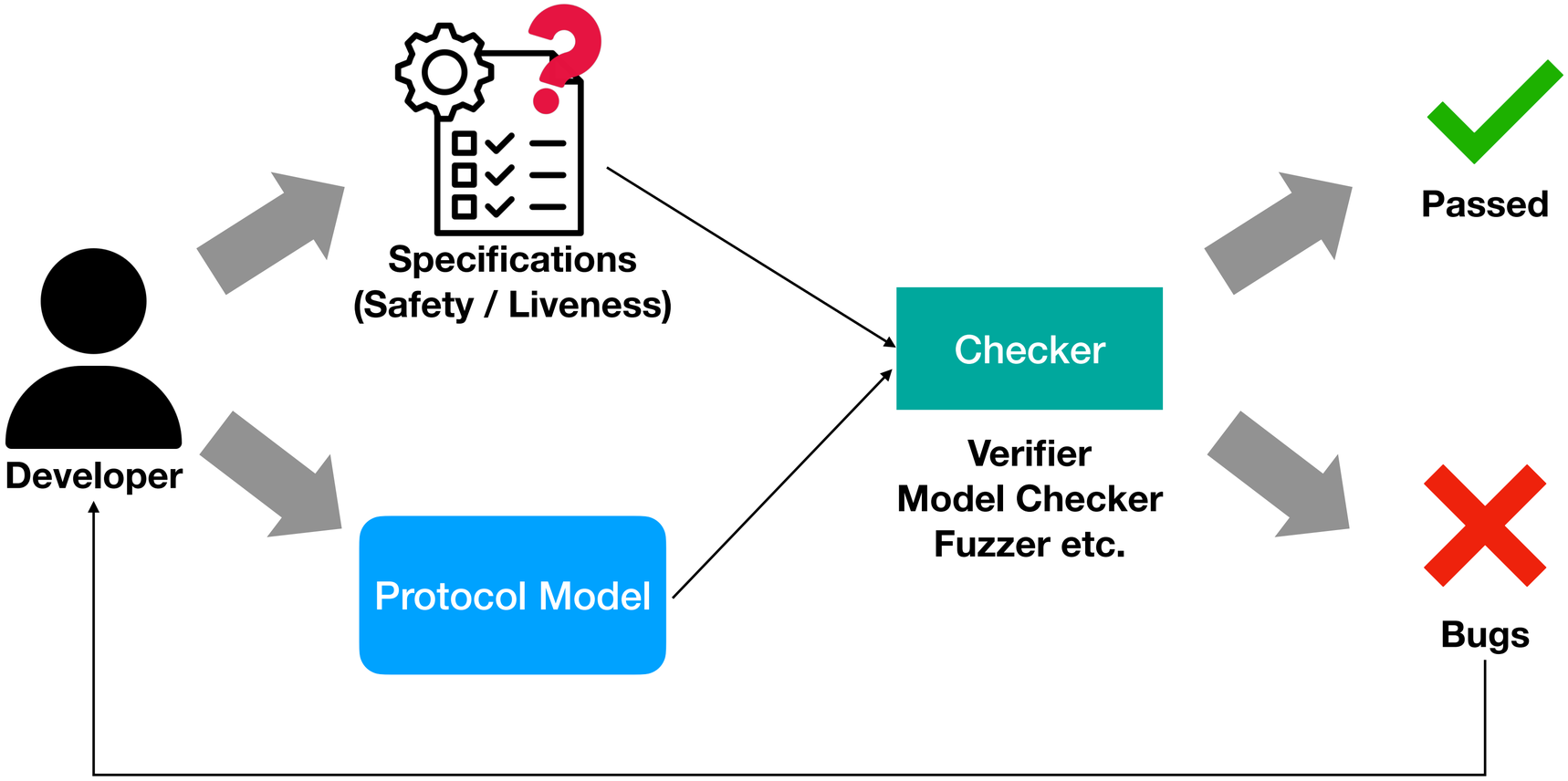
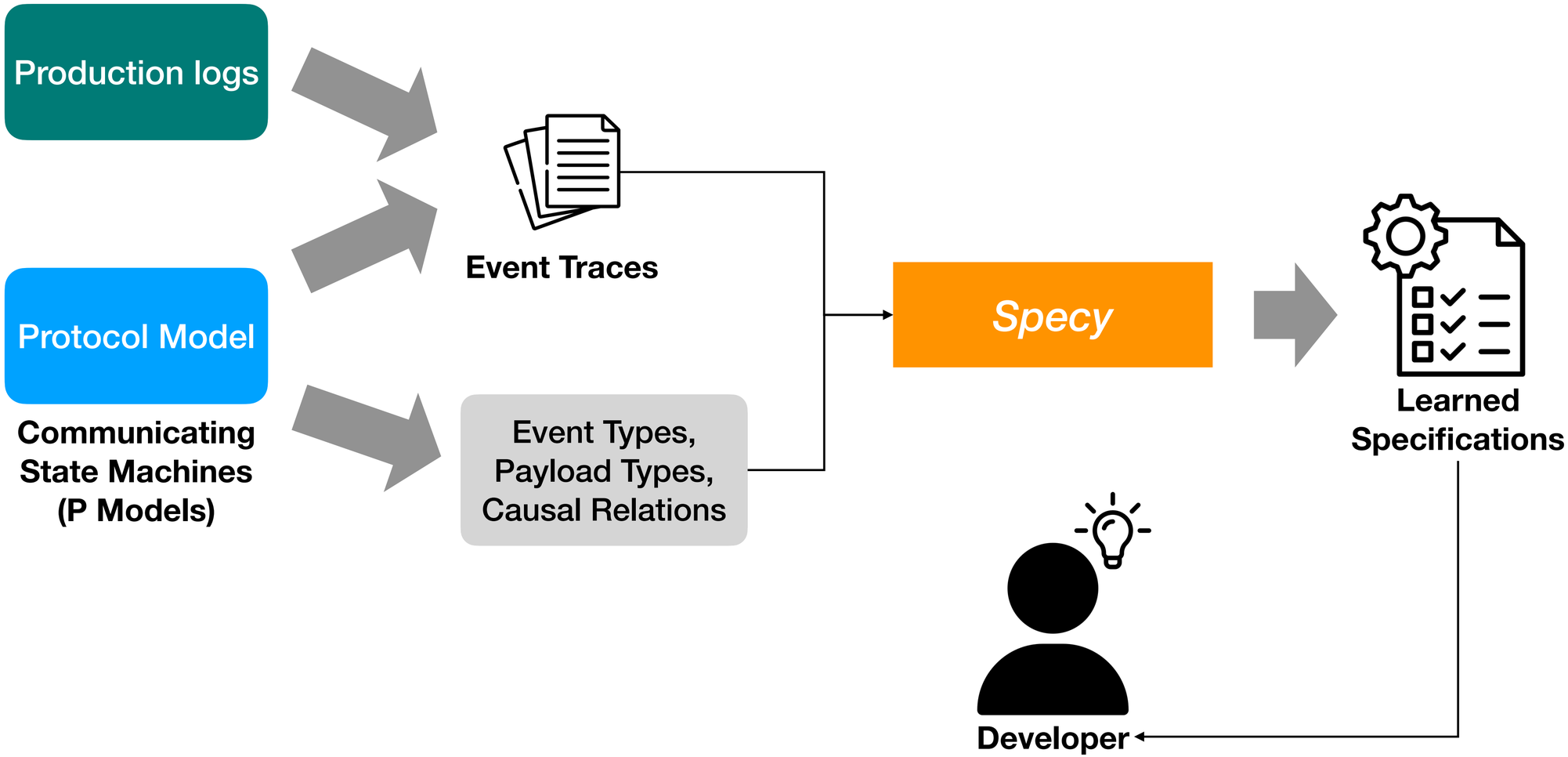
Specy: Automatically Learning Specifications for Distributed Systems from Event Traces

