Plain and Simple Inductive Invariant Inference for Distributed Protocols in TLA+

William Schultz[*]**,** Ian Dardik[†], Stavros Tripakis[*]

FMCAD 2022

Northeastern University[*]*, Carnegie Mellon University*[†]

Designing Distributed Systems

Designing Distributed Systems

- Distributed systems found in all modern cloud, data storage systems

Designing Distributed Systems

- Distributed systems found in all modern cloud, data storage systems
- Underlying protocols are difficult to get right, error-prone [1]

redis

mongoDB

aws

Cockroach DB

Cloud Spanner

Tendermint

Table of errors

Protocol	Reference	Violation	Counter-example
PBFT[1]	[Castro and Liskov 1999]	liveness	[Berger et al. 2021]
Chord	[Stoica et al. 2001; Liben-Nowell et al. 2002]	liveness[2]	[Zave 2012; Zave 2017]
Pastry	[Rowstron and Druschel 2001]	safety	[Azmy et al. 2016; Azmy et al. 2018]
Generalised Paxos	[Lamport 2005]	non-triviality[3]	[Sutra and Shapiro 2010]
FaB Paxos	[Martin and Alvisi 2005; Martin and Alvisi 2006]	liveness	[Abraham et al. 2017]
Multi-Paxos[4]	[Chandra et al. 2007]	safety	[Michael et al. 2017]
Zyzzyva	[Kotla et al. 2007; Kotla et al. 2010]	safety	[Abraham et al. 2017]
CRAQ	[Terrace and Freedman 2009]	safety[5]	[Whittaker 2020]
JPaxos	[Kończak et al. 2011]	safety	[Michael et al. 2017]
VR Revisited	[Liskov and Cowling 2012]	safety	[Michael et al. 2017]
EPaxos	[Moraru et al. 2013]	safety	[Sutra 2020]
EPaxos	[Moraru et al. 2013]	safety	[Whittaker 2021]
Raft	[Ongaro and Ousterhout 2014]	liveness[6]	[Hoch 2014]
Raft	[Ongaro 2014]	safety[7]	[Amos and Zhang 2015; Ongaro 2015]
Raft	[Ongaro and Ousterhout 2014; Ongaro 2014]	liveness	[Howard and Abraham 2020; Jensen et al. 2021]
hBFT	[Duan et al. 2015]	safety	[Shrestha et al. 2019]
Tendermint	[Buchman 2016]	liveness	[Cachin and Vukolić 2017]
CAESAR	[Arun et al. 2017]	liveness	[Enes et al. 2021]
DPaxos	[Nawab et al. 2018]	safety	[Whittaker et al. 2021]
Sync HotStuff	[Abraham et al. 2019]	safety & liveness	[Momose and Cruz 2019]
Gasper	[Buterin et al. 2020]	safety & liveness	[Neu et al. 2021]

[1] List of bugs found in distributed protocols: https://github.com/dranov/protocol-bugs-list

Designing Distributed Systems

- Distributed systems found in all modern cloud, data storage systems
- Underlying protocols are difficult to get right, error-prone [1]

Table of errors

Protocol	Reference	Violation	Counter-example
PBFT[1]	[Castro and Liskov 1999]	liveness	[Berger et al. 2021]
Chord	[Stoica et al. 2001; Liben-Nowell et al. 2002]	liveness[2]	[Zave 2012; Zave 2017]
Pastry	[Rowstron and Druschel 2001]	safety	[Azmy et al. 2016; Azmy et al. 2018]
VR Revisited	[Liskov and Cowling 2012]	safety	[Michael et al. 2017]
EPaxos	[Moraru et al. 2013]	safety	[Sutra 2020]
EPaxos	[Moraru et al. 2013]	safety	[Whittaker 2021]
Raft	[Ongaro and Ousterhout 2014]	liveness[6]	[Hoch 2014]
Raft	[Ongaro 2014]	safety[7]	[Amos and Zhang 2015; Ongaro 2015]
Raft	[Ongaro and Ousterhout 2014; Ongaro 2014]	liveness	[Howard and Abraham 2020; Jensen et al. 2021]
hBFT	[Duan et al. 2015]	safety	[Shrestha et al. 2019]
Tendermint	[Buchman 2016]	liveness	[Cachin and Vukolić 2017]
CAESAR	[Arun et al. 2017]	liveness	[Enes et al. 2021]
DPaxos	[Nawab et al. 2018]	safety	[Whittaker et al. 2021]
Sync HotStuff	[Abraham et al. 2019]	safety & liveness	[Momose and Cruz 2019]
Gasper	[Buterin et al. 2020]	safety & liveness	[Neu et al. 2021]

[Ongaro2015]

bug in single-server membership changes 4974 views

Subscribe

onga...@gmail.com Jul 10, 2015, 12:58:53 AM
to raft...@googlegroups.com

Hi raft-dev,

Unfortunately, I need to announce a bug in the dissertation version of membership changes (the single-server changes, not joint consensus). The bug is potentially severe, but the fix I'm proposing is easy to implement.

When Huanchen Zhang and Brandon Amos were working on a class project at CMU to formalize single-server membership changes, Huanchen found the bug and the counter-example below by hand. They contacted me over email on May 14th, and I chose to keep this quiet for a while until we had agreed upon a solution to propose to the list. After several incorrect and/or ugly attempts, I came up with the solution proposed below. I apologize for keeping this information from you for so long.

[1] List of bugs found in distributed protocols: https://github.com/dranov/protocol-bugs-list

Designing Distributed Systems

- Distributed systems found in all modern cloud, data storage systems
- Underlying protocols are difficult to get right, error-prone [1]

redis

mongoDB

aws

Cockroach DB

Cloud Spanner

Tendermint

Table of errors

Protocol	Reference	Violation	Counter-example
PBFT[1]	[Castro and Liskov 1999]	liveness	[Berger et al. 2021]
Chord	[Stoica et al. 2001; Liben-Nowell et al. 2002]	liveness[2]	[Zave 2012; Zave 2017]
Pastry	[Rowstron and Druschel 2001]	safety	[Azmy et al. 2016; Azmy et al. 2018]
VR Revisited	[Liskov and Cowling 2012]	safety	[Michael et al. 2017]
	[Moraru et al. 2013]	safety	[Sutra 2020]
			[Whittaker 2021]
			[Hoch 2014]
			Amos and Zhang 2015; Ongaro 2015]
			[Howard and Abraham 2020; Jensen et al. 2021]
			[Shrestha et al. 2019]
			[Cachin and Vukolić 2017]
			[Enes et al. 2021]
			[Whittaker et al. 2021]
			[Momose and Cruz 2019]
			[Neu et al. 2021]

[Ongaro2015]

bug in single-server membership changes 4974 views

Subscribe

onga...@gmail.com Jul 10, 2015, 12:58:53 AM
to raft...@googlegroups.com

Hi raft-dev,

Unfortunately, I need to announce a bug in the dissertation version of membership changes (the single-server changes, not joint consensus). The bug is potentially severe, but the fix I'm proposing is easy to implement.

When Huanchen Zhang and Brandon Amos were working on a class project at CMU to formalize single-server membership changes, Huanchen found the bug and the counter-example below by hand. They contacted me over email on May 14th, and I chose to keep this quiet for a while until we had agreed upon a solution to propose to the list. After several incorrect and/or ugly attempts, I came up with the solution proposed below. I apologize for keeping this information from you for so long.

[Sutra2019]

On the correctness of Egalitarian Paxos

Pierre Sutra

Télécom SudParis
9, rue Charles Fourier
91000 Évry, France

Abstract

This paper identifies a problem in both the TLA$^+$ specification and the implementation of the Egalitarian Paxos protocol. It is related to how replicas switch from one ballot to another when computing the dependencies of a command. The problem may lead replicas to diverge and break the linearizability of the replicated service.

[1] List of bugs found in distributed protocols: https://github.com/dranov/protocol-bugs-list

How to verify real world distributed protocols?

Distributed Protocol Verification

[1] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: safety verification by interactive generalization. SIGPLAN Not. 51, 6 (June 2016), 614–630. https://doi.org/10.1145/2980983.2908118
[2] https://github.com/wilcoxjay/mypyvy
[3] Leslie Lamport. 2002. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., USA.
[4] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon web services uses formal methods. Commun. ACM 58, 4 (April 2015), 66–73. https://doi.org/10.1145/2699417

Distributed Protocol Verification

- Bounded verification e.g. explicit state, bounded model checking may suffice for initial design verification.

[1] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: safety verification by interactive generalization. SIGPLAN Not. 51, 6 (June 2016), 614–630. https://doi.org/10.1145/2980983.2908118
[2] https://github.com/wilcoxjay/mypyvy
[3] Leslie Lamport. 2002. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., USA.
[4] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon web services uses formal methods. Commun. ACM 58, 4 (April 2015), 66–73. https://doi.org/10.1145/2699417

Distributed Protocol Verification

- Bounded verification e.g. explicit state, bounded model checking may suffice for initial design verification.
- Unbounded safety verification requires development of ***inductive invariant***.

[1] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: safety verification by interactive generalization. SIGPLAN Not. 51, 6 (June 2016), 614–630. https://doi.org/10.1145/2980983.2908118
[2] https://github.com/wilcoxjay/mypyvy
[3] Leslie Lamport. 2002. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., USA.
[4] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon web services uses formal methods. Commun. ACM 58, 4 (April 2015), 66–73. https://doi.org/10.1145/2699417

Distributed Protocol Verification

- Bounded verification e.g. explicit state, bounded model checking may suffice for initial design verification.
- Unbounded safety verification requires development of ***inductive invariant***.
- Finding inductive invariants manually is difficult.

[1] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: safety verification by interactive generalization. SIGPLAN Not. 51, 6 (June 2016), 614–630. https://doi.org/10.1145/2980983.2908118
[2] https://github.com/wilcoxjay/mypyvy
[3] Leslie Lamport. 2002. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., USA.
[4] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon web services uses formal methods. Commun. ACM 58, 4 (April 2015), 66–73. https://doi.org/10.1145/2699417

Distributed Protocol Verification

- Bounded verification e.g. explicit state, bounded model checking may suffice for initial design verification.
- Unbounded safety verification requires development of ***inductive invariant***.
- Finding inductive invariants manually is difficult.
- Tools for inductive invariant inference have been developed for Ivy [1], mypyvy [2], etc.

[1] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: safety verification by interactive generalization. SIGPLAN Not. 51, 6 (June 2016), 614–630. https://doi.org/10.1145/2980983.2908118
[2] https://github.com/wilcoxjay/mypyvy
[3] Leslie Lamport. 2002. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., USA.
[4] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon web services uses formal methods. Commun. ACM 58, 4 (April 2015), 66–73. https://doi.org/10.1145/2699417

Distributed Protocol Verification

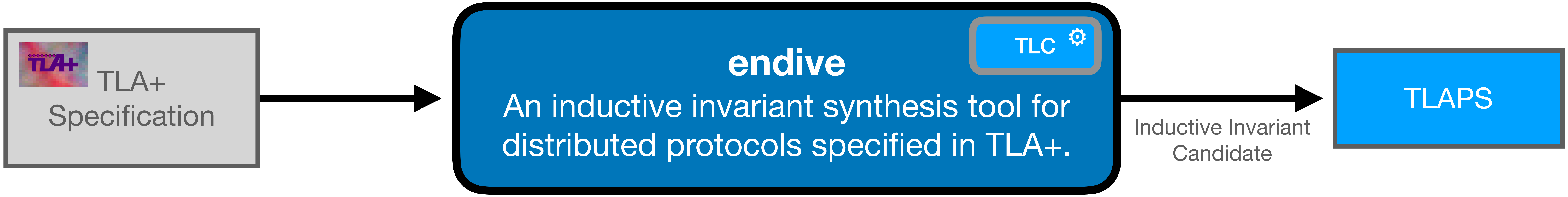
- Bounded verification e.g. explicit state, bounded model checking may suffice for initial design verification.
- Unbounded safety verification requires development of ***inductive invariant***.
- Finding inductive invariants manually is difficult.
- Tools for inductive invariant inference have been developed for Ivy [1], mypyvy [2], etc.
- No existing approaches for ***TLA+*** [3], a specification language used widely in industry [4].

