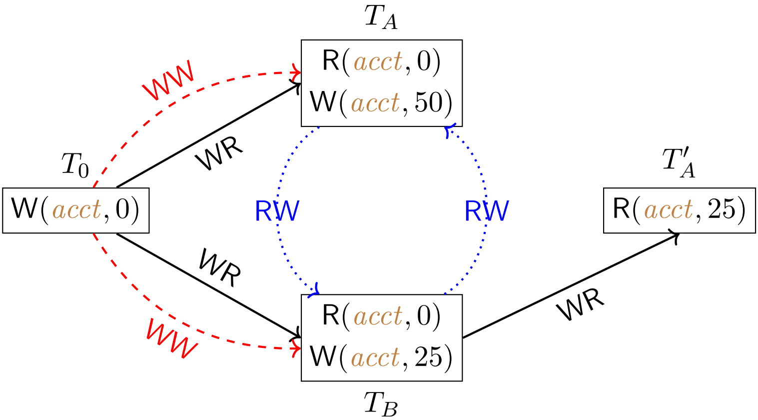
T_A reads from T_0 which is overwritten by T_B



RW: "read-write" dependency

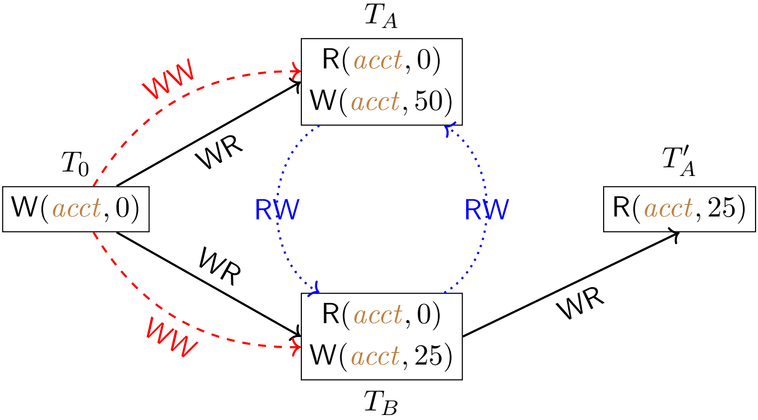
Dependency Graph based Characterization of SI

T_B reads from T_0 which is overwritten by T_A



RW: "read-write" dependency

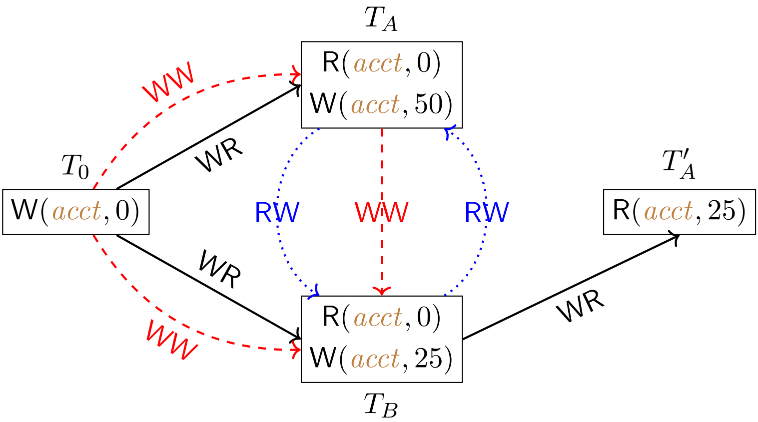
Dependency Graph based Characterization of SI



The cycle $T_A \xrightarrow{\text{RW}} T_B \xrightarrow{\text{RW}} T_A$ is allowed by SI.

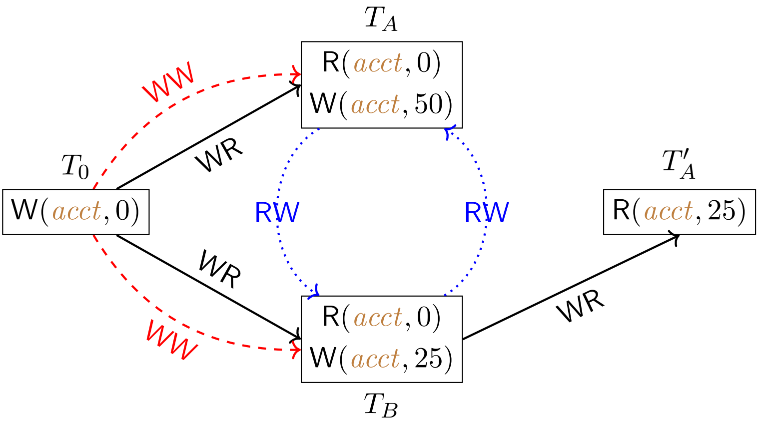
Dependency Graph based Characterization of SI

Suppose that $T_A \xrightarrow{\text{WW}} T_B$



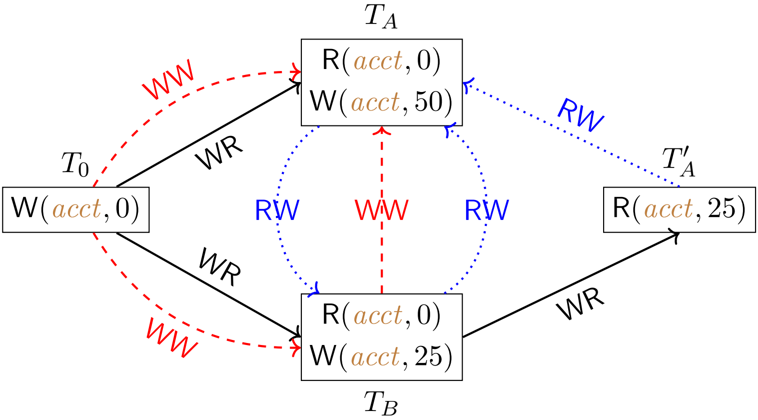
undesired cycle for SI: $T_A \xrightarrow{\text{WW}} T_B \xrightarrow{\text{RW}} T_A$

Dependency Graph based Characterization of SI



Dependency Graph based Characterization of SI

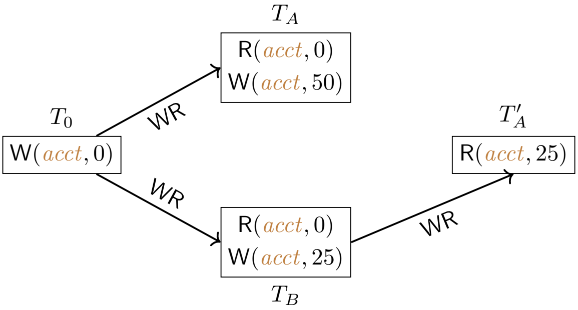
$\boxed{\textit{Suppose that } T_B \xrightarrow{\mathsf{WW}} T_A}$



undesired cycle for SI: $T_B \xrightarrow{\mathsf{WW}} T_A \xrightarrow{\mathsf{RW}} T_B$

Dependency Graph based Characterization of SI

We have considered both bases $T_A \xrightarrow{\text{WW}} T_B$ and $T_B \xrightarrow{\text{WW}} T_A$, and each case leads to an undesired cycle for SI.



Therefore, this history does not satisfy SI.