Mike He, Ankush Desai, Aishwarya Jagarapu, Doug Terry, Sharad Malik, Aarti Gupta

Guaranteeing correctness of distributed systems in development



Specy: Automatically Learning Specifications for Distributed Systems



Insight 1

Most specifications can be expressed as (restricted) first-order logic (FOL) formulas over *events*

Examples

External Consistency (Google Spanner)

If a transaction T1 commits before T2 starts, then the timestamp of T1 is less than that of T2

$$\forall e_0, e_1 : eCommit \,.\, e_0 \,.\, commit < e_1 \,.\, start \rightarrow e_0 \,.\, ts < e_1 \,.\, ts$$

Election Safety (Raft)

The leader elected at each term is unique

$$\forall e_0, e_1 : eElected \,.\, e_0 \,.\, term = e_1 \,.\, term \rightarrow e_0 \,.\, leader = e_1 \,.\, leader$$

Whitelist Safety (Firewall)

For each event that is allowed by the firewall, there exists an earlier event that grants the permission to the sender

$$\forall e_0 : eRecv \,.\, e_0 \,.\, allowed \rightarrow \exists e_1 : eGrant \,.\, e_1 \prec e_0 \wedge e_0 \,.\, sender = e_1 \,.\, host$$

Happens-before relation

Quorum Votes (Consensus)

Every decision must have received a majority of votes

$$\forall e_0 : eDecide \,.\, \exists_{\geq quorum} e_1 : eVote \,.\, e_1 \prec e_0 \wedge e_0 \,.\, ballot = e_1 \,.\, vote$$

At least a quorum number of eVote event

Example: Ring Leader Election Protocol

```
1  type tNodeId = int;
2  event eNominate: (vote: tNodeId);
3  event eElectedAsLeader: (nodeId: tNodeId);
4
5  machine Node {
6    var id: tNodeId; # unique identifier of the node
7    var right: machine; # the node to its right
8    start state Init {
9      entry (cfg: (nodeId:tNodeId,next:machine)) {
10       id = cfg.nodeId;
11       right = next;
12       goto Nominating;
13     }
14   }
15   state Nominating {
16     entry {
17         send right,eNominate,(vote=id,)
18     }
19     on eNominate do (payload:(ballot:tNodeId)) {
20       if (payload.vote == id) {
21         announce eElectedAsLeader,(nodeId=id,);
22         goto Won;
23       } else if (payload.vote > id) {
24         send right,eNominate,(vote=payload.vote,);
25       } else {
26         send right,eNominate,(vote=id,);
27       }
28     }
29   }
30   state Won {ignore eNominate;}
31 }
```

Events and types declarations

Node state machine

Local states (variables)

Init state

Initializes the local states, then shift to *Nominating state.*

Nominating state

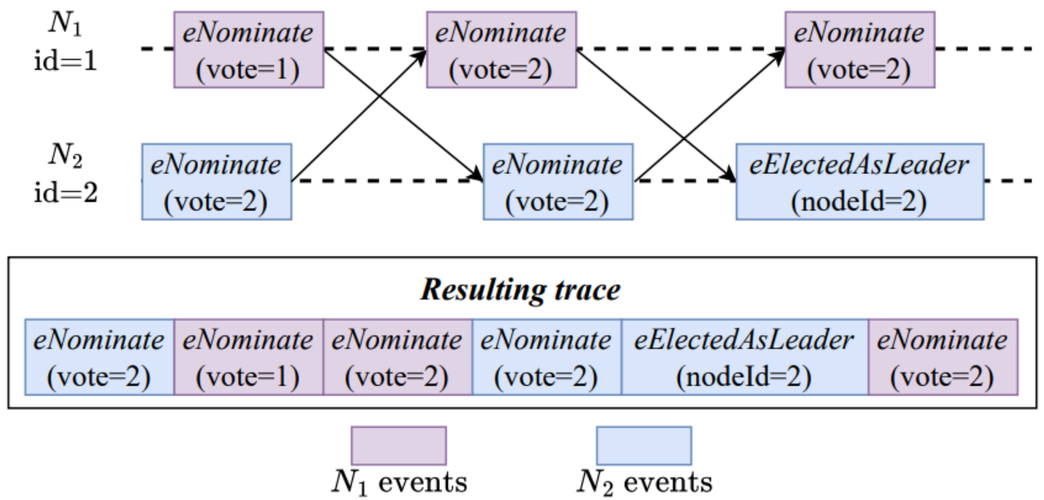
Upon entry, send its own id to the node to its right, then start listening to eNominate

Won state

Enter this state after winning the election

(a) Ring Leader Election protocol modeled in P.

Example: Ring Leader Election Protocol



(b) An example event trace with 2 nodes N_1 andN_2

Unique Leader:

$$\forall e_0, e_1 : eElectedAsLeader \,.\, e_0 \,.\, nodeId = e_1 \,.\, nodeId$$

Leader has the highest id

$$\forall e_0 : eElectedAsLeader, e_1 : eNominate \,.\, e_0 \,.\, nodeId \geq e_1 \,.\, vote$$

Leader exists

$$\forall e_0 : eElectedAsLeader \,.\, \exists e_1 : eNominate \,.\, e_1 \prec e_0 \wedge e_0 \,.\, nodeId = e_1 \,.\, vote$$

Insight 2

Specifications expressed as FOL formulas over events can be decomposed in to the following components

$$\forall (e_i : \tau_i)^+ . \boxed{G} \rightarrow \exists (e_j : \tau_j)^* . \boxed{W} \wedge \boxed{H}$$