[1] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: safety verification by interactive generalization. SIGPLAN Not. 51, 6 (June 2016), 614–630. https://doi.org/10.1145/2980983.2908118
[2] https://github.com/wilcoxjay/mypyvy
[3] Leslie Lamport. 2002. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., USA.
[4] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon web services uses formal methods. Commun. ACM 58, 4 (April 2015), 66–73. https://doi.org/10.1145/2699417

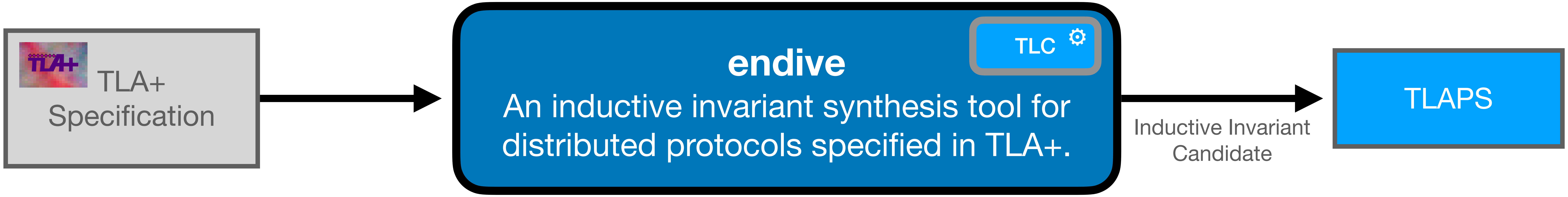
Our Contributions

endive
An inductive invariant synthesis tool for distributed protocols specified in TLA+.

Our Contributions



Our Contributions



- Only existing tool that infers inductive invariants for distributed protocols specified in TLA+.
- Competitive with other state of the art invariant inference tools on diverse benchmark.
- Uniquely solves industrial scale, Raft-based reconfiguration protocol.

Outline

Preliminaries: Safety Verification

Our Approach

Evaluation

Conclusion

Outline

Preliminaries: Safety Verification

Our Approach

Evaluation

Conclusion

Safety Verification of Transition Systems

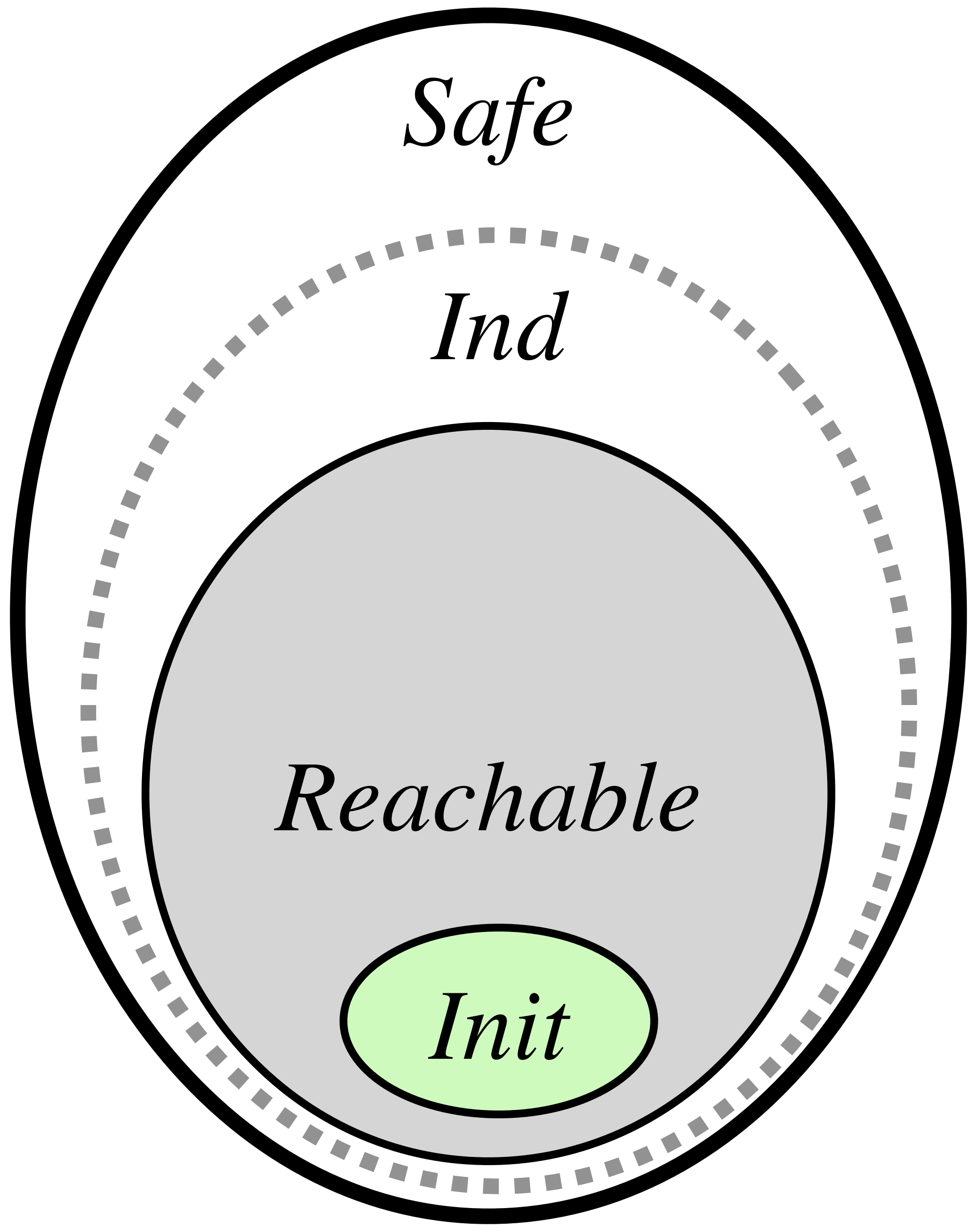
Let $M = (Init, Next)$ be a transition system.

Safety Verification of Transition Systems

Let $M = (Init, Next)$ be a transition system.

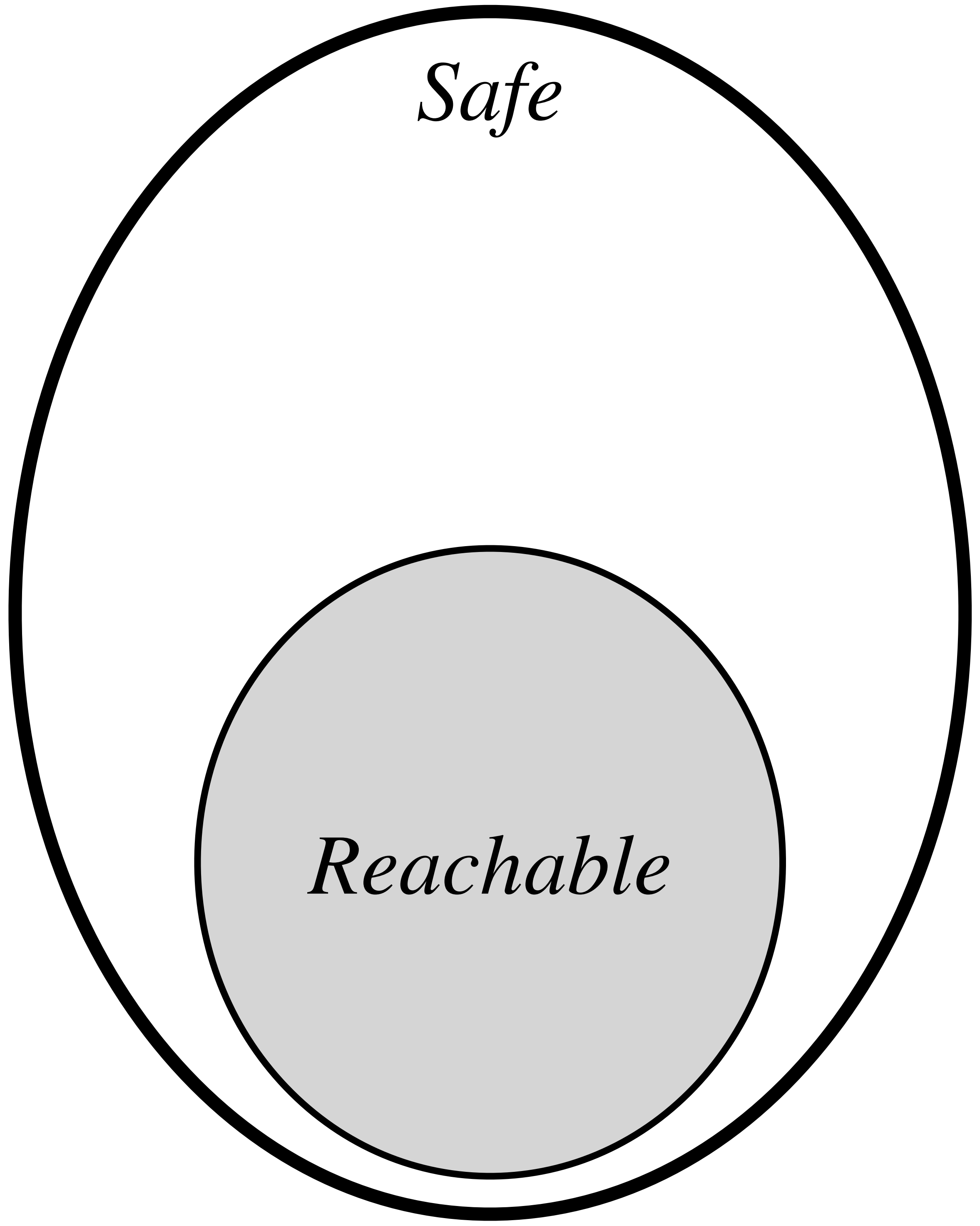
To verify that a predicate $Safe$ is an invariant of M, find an ***inductive invariant***, Ind, satisfying

$Init \Rightarrow Ind$	(initiation)
$Ind \wedge Next \Rightarrow Ind'$	(consecution)
$Ind \Rightarrow Safe$	