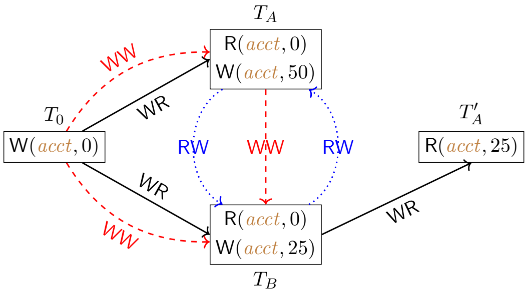
Dependency Graph based Characterization of SI

Theorem (Equivalence of Theorem 4.1 of [Cerone and Gotsman, 2018])

Informally, a history satisfies SI if and only if
there exists a dependency graph $\mathcal{G}$ for it such that
the induced graph of $\mathcal{G}$ $\boxed{((\mathsf{SO}_{\mathcal{G}} \cup \mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}}) \ ; \ \mathsf{RW}_{\mathcal{G}}?)}$ *is acyclic.*

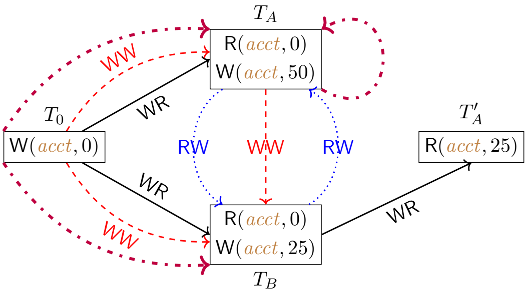
Dependency Graph based Characterization of SI

induced graph $\boxed{((\mathsf{SO}_{\mathcal{G}} \cup \mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}})\ ;\ \mathsf{RW}_{\mathcal{G}}?)}$ *for* $\mathcal{G}$



Dependency Graph based Characterization of SI

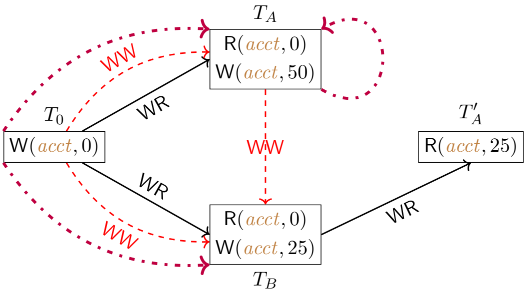
$$\textit{induced graph}\ \boxed{((\mathsf{SO}_{\mathcal{G}} \cup \mathsf{WR}_{\mathcal{G}} \cup \textcolor{red}{\mathsf{WW}_{\mathcal{G}}})\ ;\ \textcolor{blue}{\mathsf{RW}_{\mathcal{G}}}?)}\ \textit{for}\ \mathcal{G}$$



first composing (;) the SO/WR/WW edges with the RW edges

Dependency Graph based Characterization of SI

induced graph $\boxed{((\mathsf{SO}_{\mathcal{G}} \cup \mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}})\ ;\ \mathsf{RW}_{\mathcal{G}}?)}$ *for* $\mathcal{G}$

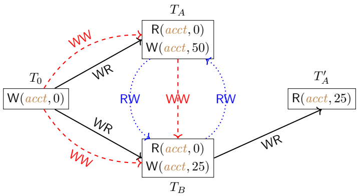


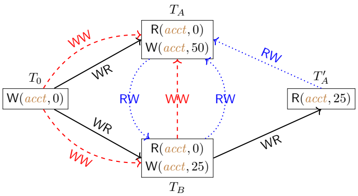
first composing (;) the SO/WR/WW edges with the RW edges

then deleting all the RW edges

Polygraph: A Family of Dependency Graphs

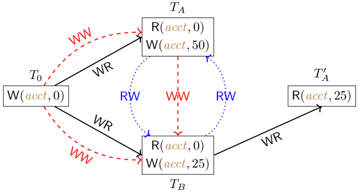
Consider the two cases of WW dependencies between T_A and T_B.

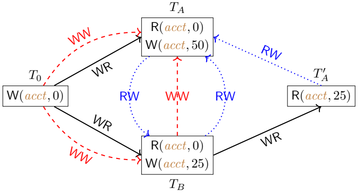


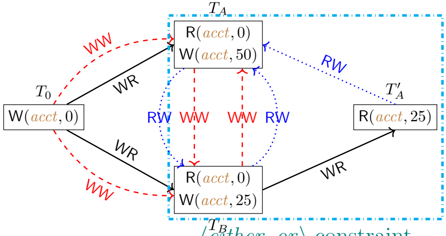


Polygraph: A Family of Dependency Graphs

Consider the two cases of WW dependencies between T_A and T_B.





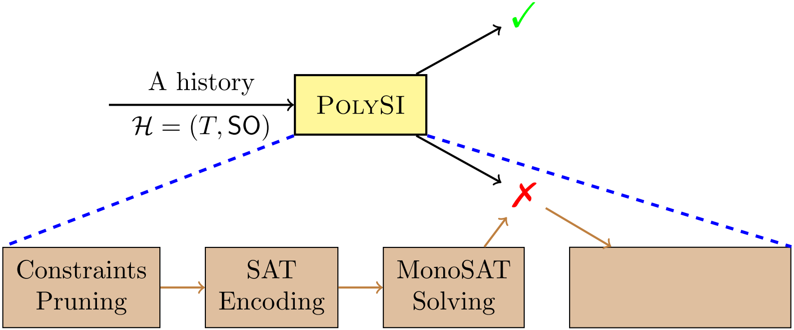


$\langle either, or \rangle$ constraint

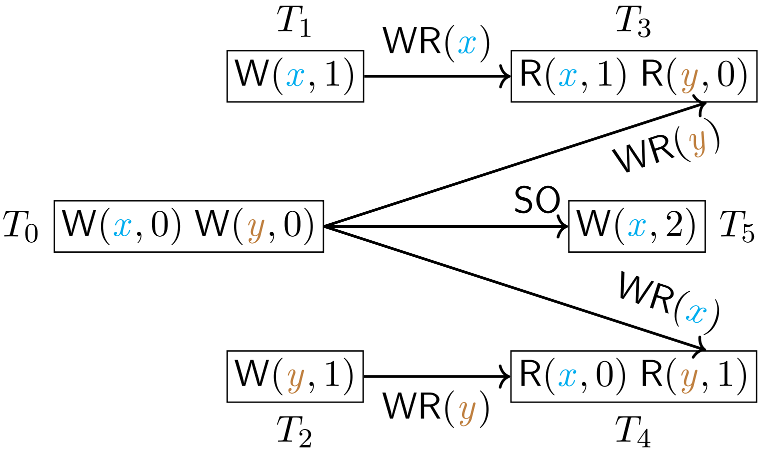
polygraph: $\langle either \triangleq \{T_A \xrightarrow{\text{WW}} T_B\}, or \triangleq \{T_B \xrightarrow{\text{WW}} T_A, T'_A \xrightarrow{\text{RW}} T_A\}\rangle$

PolySI: A Running Example

To explain the whole PolySI procedure with a running example.