Incremental Inductive Invariant Inference

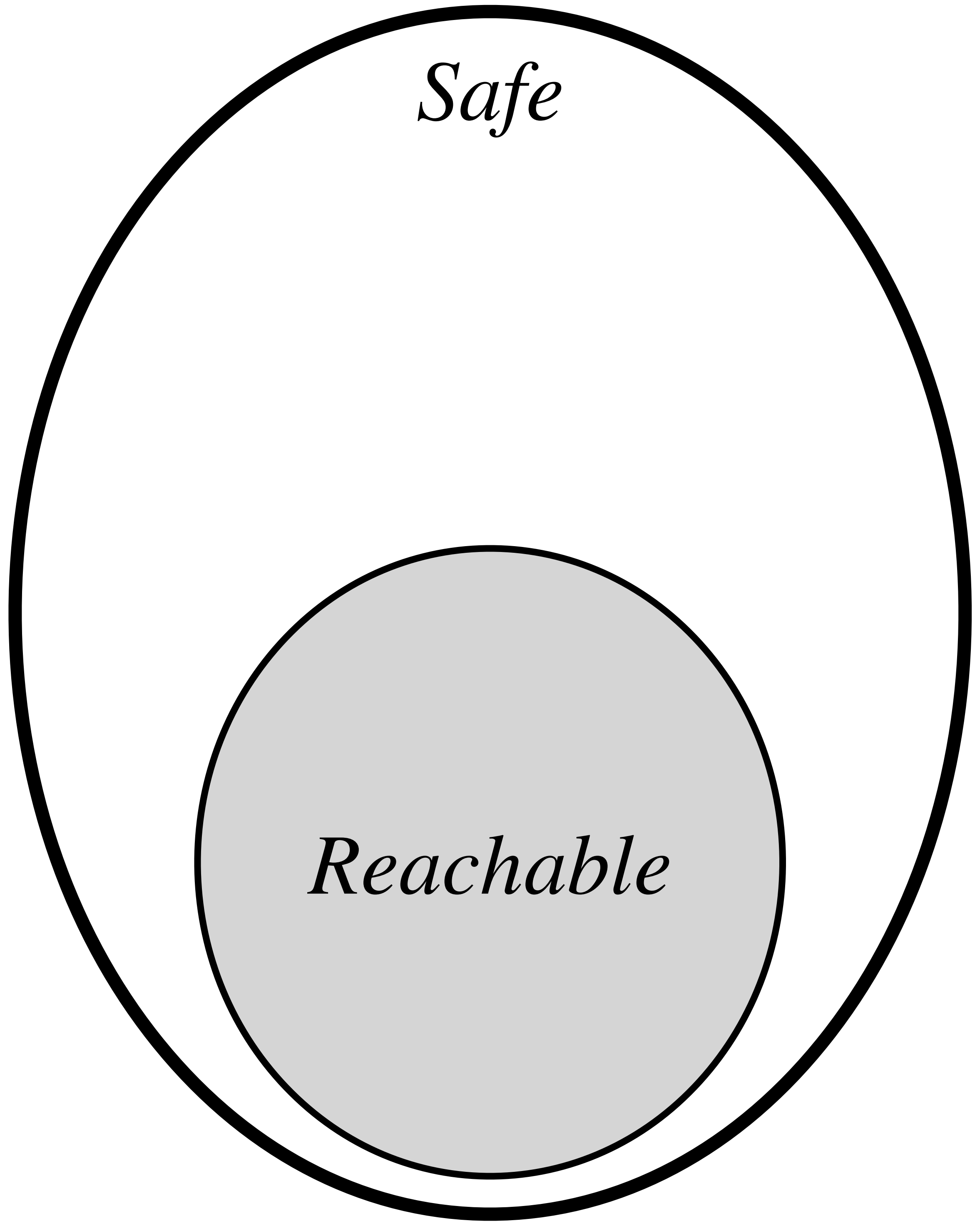
Standard view of inductive invariant inference (e.g. IC3/PDR): an *incremental* lemma synthesis problem



Incremental Inductive Invariant Inference

Standard view of inductive invariant inference (e.g. IC3/PDR): an *incremental* lemma synthesis problem

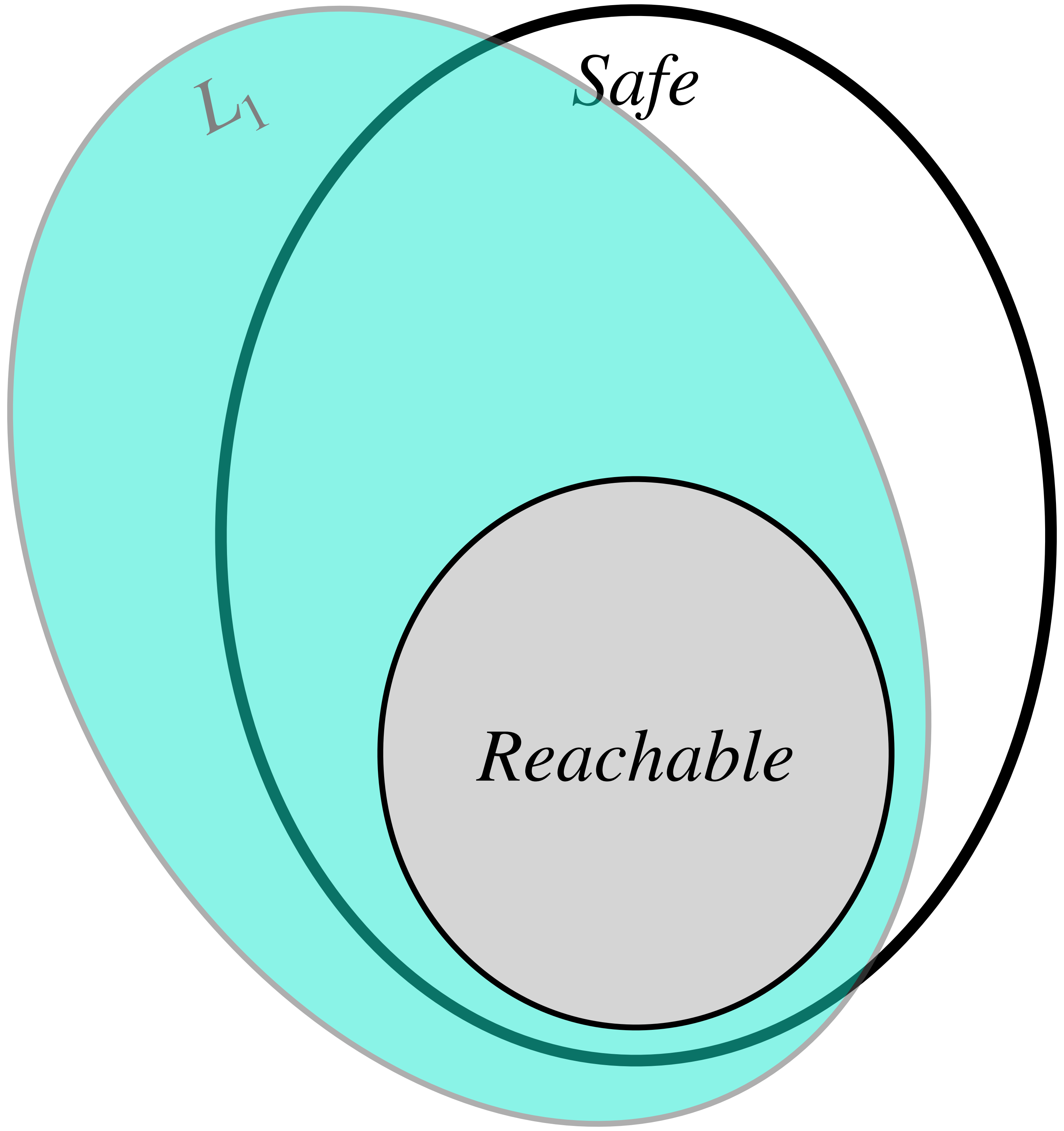
$$Ind \triangleq Safe$$



Incremental Inductive Invariant Inference

Standard view of inductive invariant inference (e.g. IC3/PDR): an *incremental* lemma synthesis problem

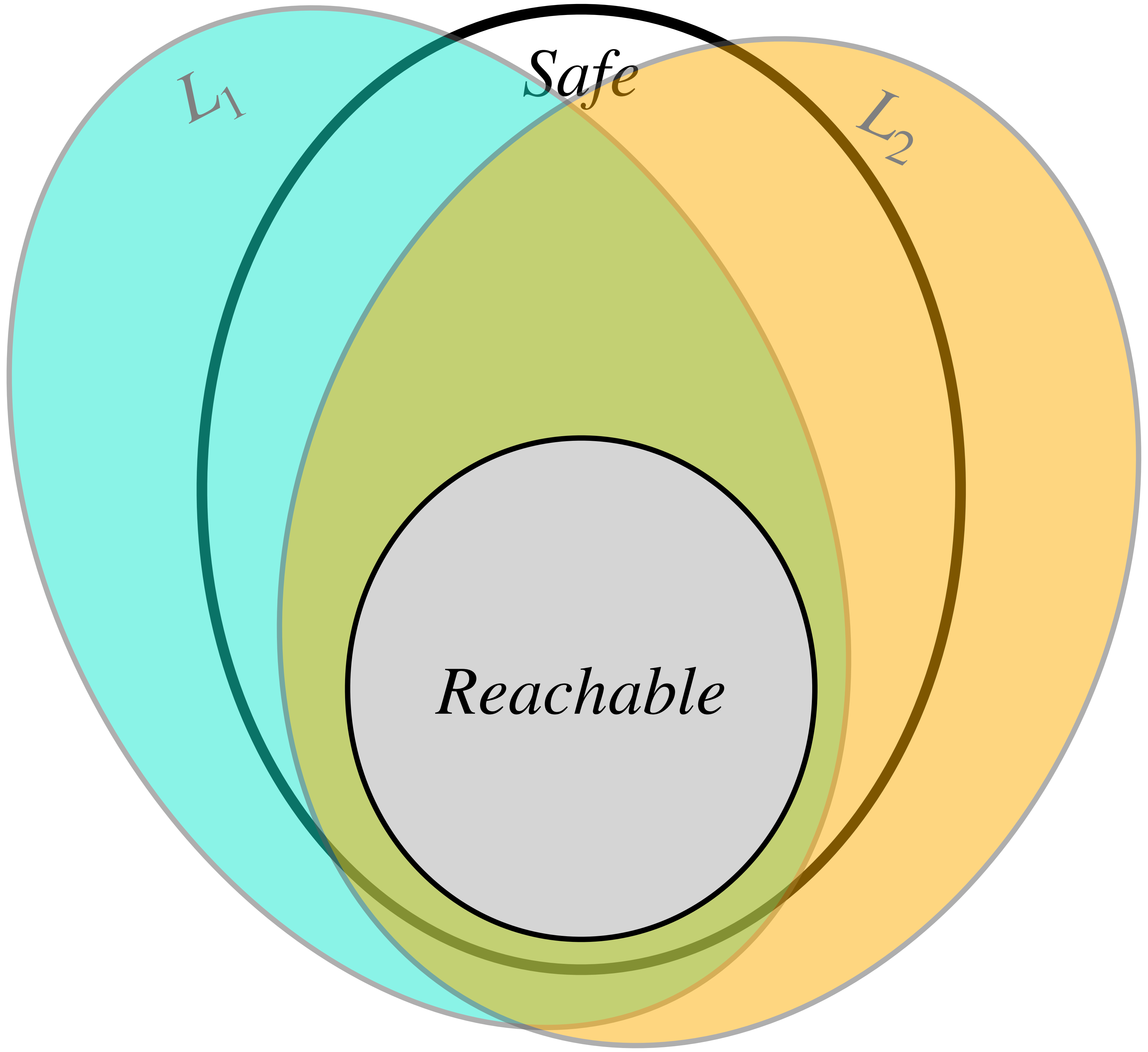
$$
\begin{aligned}
Ind \triangleq\ & Safe \\
& \land L_1
\end{aligned}
$$



Incremental Inductive Invariant Inference

Standard view of inductive invariant inference (e.g. IC3/PDR): an *incremental* lemma synthesis problem

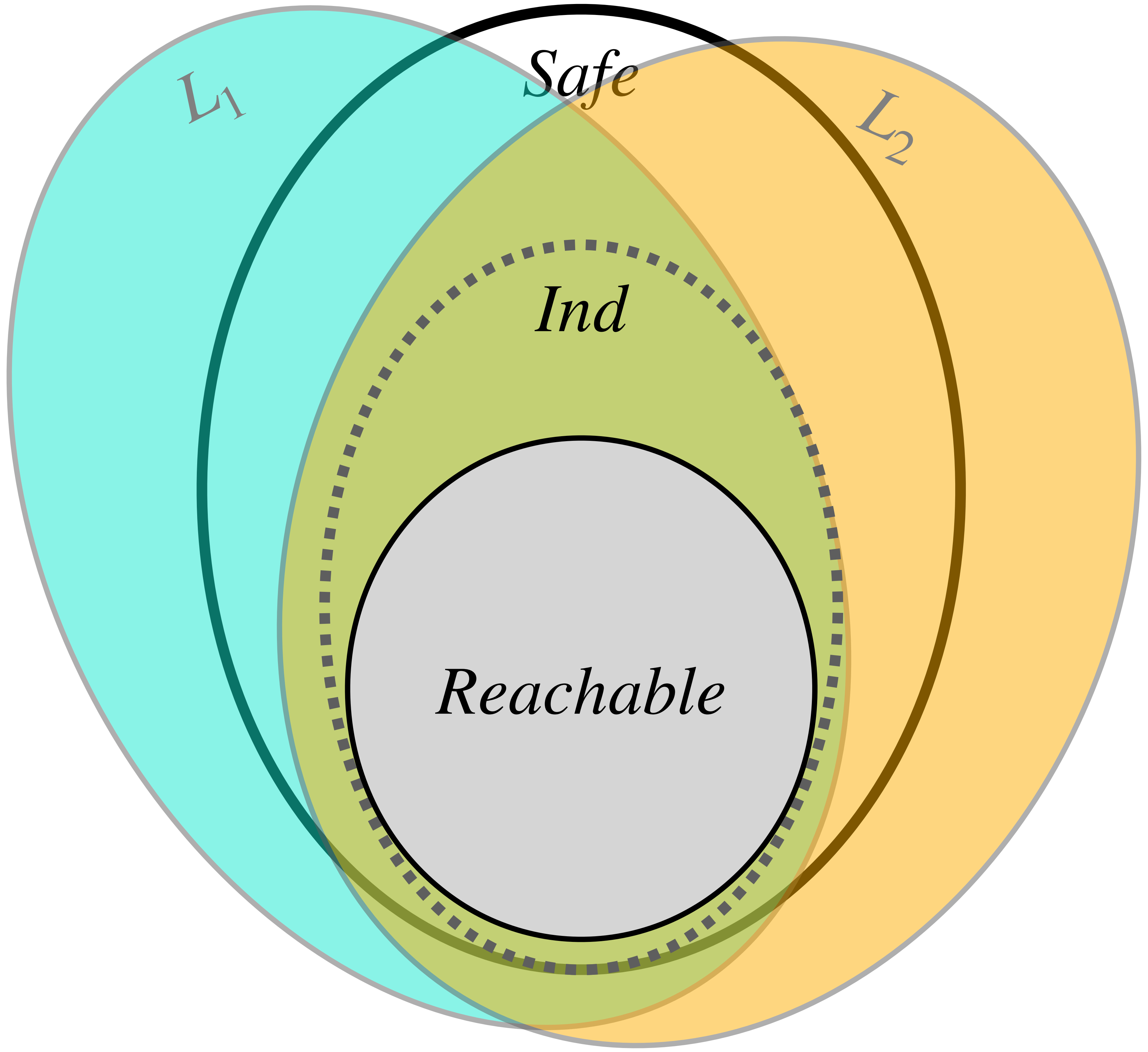
$$
\begin{aligned}
Ind \triangleq\ & Safe \\
& \wedge L_1 \\
& \wedge L_2
\end{aligned}
$$



Incremental Inductive Invariant Inference

Standard view of inductive invariant inference (e.g. IC3/PDR): an *incremental* lemma synthesis problem

$$
\begin{aligned}
Ind \triangleq\ & Safe \\
& \wedge L_1 \\
& \wedge L_2 \\
& \vdots \\
& \wedge L_n
\end{aligned}
$$

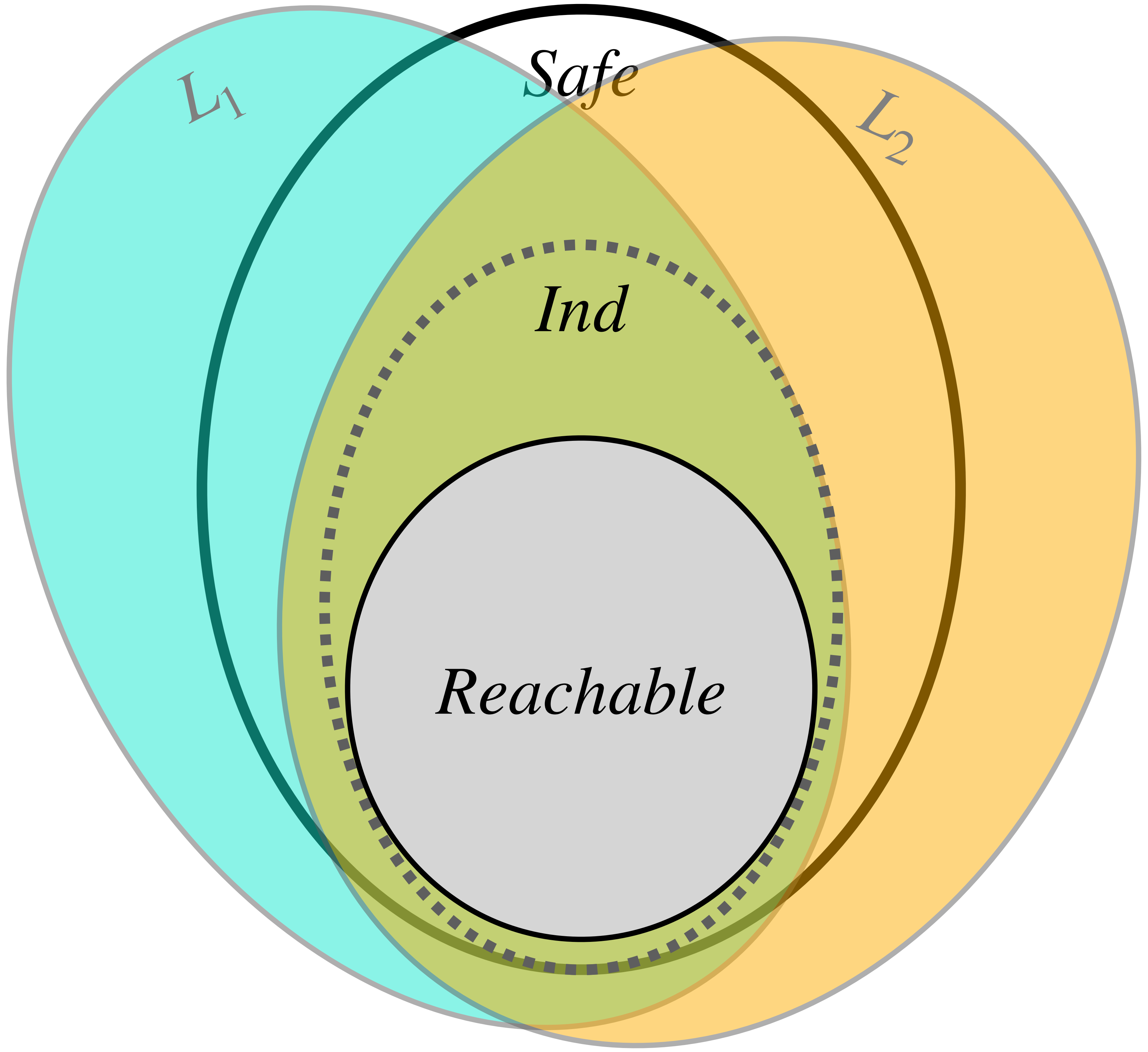


Incremental Inductive Invariant Inference

Standard view of inductive invariant inference (e.g. IC3/PDR): an *incremental* lemma synthesis problem

$$
\begin{aligned}
Ind \triangleq\ & Safe \\
& \land L_1 \\
& \land L_2 \\
& \vdots \\
& \land L_n
\end{aligned}
$$

Lemma Invariants



Note that $L_1, \ldots, L_n$ are, individually, invariants (not necessarily inductive).

Outline

Preliminaries: Safety Verification

Our Approach

Evaluation

Conclusion

Our Approach

Our Approach

Our overall inductive invariant inference technique consists of 2 components:

1. **Lemma Invariant Generation**: Use (plain) invariant synthesis engine to generate candidate lemma invariants.

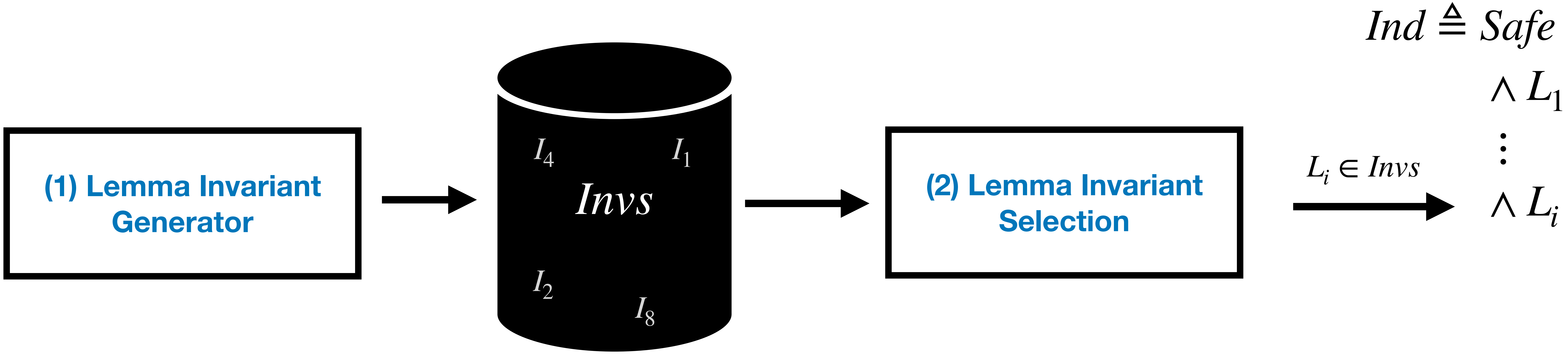
2. **Lemma Invariant Selection**: Select lemma invariants to greedily eliminate sets of *counterexamples to induction* (CTIs).

Our Approach

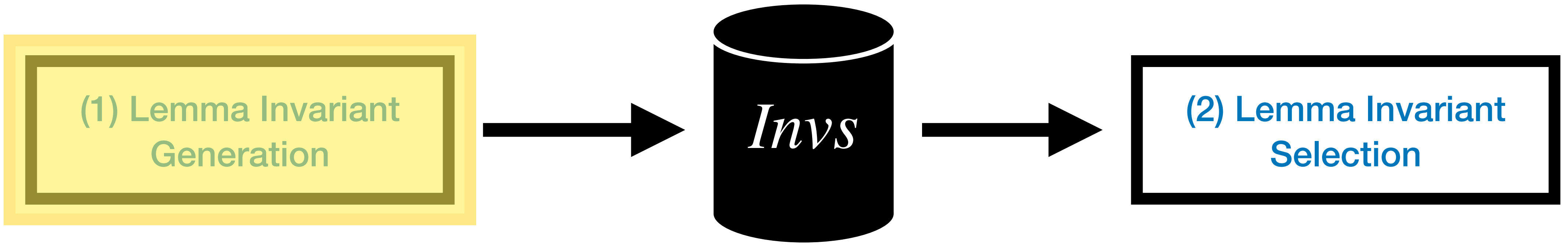
Our overall inductive invariant inference technique consists of 2 components:

1. **Lemma Invariant Generation**: Use (plain) invariant synthesis engine to generate candidate lemma invariants.

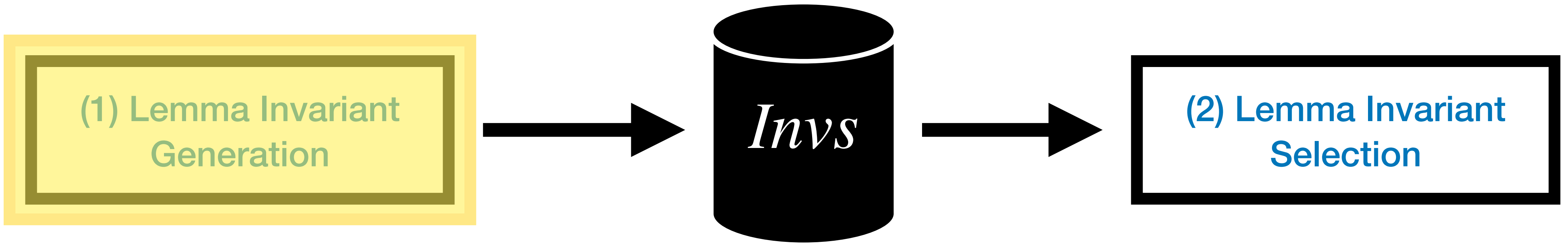
2. **Lemma Invariant Selection**: Select lemma invariants to greedily eliminate sets of *counterexamples to induction* (CTIs).