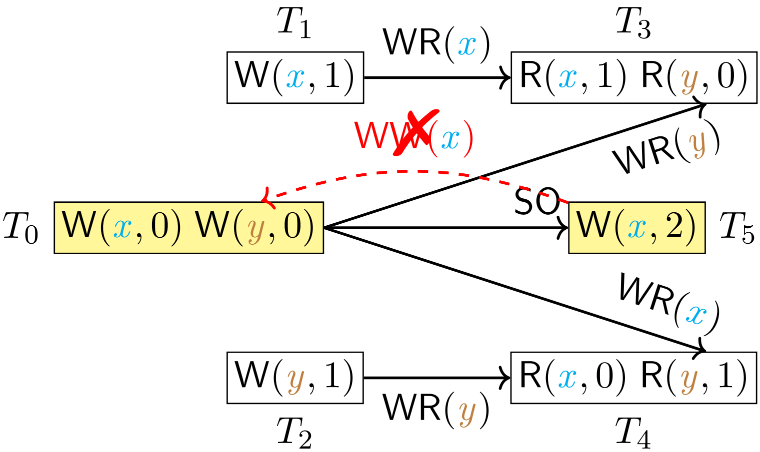


PolySI: A Running Example



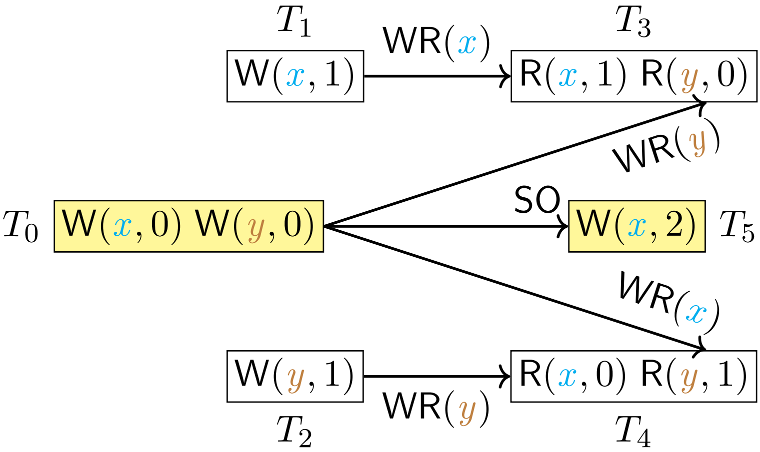
WW between T_0, T_1, and T_5 (on x) and between T_0 and T_2 (on y)

PolySI: A Running Example

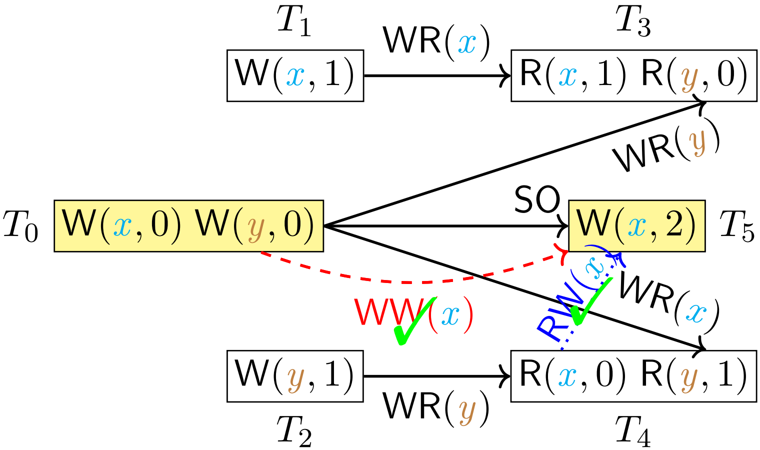


The $T_5 \xrightarrow{\text{WW}(x)} T_0$ case is pruned due to $T_0 \xrightarrow{\text{SO}} T_5 \xrightarrow{\text{WW}(x)} T_0$.

PolySI: A Running Example

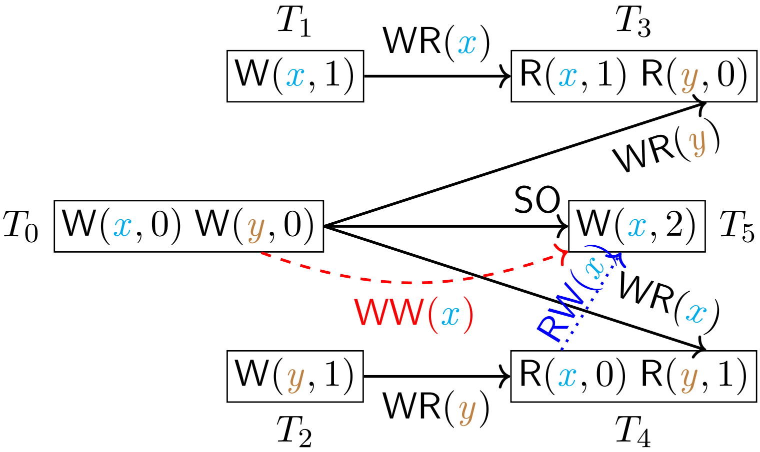


PolySI: A Running Example

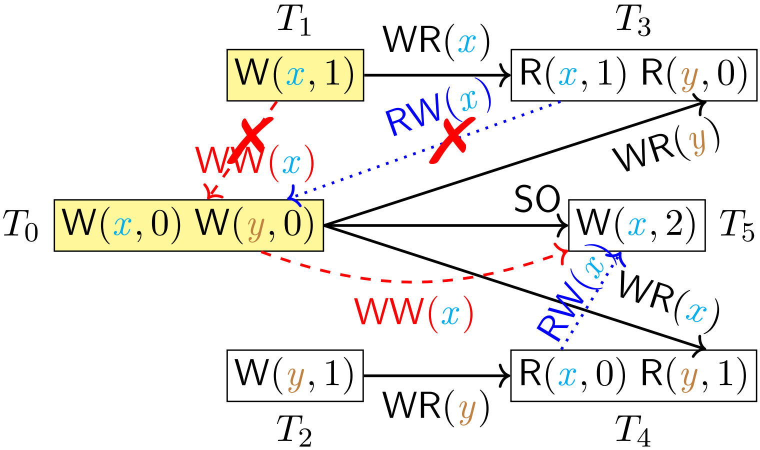


The $T_0 \xrightarrow{\mathsf{WW}(x)} T_5$ case becomes known.