Lemma Invariant Generation

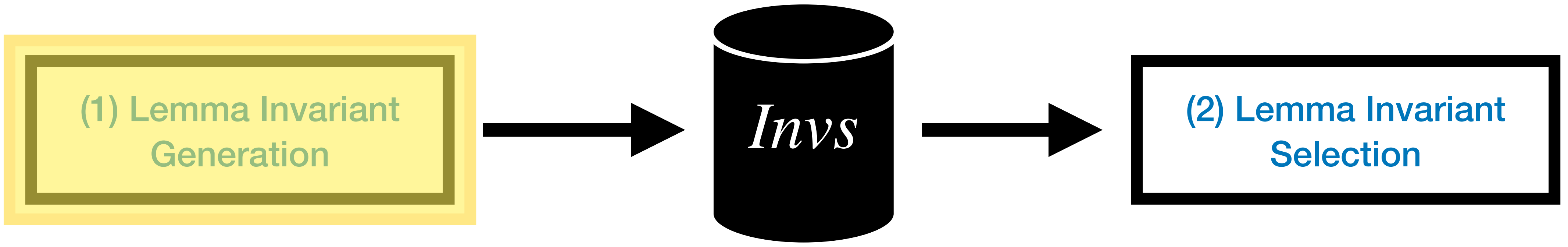


Lemma Invariant Generation



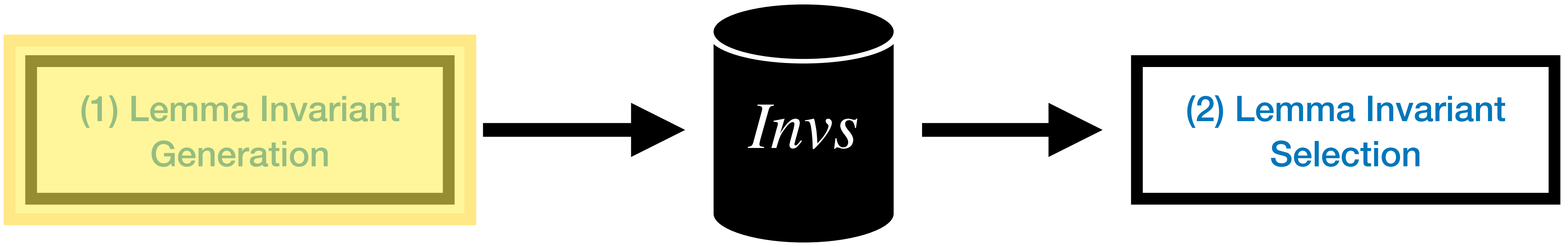
- Syntax guided, sampling based approach
  - TLA+ specification $M = (Init, Next)$ finite instance size ("*small scope hypothesis*")
  - Grammar G

Lemma Invariant Generation



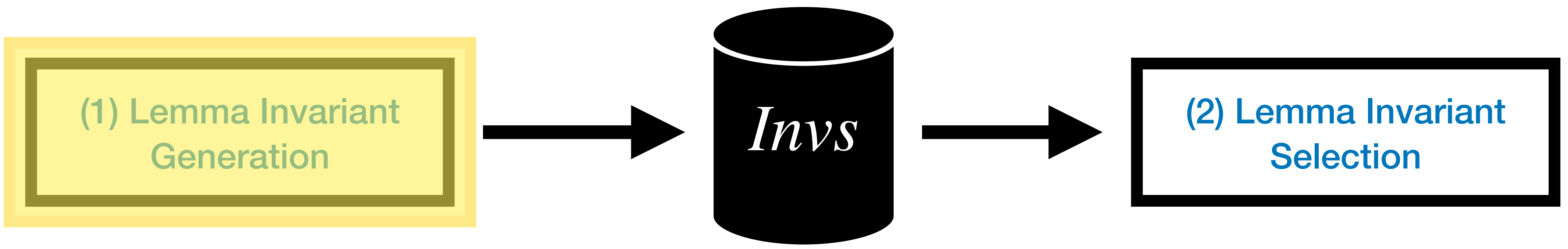
- Syntax guided, sampling based approach
    - TLA+ specification $M = (Init, Next)$ finite instance size (“*small scope hypothesis*”)
    - Grammar G
- Randomly sample candidates generated by G

Lemma Invariant Generation



- Syntax guided, sampling based approach
  - TLA+ specification $M = (Init, Next)$ finite instance size (“*small scope hypothesis*”)
  - Grammar G
- Randomly sample candidates generated by G
- Check candidates exhaustively on M with TLC model checker

Lemma Invariant Generation

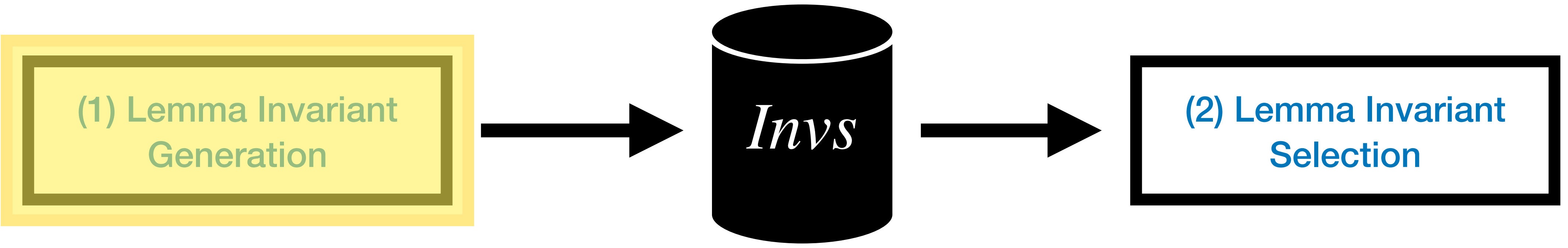


- Syntax guided, sampling based approach
  - TLA+ specification $M = (Init, Next)$ finite instance size ("*small scope hypothesis*")
  - Grammar G
- Randomly sample candidates generated by G
- Check candidates exhaustively on M with TLC model checker

$$
\begin{aligned}
\langle seed \rangle &::= \; locked[s] \mid s \in held[c] \mid held[c] = \emptyset \\
\langle quant \rangle &::= \forall s \in Server : \forall c \in Client \\
\langle expr \rangle &::= \langle seed \rangle \mid \neg \langle expr \rangle \mid \langle expr \rangle \vee \langle expr \rangle \\
\langle pred \rangle &::= \langle quant \rangle : \langle expr \rangle
\end{aligned}
$$

Example invariant grammar for *lockserver.*

Lemma Invariant Generation



- Syntax guided, sampling based approach
  - TLA+ specification $M = (Init, Next)$ finite instance size (“*small scope hypothesis*”)
  - Grammar G
- Randomly sample candidates generated by G
- Check candidates exhaustively on M with TLC model checker

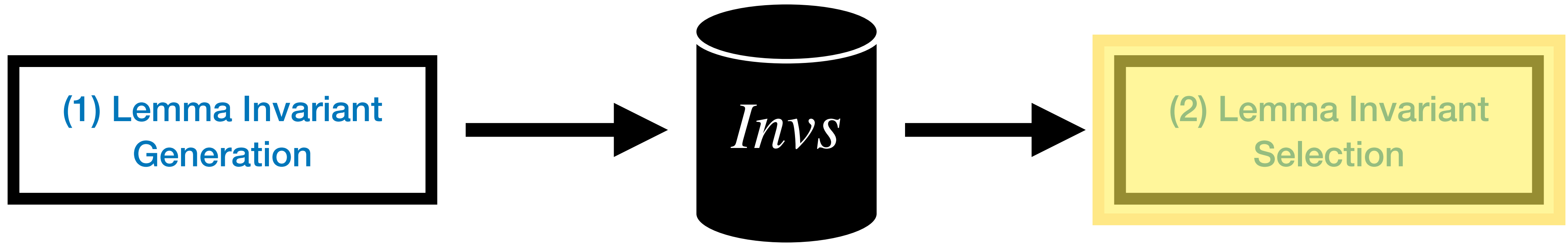
$$
\begin{aligned}
\langle seed \rangle &::= \ locked[s] \mid s \in held[c] \mid held[c] = \emptyset \\
\langle quant \rangle &::= \forall s \in Server : \forall c \in Client \\
\langle expr \rangle &::= \langle seed \rangle \mid \neg \langle expr \rangle \mid \langle expr \rangle \vee \langle expr \rangle \\
\langle pred \rangle &::= \langle quant \rangle : \langle expr \rangle
\end{aligned}
$$