PolySI: A Running Example

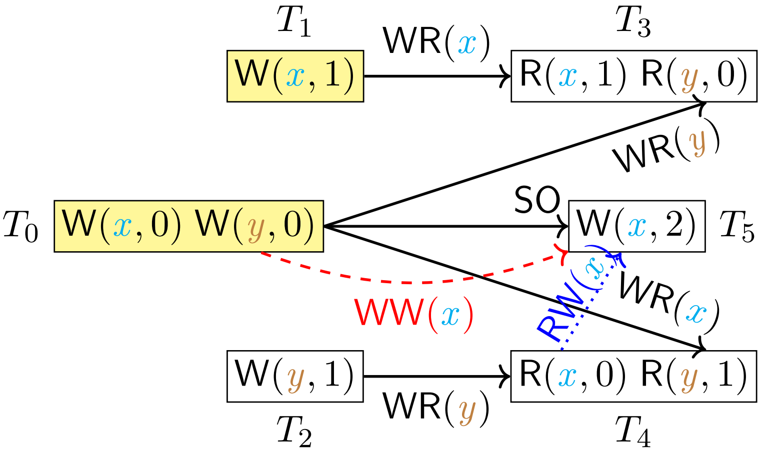


PolySI: A Running Example

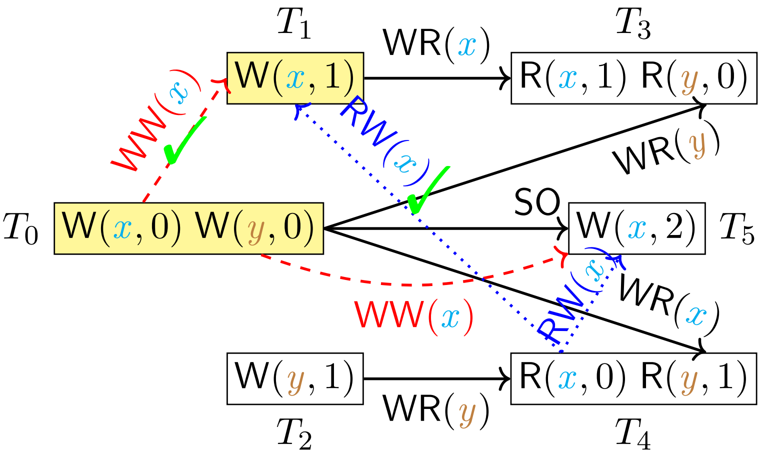


The $T_1 \xrightarrow{\text{WW}(x)} T_0$ case is pruned due to $T_3 \xrightarrow{\text{RW}(x)} T_0 \xrightarrow{\text{WR}(y)} T_3$.

PolySI: A Running Example

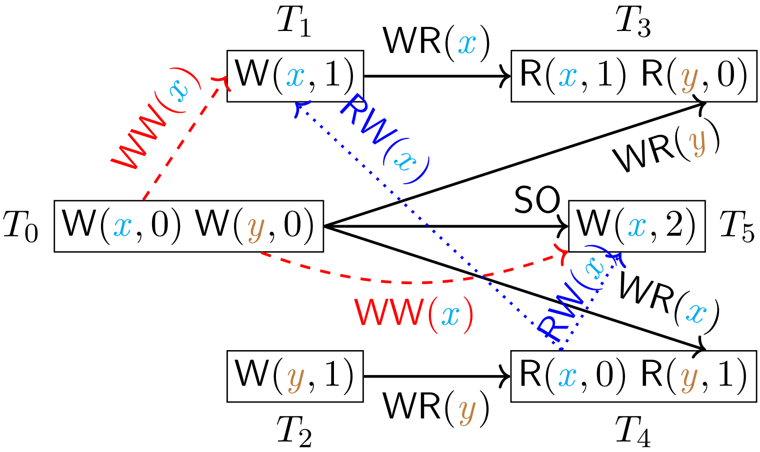


PolySI: A Running Example

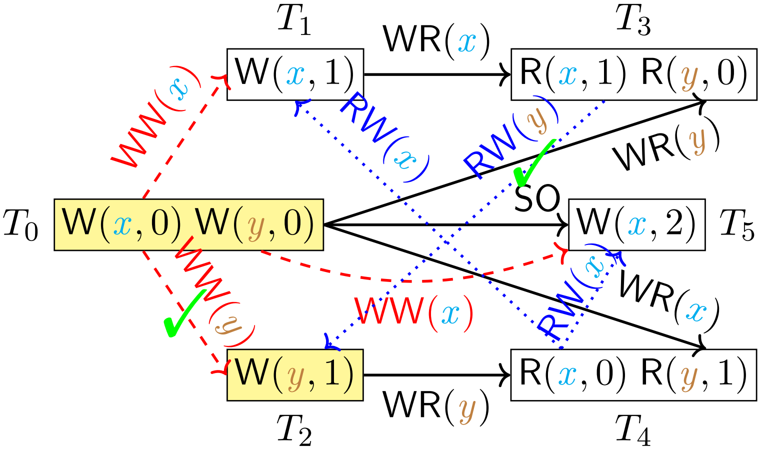


The $T_0 \xrightarrow{\mathsf{WW}(x)} T_1$ case becomes known.

PolySI: A Running Example



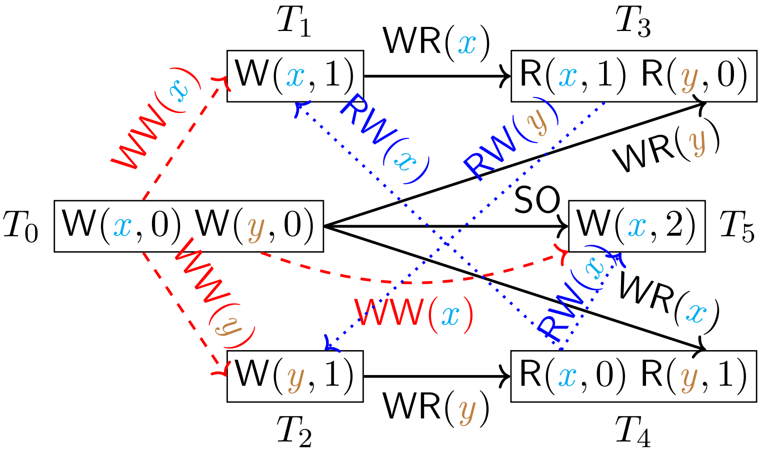
PolySI: A Running Example



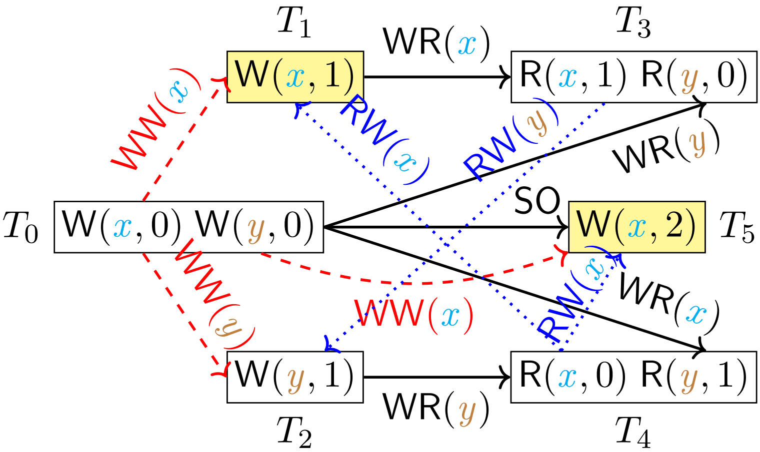
The $T_2 \xrightarrow{\text{WW}(y)} T_0$ case is pruned,

while the $T_0 \xrightarrow{\text{WW}(y)} T_2$ case becomes known.

PolySI: A Running Example



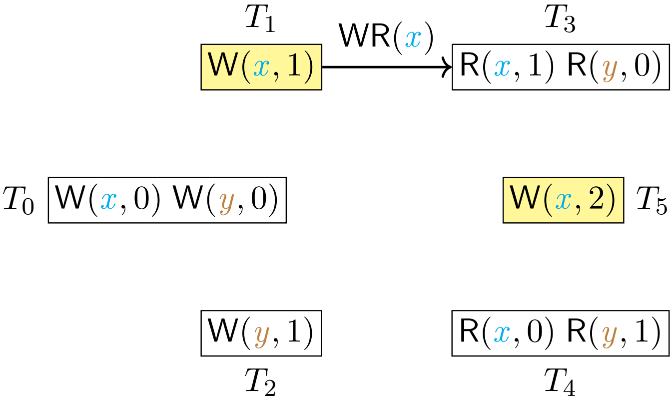
PolySI: A Running Example



The WW order between T_1 and T_5 is still uncertain after pruning.

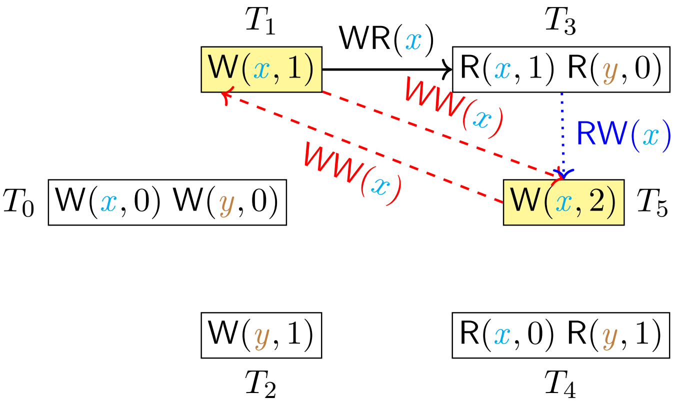
PolySI: A Running Example

$\langle \qquad , \qquad \rangle$



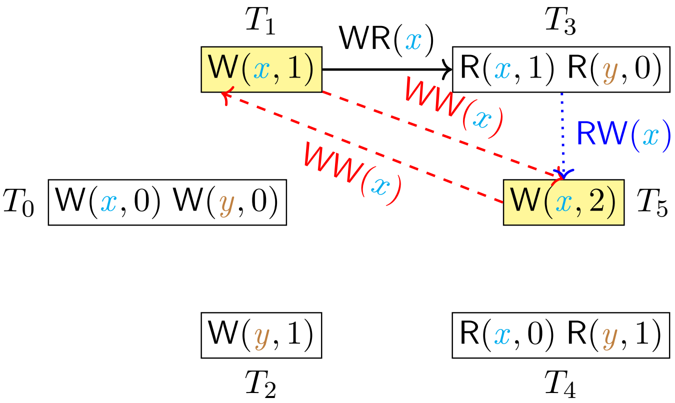
PolySI: A Running Example

$$\langle either = \{T_1 \xrightarrow{\mathsf{WW}(x)} T_5, T_3 \xrightarrow{\mathsf{RW}(x)} T_5\}, or = \{T_5 \xrightarrow{\mathsf{WW}(x)} T_1\}\rangle$$



PolySI: A Running Example

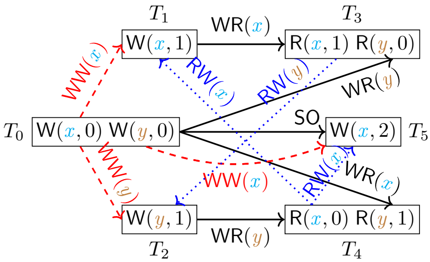
$$\langle either = \{T_1 \xrightarrow{\mathsf{WW}(x)} T_5, T_3 \xrightarrow{\mathsf{RW}(x)} T_5\}, or = \{T_5 \xrightarrow{\mathsf{WW}(x)} T_1\}\rangle$$

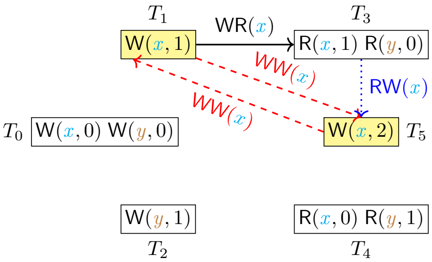


$$\underbrace{(\mathsf{BV}_{1,5} \land \mathsf{BV}_{3,5} \land \neg\mathsf{BV}_{5,1})}_{either} \lor \underbrace{(\mathsf{BV}_{5,1} \land \neg\mathsf{BV}_{1,5} \land \neg\mathsf{BV}_{3,5})}_{or}$$

PolySI: A Running Example

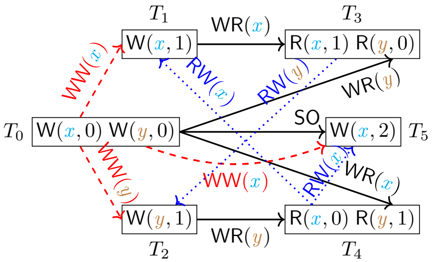
induced graph $\mathcal{I}$ $\boxed{((\mathsf{SO}_{\mathcal{G}} \cup \mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}}) \ ; \ \mathsf{RW}_{\mathcal{G}}?)}$

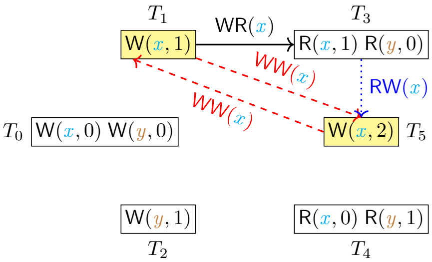




PolySI: A Running Example

induced graph $\mathcal{I}$ $\boxed{((\mathsf{SO}_{\mathcal{G}} \cup \mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}}) \; ; \; \mathsf{RW}_{\mathcal{G}}?)}$





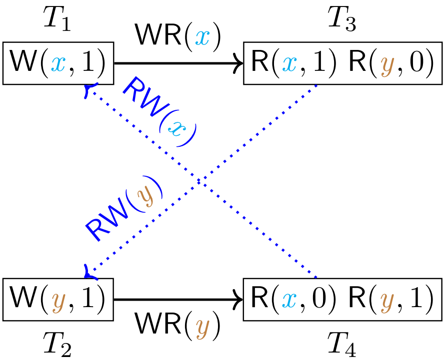
$$T_1 \xrightarrow{\mathsf{WR}} T_3 \xrightarrow{\mathsf{RW}} T_5 : \; \mathsf{BV}^{\mathcal{I}}_{1,5} = \mathsf{BV}_{1,3} \; \wedge \; \mathsf{BV}_{3,5}$$

The presence of the edge $T_1 \to T_5$ in the induced graph $\mathcal{I}$ depends on that of the edges $T_1 \to T_3$ and $T_3 \to T_5$ in the polygraph.

PolySI: A Running Example

Feed the SAT formula into the MonoSAT solver [Bayless et al., 2015] which is optimized for *cycle detection*.

PolySI: A Running Example



The MonoSAT solver finds an undesired cycle for SI.

Experimental Evaluation

(1) *Effective:* Can PolySI find SI violations in production databases?

(2) *Informative:* Can PolySI provide understandable counterexamples for SI violations?

(3) *Efficient:* How efficient is PolySI?

Workloads

Table: Workload parameters and their default values.[c]

Parameter	Default Value
#sess	20
#txns/sess	100
#ops/txn	15
#keys	10, 000
%reads	50%
distribution	zipfian

[c] We use a database schema of a *two-column table* storing keys and values.