Atomic predicates

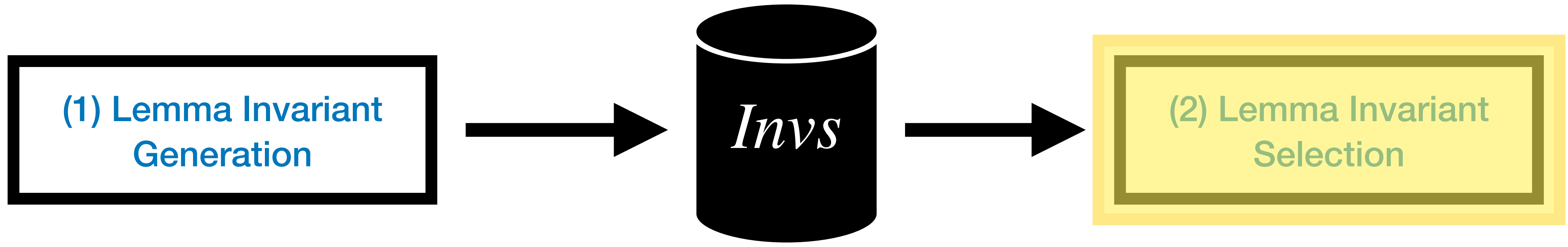
Quantifier template

Example invariant grammar for *lockserver.*

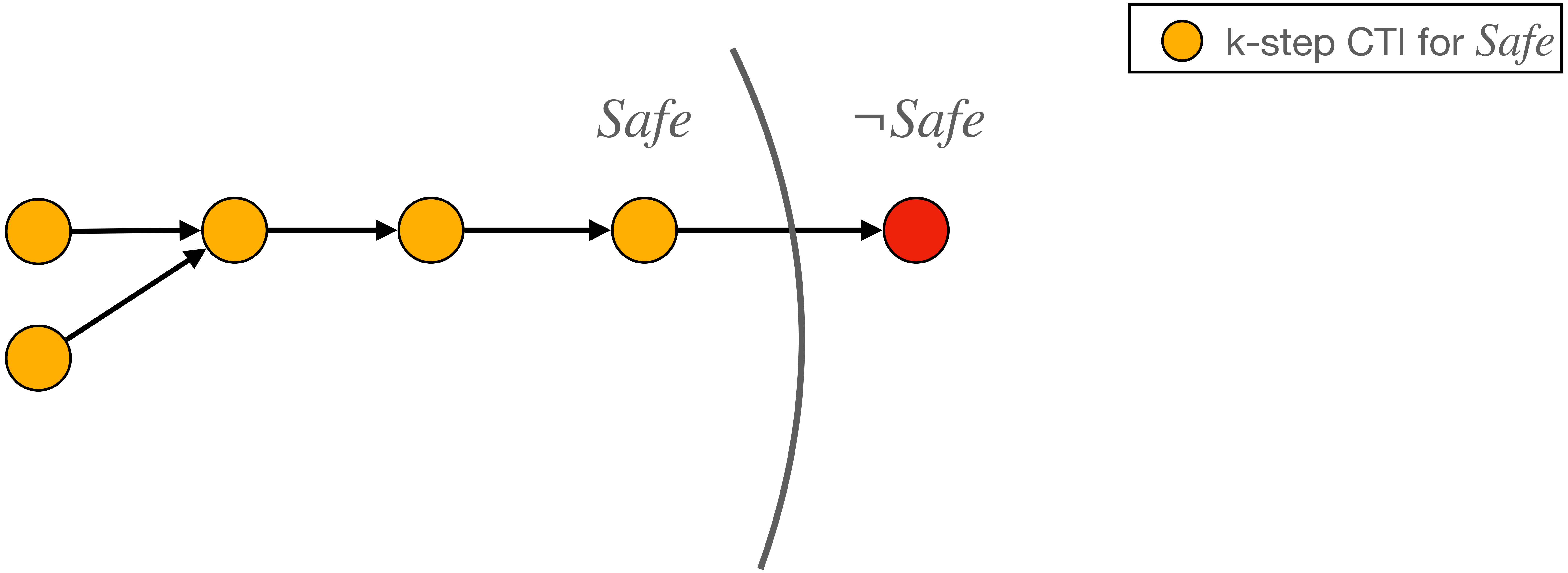
Lemma Invariant Selection



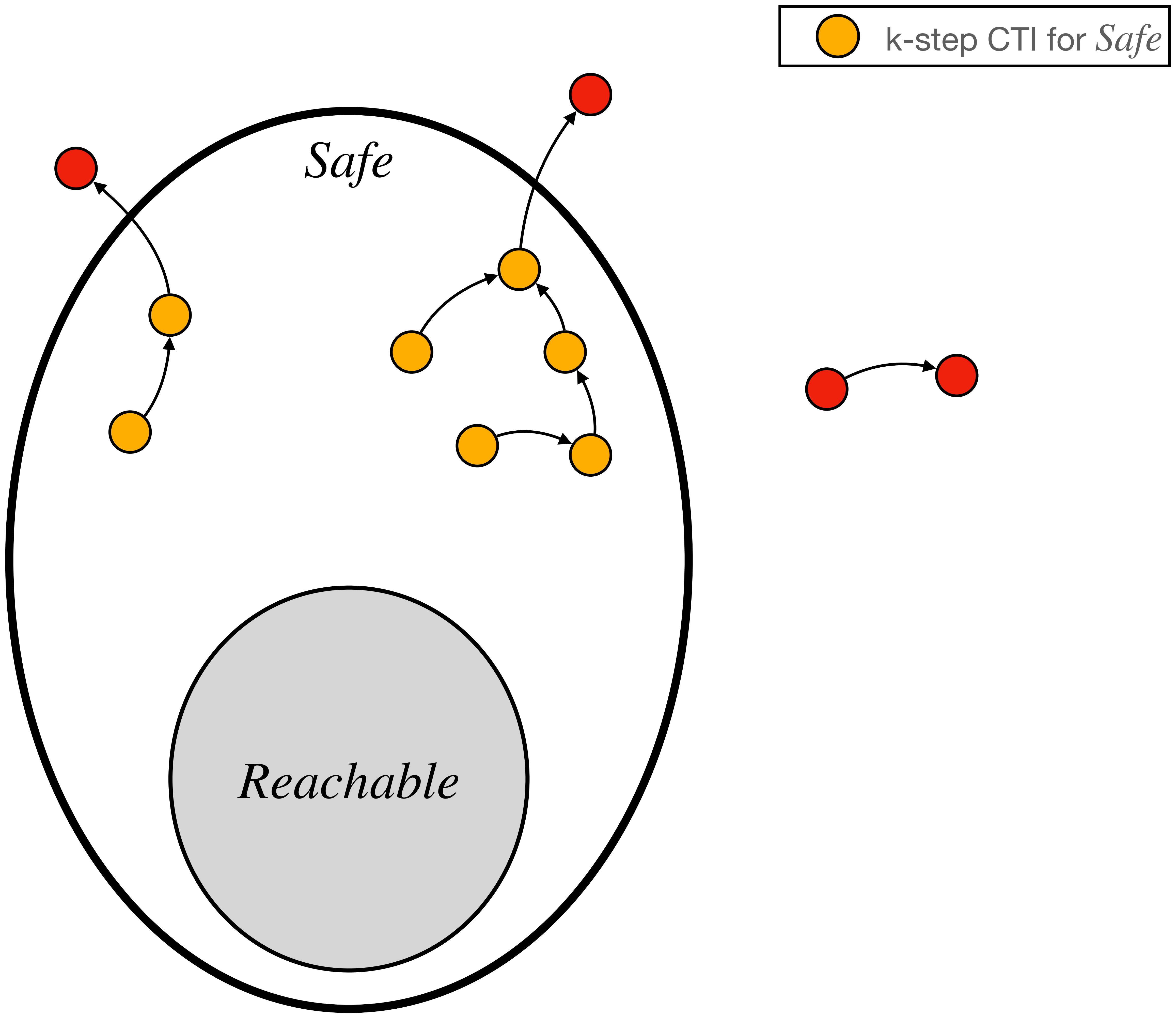
Lemma Invariant Selection



Standard approach is to find lemmas by generalizing from individual *counterexamples to induction* (CTIs)

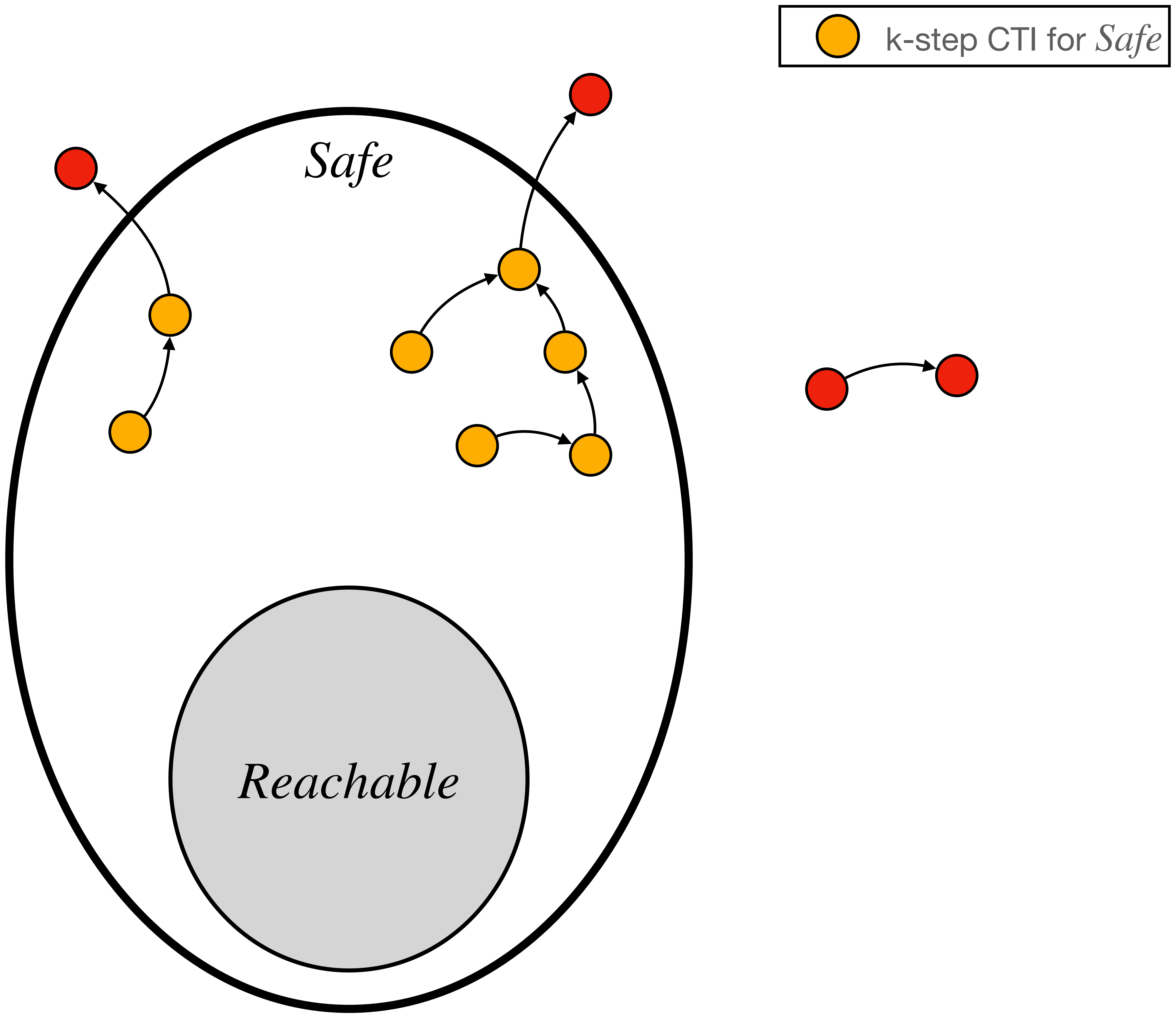


Lemma Invariant Selection



Lemma Invariant Selection

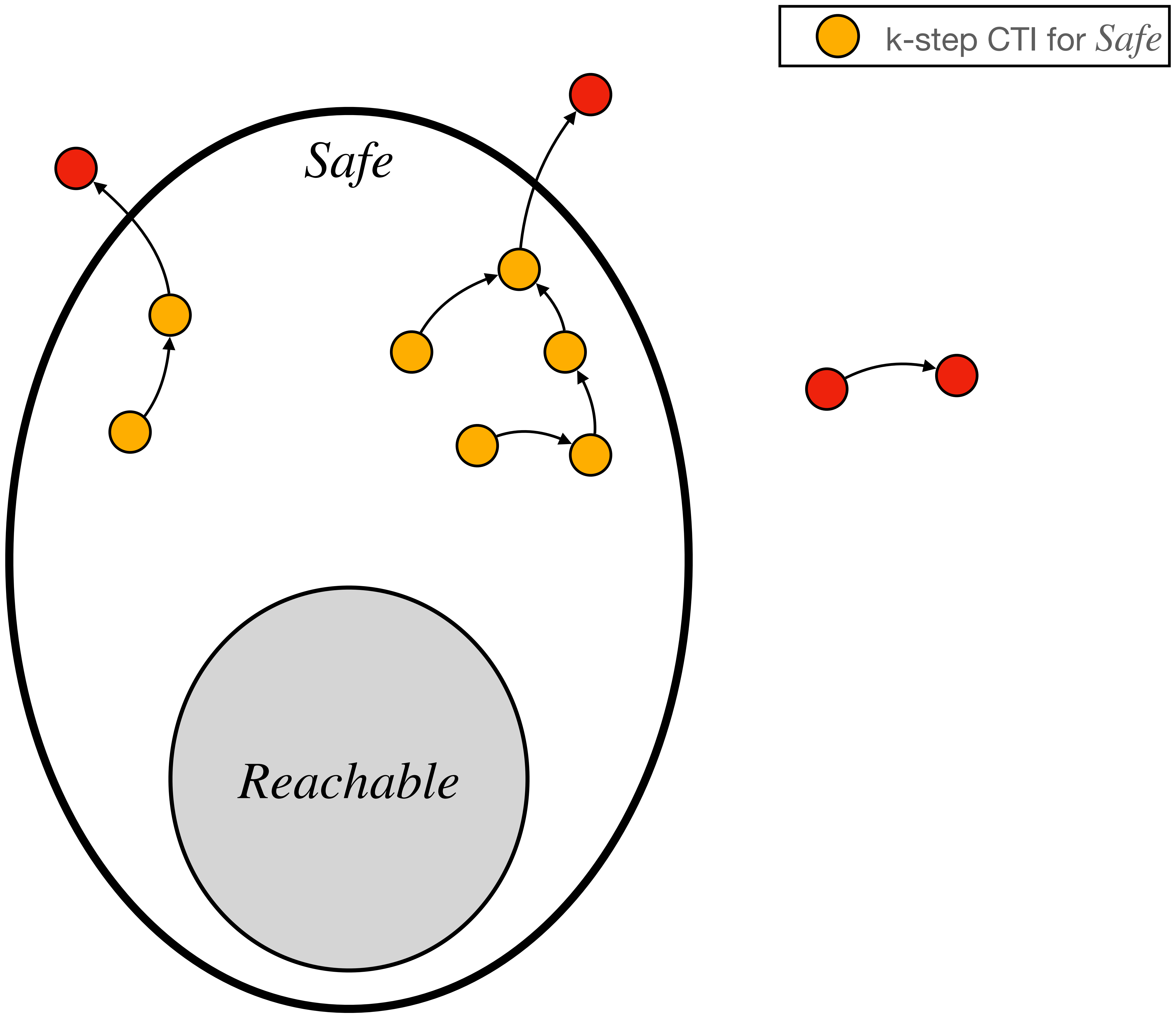
Our approach takes a data-driven view, with goals of learning *concise* invariants