Benchmarks

RuBiS: an eBay-like bidding system

TPC-C: an open standard for OLTP benchmarking

C-Twitter: a Twitter clone

GeneralRH: read-heavy workloads with 95% reads

GeneralRW: medium workloads with 50% reads

GeneralWH: write-heavy workloads with 30% reads

Reproducing Known SI Violations

Database	GitHub Stars	Kind	Release
CockroachDB[d]	25.1k	Relational	v2.1.0[e], v2.1.6
MySQL-Galera	381	Relational	v25.3.26
YugabyteDB	6.7k	Multi-model	v1.1.10.0[f]

An extensive collection of 2477 anomalous histories

[Biswas and Enea, 2019; Darnell, Accessed February 14, 2023; Jepsen, Accessed February 14, 2023]

[d] Remove SNAPSHOT isolation since (probably) v2.0.4. https://github.com/cockroachdb/cockroach/pull/27040

[e] Lessons learned from 2+ years of nightly Jepsen tests. https://www.cockroachlabs.com/blog/jepsen-tests-lessons/

[f] Acknowledged inserts can be present in reads for tens of seconds, then disappear. https://github.com/YugaByte/yugabyte-db/issues/824

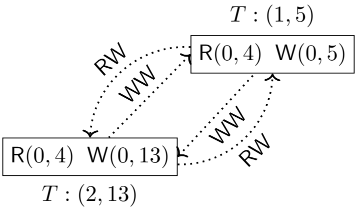
Detecting New SI Violations

Dgraph: helped the Dgraph team confirm some of their suspicions about their latest release

Database	**GitHub Stars**	**Kind**	**Release**
Dgraph	18.2k	Graph	v21.12.0
MariaDB-Galera	4.4k	Relational	v10.7.3
YugabyteDB	6.7k	Multi-model	v2.11.1.0

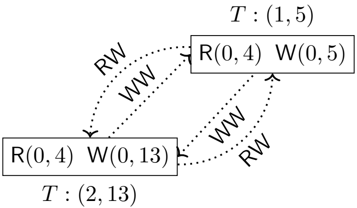
Galera: confirmed the incorrect claim on preventing “lost updates” for transactions issued on different cluster nodes

Understanding Violations (Lost Update)

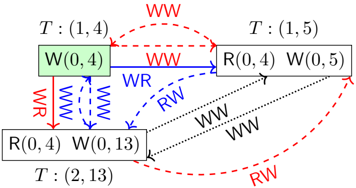


(a) Original output

Understanding Violations (Lost Update)

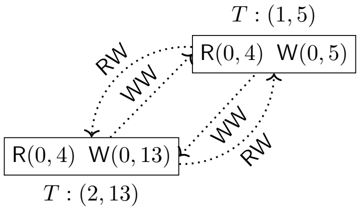


(a) Original output

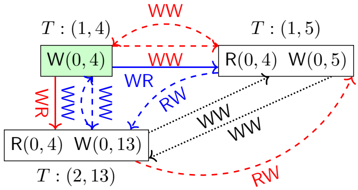


(b) Missing participants

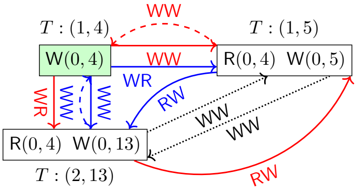
Understanding Violations (Lost Update)



(a) Original output

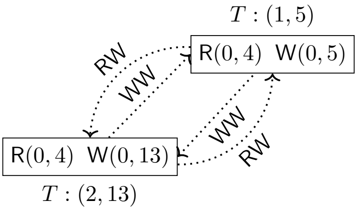


(b) Missing participants

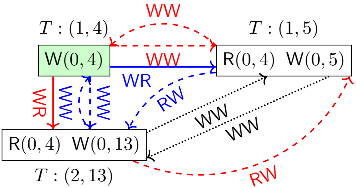


(c) Recovered scenario

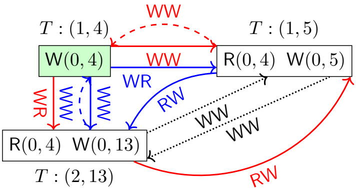
Understanding Violations (Lost Update)



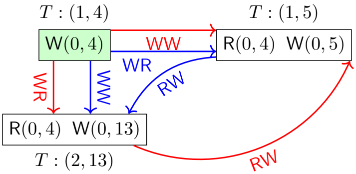
(a) Original output



(b) Missing participants



(c) Recovered scenario



(d) Finalized scenario

Performance Evaluation

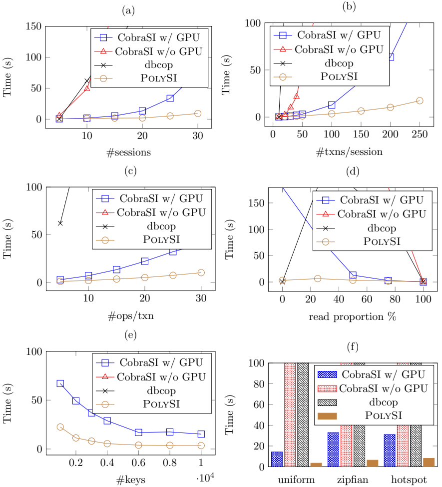
dbcop [Biswas and Enea, 2019]: the state-of-the-art SI checker

CobraSI: reducing SI checking to SER checking [Biswas and Enea, 2019] to leverage Cobra with/without GPU

Cobra [Tan et al., 2020]: the state-of-the-art SER checker using both MonoSAT and GPU

Performance Evaluation: Runtime

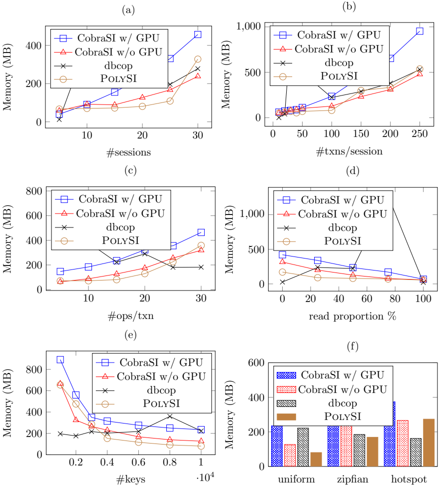
PolySI significantly outperforms the competitors.[g]



[g] All the input histories extracted from PostgreSQL satisfy SI.

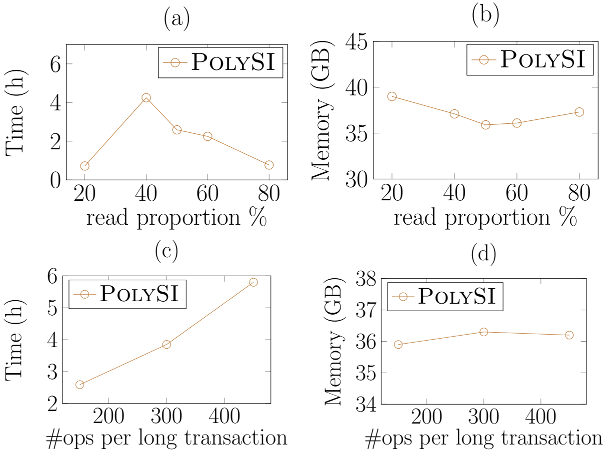
Performance Evaluation: Memory

PolySI consumes less memory.



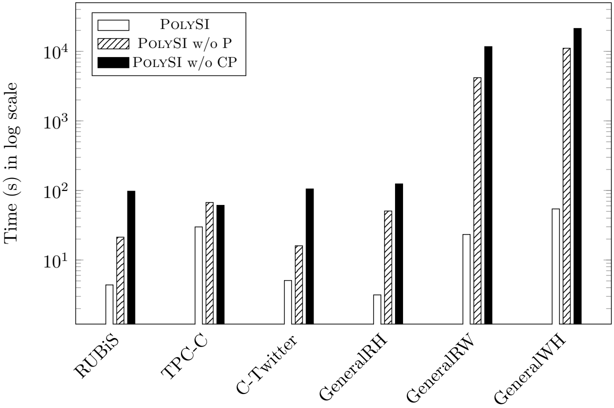
Performance Evaluation: Scalability

several hours and 35 $\sim$ 40GB memory for checking 1M transactions



Performance Evaluation: Differential Analysis

Pruning (P) is crucial to the efficiency of PolySI.[h]



[h] Compacting (C) encoding has been omitted in this presentation.

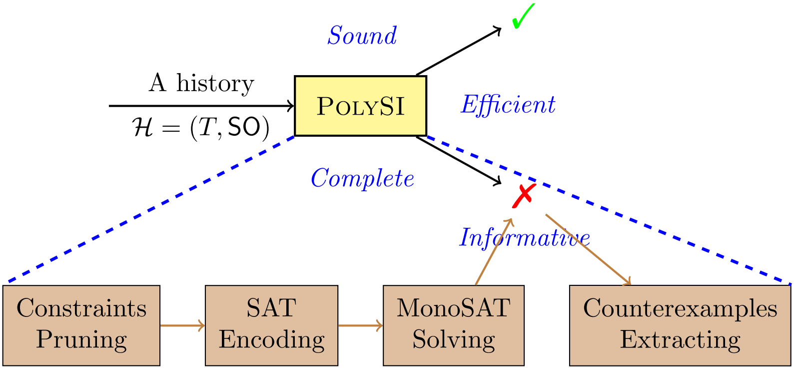
Performance Evaluation: Pruning

PolySI can prune a huge number of constraints before encoding.

Benchmark	#cons. before P	#cons. **after P**	#unk. dep. before P	#unk. dep. **after P**
TPC-C	386k	0	3628k	0
GeneralRH	4k	29	39k	77
RUBiS	14k	149	171k	839
C-Twitter	59k	277	307k	776
GeneralRW	90k	2565	401k	5435
GeneralWH	167k	6962	468k	14376

TPC-C: read-only transactions + RMW transactions

Conclusion



Sound
✓
A history
PolySI
$\mathcal{H} = (T, \mathsf{SO})$
Efficient
Complete
✗
Informative
Constraints Pruning
SAT Encoding
MonoSAT Solving
Counterexamples Extracting

Future Work

PolySI uses MonoSAT as a black-box.

Working on a **theory solver** dedicated to isolation level checking, which is deeply integrated with SAT solvers [He, Sun, and Fan, 2021].

Hengfeng Wei (hfwei@nju.edu.cn)

- Adya, Atul (1999). "Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions". PhD thesis. USA.
- Bayless, Sam, Noah Bayless, Holger H. Hoos, and Alan J. Hu (2015). "SAT modulo Monotonic Theories". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. AAAI Press, pp. 3702–3709. ISBN: 0262511290.
- Biswas, Ranadeep and Constantin Enea (Oct. 2019). "On the Complexity of Checking Transactional Consistency". In: *Proc. ACM Program. Lang.* 3.OOPSLA. DOI: 10.1145/3360591. URL: https://doi.org/10.1145/3360591.
- Cerone, Andrea and Alexey Gotsman (Jan. 2018). "Analysing Snapshot Isolation". In: *J. ACM* 65.2. ISSN: 0004-5411. DOI: 10.1145/3152396. URL: https://doi.org/10.1145/3152396.
- Darnell, Ben (Accessed February 14, 2023). *Lessons Learned from 2+ Years of Nightly Jepsen Tests*. https://www.cockroachlabs.com/blog/jepsen-tests-lessons/.

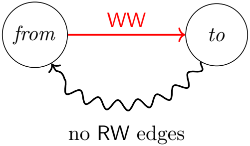
He, Fei, Zhihang Sun, and Hongyu Fan (2021). "Satisfiability modulo Ordering Consistency Theory for Multi-Threaded Program Verification". In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. PLDI 2021. Virtual, Canada: Association for Computing Machinery, pp. 1264–1279. ISBN: 9781450383912. DOI: 10.1145/3453483.3454108. URL: https://doi.org/10.1145/3453483.3454108.

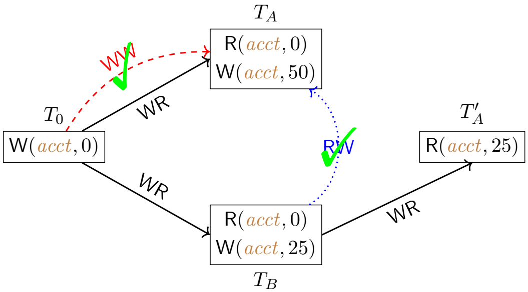
Jepsen (Accessed February 14, 2023). *Issue #824*. https://github.com/YugaByte/yugabyte-db/issues/824.

Kingsbury, Kyle and Peter Alvaro (Nov. 2020). "Elle: Inferring Isolation Anomalies from Experimental Observations". In: *Proc. VLDB Endow.* 14.3, pp. 268–280. ISSN: 2150-8097.

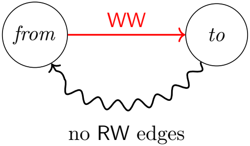
Tan, Cheng, Changgeng Zhao, Shuai Mu, and Michael Walfish (2020). "COBRA: Making Transactional Key-Value Stores Verifiably Serializable". In: *OSDI'20*. ISBN: 978-1-939133-19-9.

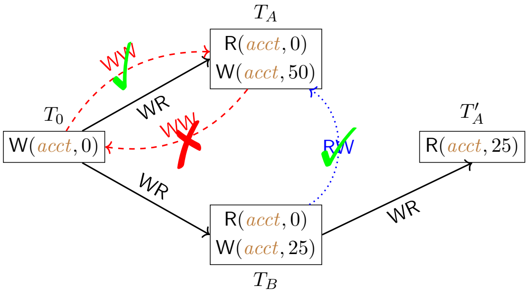
PolySI: Pruning before Encoding (the WW case)





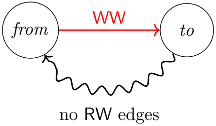
PolySI: Pruning before Encoding (the WW case)

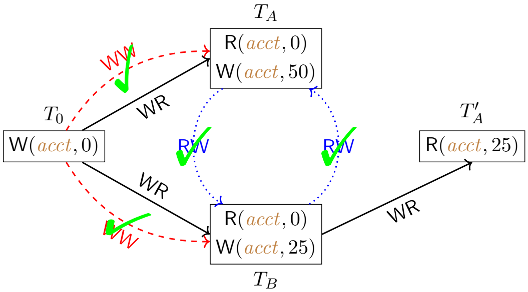




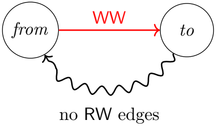
$T_A \xrightarrow{\text{WW}} T_0$ can be pruned due to the $T_A \xrightarrow{\text{WW}} T_0 \xrightarrow{\text{WR}} T_A$ cycle.