Lemma Invariant Selection

Our approach takes a data-driven view, with goals of learning *concise* invariants

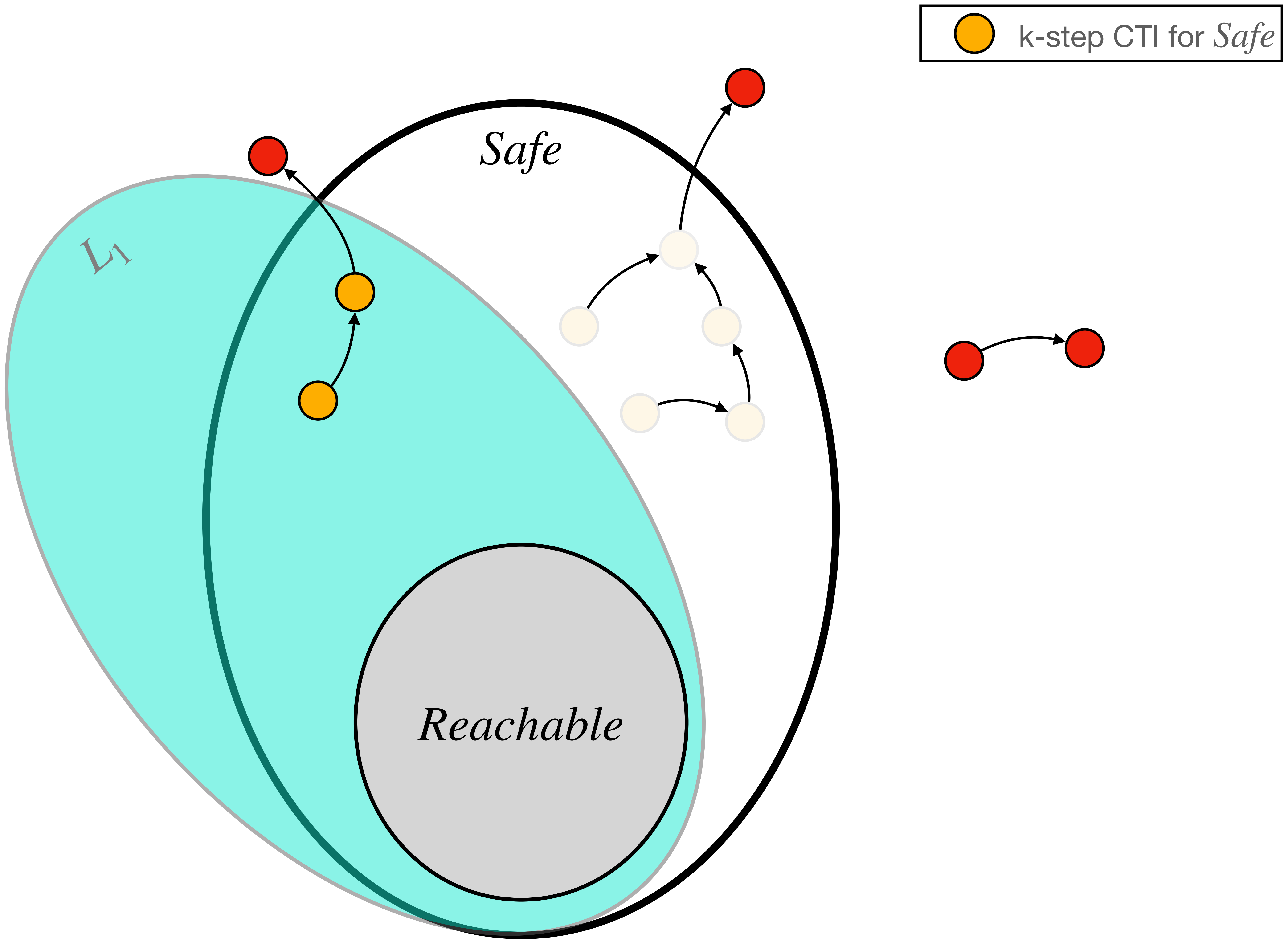
Generate many CTIs, then choose next lemma greedily i.e. one that eliminates greatest number of remaining CTIs.



Lemma Invariant Selection

Our approach takes a data-driven view, with goals of learning *concise* invariants

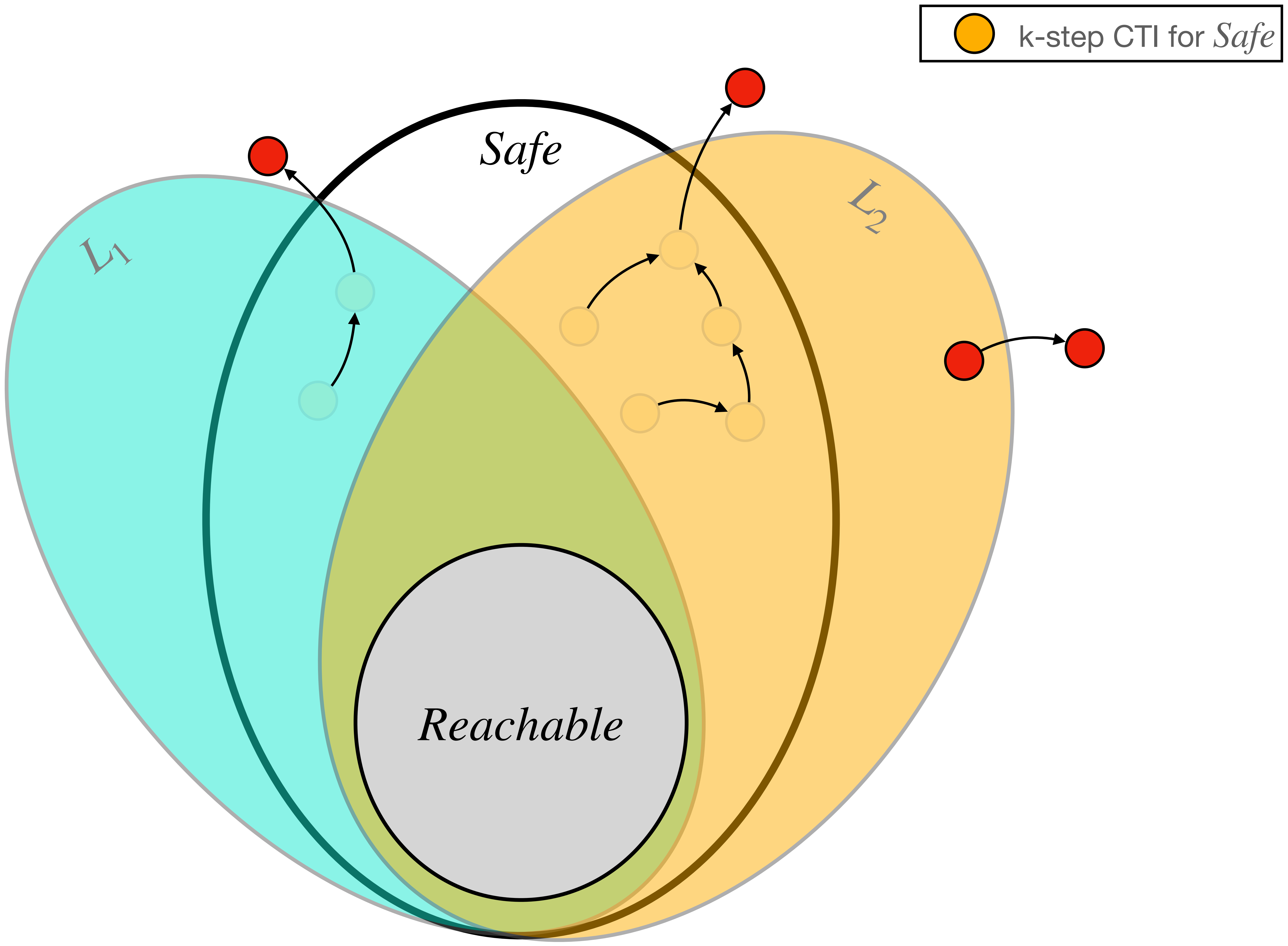
Generate many CTIs, then choose next lemma greedily i.e. one that eliminates greatest number of remaining CTIs.



Lemma Invariant Selection

Our approach takes a data-driven view, with goals of learning *concise* invariants

Generate many CTIs, then choose next lemma greedily i.e. one that eliminates greatest number of remaining CTIs.



Outline

Preliminaries: Safety Verification

Our Approach

Evaluation

Conclusion

Evaluation

Evaluation

- Compared our tool, **endive**, against 4 other state of the art invariant inference tools
  - IC3PO [1], fol-ic3 [2], SWISS [3], DistAI [4]

[1] Goel, Aman & Sakallah, Karem. (2021). On Symmetry and Quantification: A New Approach to Verify Distributed Protocols. 10.1007/978-3-030-76384-8_9.
[2] Jason R. Koenig, et al. 2020. First-order quantified separators. *In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 703–717. https://doi.org/10.1145/3385412.3386018
[3] Travis Hance, Marijn Heule, Ruben Martins, Bryan Parno, Finding Invariants of Distributed Systems: It's a Small (Enough) World After All. *NSDI 2021*: 115-131
[4] Jianan Yao, Runzhou Tao, et al. DistAI: Data-Driven Automated Invariant Learning for Distributed Protocols. *OSDI 2021*: 485-501

Evaluation

- Compared our tool, **endive**, against 4 other state of the art invariant inference tools
    - IC3PO [1], fol-ic3 [2], SWISS [3], DistAI [4]
- Other tools accept Ivy/mypyvy so translation to TLA+ was necessary.

[1] Goel, Aman & Sakallah, Karem. (2021). On Symmetry and Quantification: A New Approach to Verify Distributed Protocols. 10.1007/978-3-030-76384-8_9.
[2] Jason R. Koenig, et al. 2020. First-order quantified separators. *In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 703–717. https://doi.org/10.1145/3385412.3386018
[3] Travis Hance, Marijn Heule, Ruben Martins, Bryan Parno, Finding Invariants of Distributed Systems: It's a Small (Enough) World After All. *NSDI 2021*: 115-131
[4] Jianan Yao, Runzhou Tao, et al. DistAI: Data-Driven Automated Invariant Learning for Distributed Protocols. *OSDI 2021*: 485-501

Evaluation

- Compared our tool, **endive**, against 4 other state of the art invariant inference tools
  - IC3PO [1], fol-ic3 [2], SWISS [3], DistAI [4]
- Other tools accept Ivy/mypyvy so translation to TLA+ was necessary.
- Benchmark consists of
  - 29 concurrent/distributed protocols of varying complexity (e.g. 2PC, simple consensus)
  - 1 industrial scale Raft-based reconfiguration protocol.