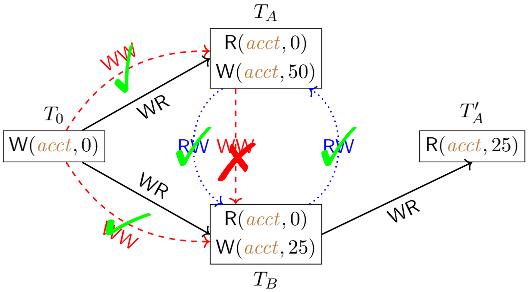
PolySI: Pruning before Encoding (the WW case)





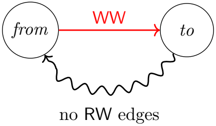
PolySI: Pruning before Encoding (the WW case)

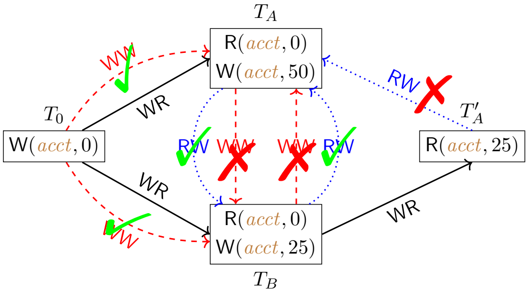




$T_A \xrightarrow{\text{WW}} T_B$ is pruned due to the $T_A \xrightarrow{\text{WW}} T_B \xrightarrow{\text{RW}} T_A$ cycle.

PolySI: Pruning before Encoding (the WW case)

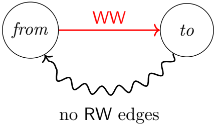


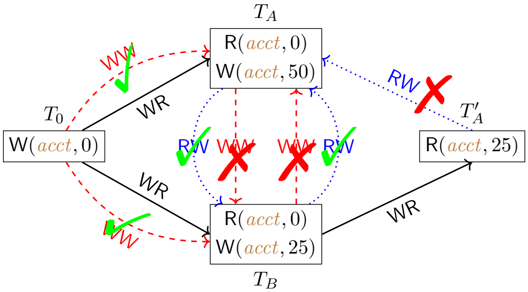


$T_A \xrightarrow{\text{WW}} T_B$ is pruned due to the $T_A \xrightarrow{\text{WW}} T_B \xrightarrow{\text{RW}} T_A$ cycle.

$T_B \xrightarrow{\text{WW}} T_A$ is pruned due to the $T_B \xrightarrow{\text{WW}} T_A \xrightarrow{\text{RW}} T_B$ cycle.

PolySI: Pruning before Encoding (the WW case)





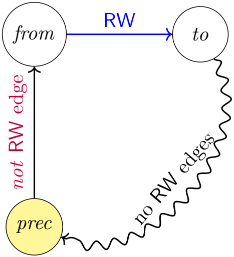
$T_A \xrightarrow{\text{WW}} T_B$ is pruned due to the $T_A \xrightarrow{\text{WW}} T_B \xrightarrow{\text{RW}} T_A$ cycle.

$T_B \xrightarrow{\text{WW}} T_A$ is pruned due to the $T_B \xrightarrow{\text{WW}} T_A \xrightarrow{\text{RW}} T_B$ cycle.

Therefore, we are sure that the history does *not* satisfy SI.

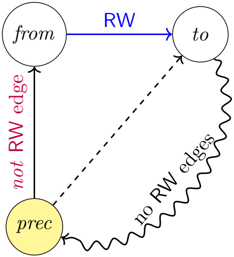
PolySI: Pruning before Encoding (the RW case)

PolySI: Pruning before Encoding (the RW case)



Check if there is a path from *to* to any immediate predecessor *prec* of *from* that does not contain RW edges.

PolySI: Pruning before Encoding (the RW case)



Check if there is a path from *to* to any immediate predecessor *prec* of *from* that does not contain RW edges.

Dependency Graph based Characterization of SI

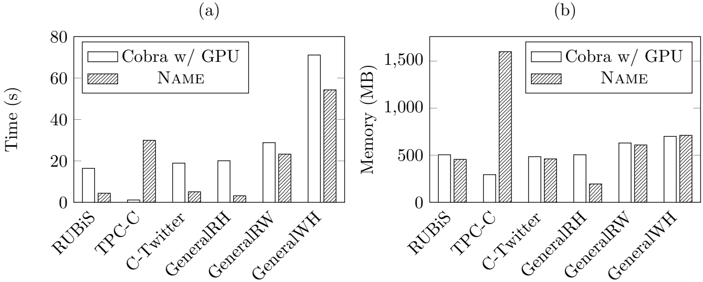
Theorem (Theorem 4.1 of [Cerone and Gotsman, 2018])

For a history $\mathcal{H} = (T, \mathsf{SO})$,

$$\mathcal{H} \models \text{SI} \iff \mathcal{H} \models \textsc{Int} \land$$
$$\exists\, \mathsf{WR}, \mathsf{WW}, \mathsf{RW}.\ \mathcal{G} = (\mathcal{H}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW}) \land$$
$$(((\mathsf{SO}_{\mathcal{G}} \cup \mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}})\ ;\ \mathsf{RW}_{\mathcal{G}}?)\ \textit{ is acyclic}).$$

Performance Evaluation: Cobra with GPU

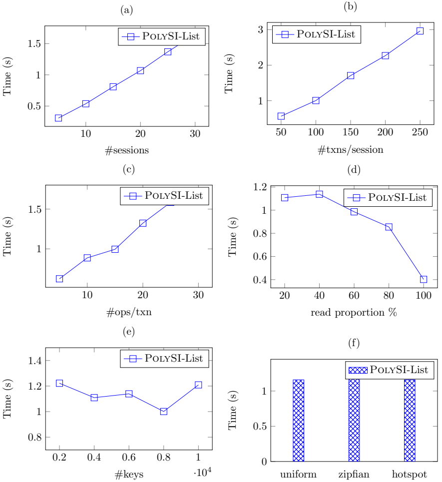
PolySI outperforms Cobra with GPU in 5 of the 6 benchmarks.



Cobra implements a specific optimization for RMW workloads (like TPC-C) *before* pruning and encoding.

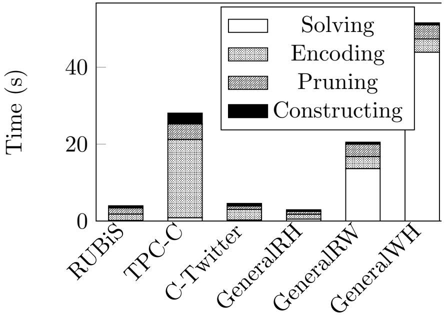
Performance Evaluation: PolySI-List

Integrate list inference of Elle [Kingsbury and Alvaro, 2020] into PolySI



Performance Evaluation: Decomposition

TPC-C incurs more overhead in *encoding* as the number of operations in total is 5x more than the others.



The *solving* time depends on the remaining constraints and unknown dependencies after pruning.