[1] Goel, Aman & Sakallah, Karem. (2021). On Symmetry and Quantification: A New Approach to Verify Distributed Protocols. 10.1007/978-3-030-76384-8_9.
[2] Jason R. Koenig, et al. 2020. First-order quantified separators. *In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 703–717. https://doi.org/10.1145/3385412.3386018
[3] Travis Hance, Marijn Heule, Ruben Martins, Bryan Parno, Finding Invariants of Distributed Systems: It's a Small (Enough) World After All. *NSDI 2021*: 115-131
[4] Jianan Yao, Runzhou Tao, et al. DistAI: Data-Driven Automated Invariant Learning for Distributed Protocols. *OSDI 2021*: 485-501

Evaluation

Solved Benchmarks

endive able to solve 26 / 30 of the benchmarks successfully.

	Protocol	endive	IC3PO	fol-ic3	SWISS	DistAI
1	tla-consensus	✅	✅	✅	✅	✅
2	tla-tcommit	✅	✅	✅	✅	✅
3	i4-lock-server	✅	✅	✅	✅	❌
4	ex-quorum-leader-election	✅	✅	✅	✅	✅
5	pyv-toy-consensus-forall	✅	✅	✅	✅	❌
6	tla-simple	✅	✅	❌	✅	❌
7	ex-lockserv-automaton	✅	✅	✅	❌	✅
8	tla-simpleregular	✅	✅	✅	✅	❌
9	pyv-sharded-kv	✅	✅	✅	✅	✅
10	pyv-lockserv	✅	✅	✅	✅	✅
11	tla-twophase	✅	✅	✅	✅	✅
12	i4-learning-switch	❌	✅	❌	❌	✅
13	ex-simple-decentralized-lock	✅	✅	✅	✅	✅
14	i4-two-phase-commit	✅	✅	✅	✅	✅
15	pyv-consensus-wo-decide	✅	✅	✅	✅	❌
16	pyv-consensus-forall	✅	✅	✅	✅	❌
17	pyv-learning-switch	❌	✅	❌	✅	✅
18	i4-chord-ring-maintenance	❌	✅	❌	❌	❌
19	pyv-sharded-kv-no-lost-keys	✅	✅	✅	✅	❌
20	ex-naive-consensus	✅	✅	✅	✅	❌
21	pyv-client-server-ae	✅	✅	✅	✅	❌
22	ex-simple-election	✅	✅	✅	✅	❌
23	pyv-toy-consensus-epr	✅	✅	✅	✅	❌
24	ex-toy-consensus	✅	✅	✅	✅	❌
25	pyv-client-server-db-ae	✅	✅	❌	✅	❌
26	pyv-hybrid-reliable-broadcast	❌	✅	✅	❌	❌
27	pyv-firewall	✅	✅	✅	✅	❌
28	ex-majorityset-leader-election	✅	✅	❌	✅	❌
29	pyv-consensus-epr	✅	✅	✅	✅	❌
30	mldr	✅	❌	❌	❌	❌

Evaluation

Solved Benchmarks

endive able to solve 26 / 30 of the benchmarks successfully.

Uniquely solved *mldr, an* industrial scale Raft-based reconfiguration protocol.

	Protocol	endive	IC3PO	fol-ic3	SWISS	DistAI
1	tla-consensus	✅	✅	✅	✅	✅
2	tla-tcommit	✅	✅	✅	✅	✅
3	i4-lock-server	✅	✅	✅	✅	❌
4	ex-quorum-leader-election	✅	✅	✅	✅	✅
5	pyv-toy-consensus-forall	✅	✅	✅	✅	❌
6	tla-simple	✅	✅	❌	✅	❌
7	ex-lockserv-automaton	✅	✅	✅	❌	✅
8	tla-simpleregular	✅	✅	✅	✅	❌
9	pyv-sharded-kv	✅	✅	✅	✅	✅
10	pyv-lockserv	✅	✅	✅	✅	✅
11	tla-twophase	✅	✅	✅	✅	✅
12	i4-learning-switch	❌	✅	❌	❌	✅
13	ex-simple-decentralized-lock	✅	✅	✅	✅	✅
14	i4-two-phase-commit	✅	✅	✅	✅	✅
15	pyv-consensus-wo-decide	✅	✅	✅	✅	❌
16	pyv-consensus-forall	✅	✅	✅	✅	❌
17	pyv-learning-switch	❌	✅	❌	✅	✅
18	i4-chord-ring-maintenance	❌	✅	❌	❌	❌
19	pyv-sharded-kv-no-lost-keys	✅	✅	✅	✅	❌
20	ex-naive-consensus	✅	✅	✅	✅	❌
21	pyv-client-server-ae	✅	✅	✅	✅	❌
22	ex-simple-election	✅	✅	✅	✅	❌
23	pyv-toy-consensus-epr	✅	✅	✅	✅	❌
24	ex-toy-consensus	✅	✅	✅	✅	❌
25	pyv-client-server-db-ae	✅	✅	❌	✅	❌
26	pyv-hybrid-reliable-broadcast	❌	✅	✅	❌	❌
27	pyv-firewall	✅	✅	✅	✅	❌
28	ex-majorityset-leader-election	✅	✅	❌	✅	❌
29	pyv-consensus-epr	✅	✅	✅	✅	❌
30	mldr	✅	❌	❌	❌	❌

Evaluation

Relative Invariant Size Comparison

$$Ind \triangleq Safe \land L_1 \land \ldots \land L_n$$

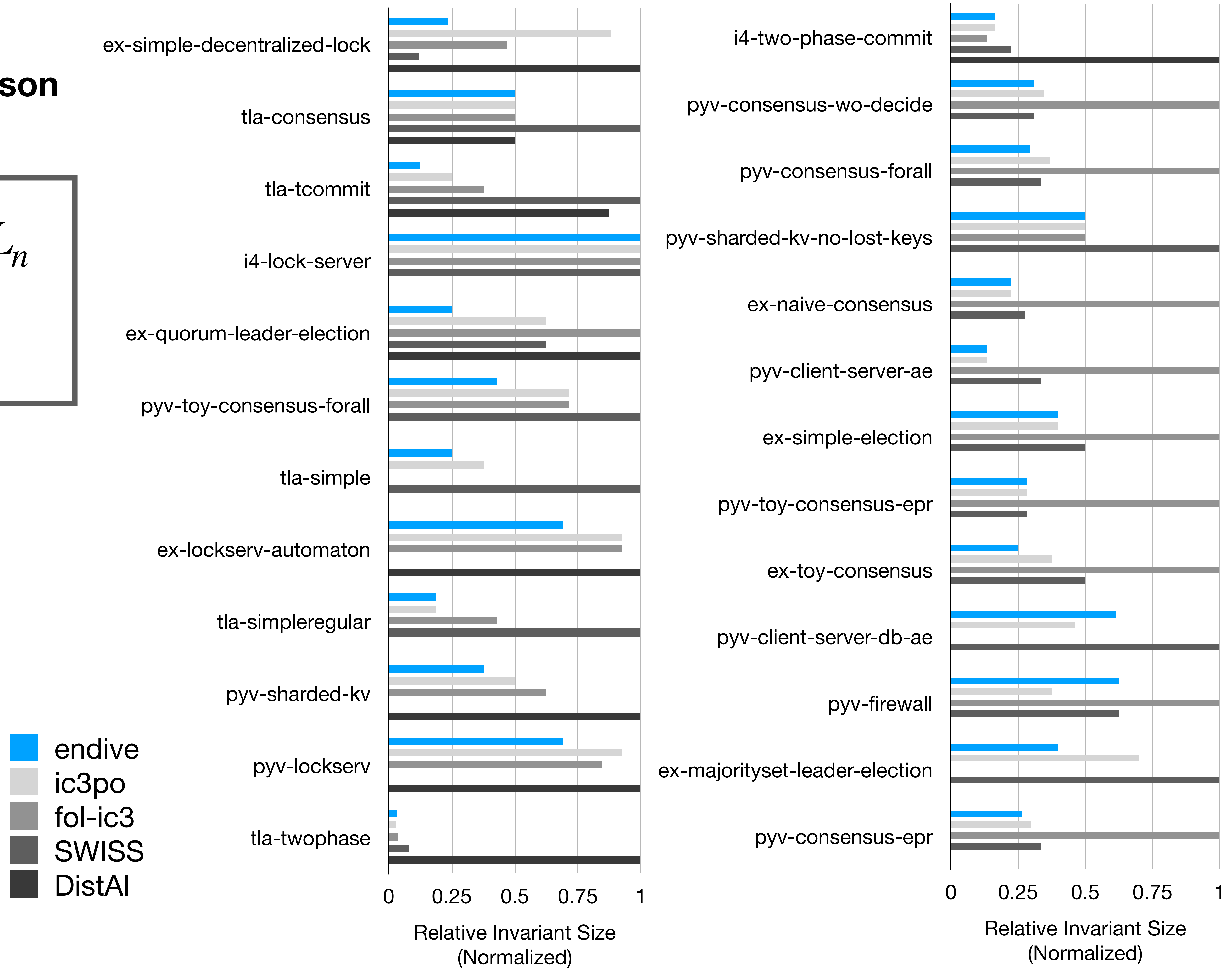
size measured as $(n + 1)$

Evaluation

Relative Invariant Size Comparison

$$Ind \triangleq Safe \wedge L_1 \wedge \ldots \wedge L_n$$

size measured as $(n + 1)$



Conclusion

Conclusion

- First technique for inferring inductive invariants for distributed protocols specified in TLA+.

Conclusion

- First technique for inferring inductive invariants for distributed protocols specified in TLA+.
- Competitive with other state of the art invariant inference tools.

Conclusion

- First technique for inferring inductive invariants for distributed protocols specified in TLA+.
- Competitive with other state of the art invariant inference tools.
- Uniquely solves industrial scale, Raft-based reconfiguration protocol.

Conclusion

- First technique for inferring inductive invariants for distributed protocols specified in TLA+.
- Competitive with other state of the art invariant inference tools.
- Uniquely solves industrial scale, Raft-based reconfiguration protocol.

Simple, greedy approach works surprisingly well for this class of protocols.

Conclusion

- First technique for inferring inductive invariants for distributed protocols specified in TLA+.
- Competitive with other state of the art invariant inference tools.
- Uniquely solves industrial scale, Raft-based reconfiguration protocol.

Simple, greedy approach works surprisingly well for this class of protocols.

Try out endive:

https://github.com/will62794/endive

Evaluation

TLAPS Proof Burden

Evaluation

TLAPS Proof Burden

Proving consecution typically most burdensome.

$$Ind \land Next \Rightarrow Ind'$$

Evaluation

TLAPS Proof Burden

Proving consecution typically most burdensome.

$$Ind \land Next \Rightarrow Ind'$$

If we have

$$Next \triangleq T_1 \lor \ldots \lor T_k$$

$$Ind \triangleq L_1 \land \ldots \land L_n$$

Evaluation

TLAPS Proof Burden

Proving consecution typically most burdensome.

$$Ind \land Next \Rightarrow Ind'$$

If we have

$$Next \triangleq T_1 \lor \ldots \lor T_k$$

$$Ind \triangleq L_1 \land \ldots \land L_n$$

We can trivially decompose into $(k \cdot n)$ subgoals

$$\begin{array}{ccc} (Ind \land T_1 \Rightarrow L_1') & \ldots & (Ind \land T_k \Rightarrow L_1') \\ \vdots & & \\ (Ind \land T_1 \Rightarrow L_n') & \ldots & (Ind \land T_k \Rightarrow L_n') \end{array}$$

Evaluation

TLAPS Proof Burden

Proving consecution typically most burdensome.

$$Ind \wedge Next \Rightarrow Ind'$$

If we have

$$Next \triangleq T_1 \vee \ldots \vee T_k$$

$$Ind \triangleq L_1 \wedge \ldots \wedge L_n$$

We can trivially decompose into $(k \cdot n)$ subgoals

$$\begin{array}{ccc} (Ind \wedge T_1 \Rightarrow L_1') & \ldots & (Ind \wedge T_k \Rightarrow L_1') \\ \vdots & & \\ (Ind \wedge T_1 \Rightarrow L_n') & \ldots & (Ind \wedge T_k \Rightarrow L_n') \end{array}$$

Proof burden metric: % of $(k \cdot n)$ subgoals that required some manual proof effort.