Hypotheses
Property holds under Guards and Witnesses

Guards
Control condition

Witnesses (optional)
Existence of certain events

Whitelist Safety (Firewall)

$$\forall e_0 : eRecv . \boxed{e_0 . allowed} \rightarrow \exists e_1 : eGrant . \boxed{e_1 \prec e_0} \wedge \boxed{e_0 . sender = e_1 . host}$$

Grammar of Target Formulas

$$\phi_{\forall} ::= (\forall e_i : \tau_i)^+ . \boxed{\bigwedge \vec{P}} \rightarrow \bigwedge \vec{P}$$

$$\phi_{\forall\exists} ::= (\forall e_i : \tau_i)^+ . \boxed{\bigwedge \vec{P}} \rightarrow (\exists_{SC} e_j : \tau_j)^+ . \boxed{\bigwedge \vec{P}} \bigwedge \vec{P}$$

Top-level rules

Preds	$P ::= P_E \mid P_F \mid \neg P_F$
Event Preds	$P_E ::= T_E = T_E \mid T_E < T_E \mid T_E > T_E$
Field Preds	$P_F ::= T_F = T_F \mid T_F < T_F$
	$\mid T_F \leq T_F \mid uP(uf(T_F^+)^+)$
SetCardinality	$SC(m) ::= m \geq 1 \mid m = C \mid m \leq C \mid m \geq C$

User-provided predicates and functions

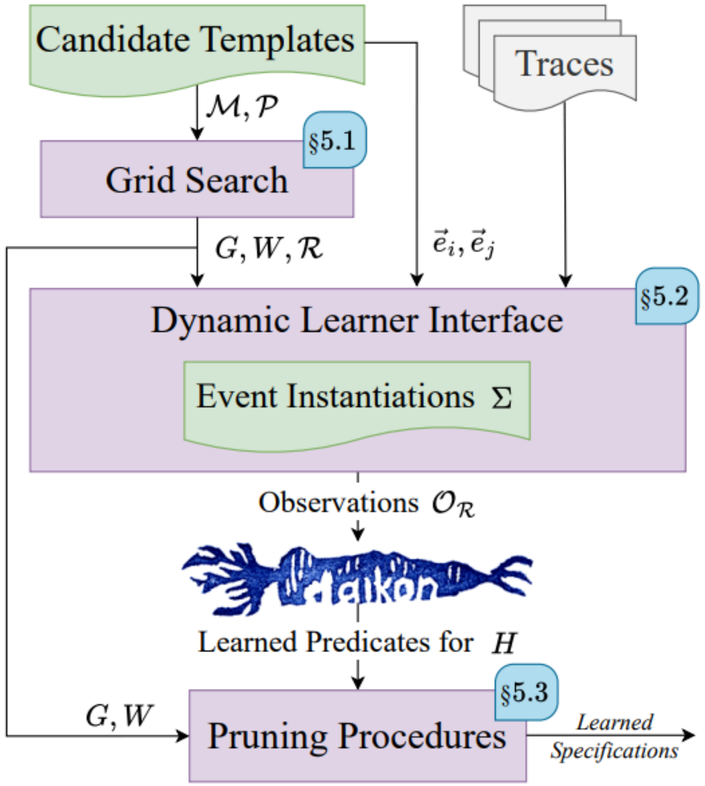
Predicates

Vars	$v \in V$
Constants	$c \in C$
Event Vars	$e \in E$
Event Types	τ_i
Terms	$T ::= T_E \mid T_F$
Event Terms	$T_E ::= \texttt{indexof}(E)$
Field Terms	$T_F ::= C \mid E.v \mid f(T_F^+)$

Terms

Learning Procedure

A novel combination of static enumerative search and dynamic learning



Enumerated

$\forall(e_i : \tau_i)^+$
$\exists(e_j : \tau_j)*$

G W

Restricts the scope for dynamic learning

Learned

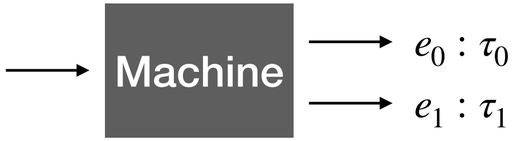
H $\exists_{SC}$

Search space construction

Identifying $\begin{matrix}\forall(e_i:\tau_i)^+\\ \exists(e_j:\tau_j)^*\end{matrix}$ (Event combinations)

When the P model is available, identify event combinations using the following *causal relation patterns*

Same-sourced-sends



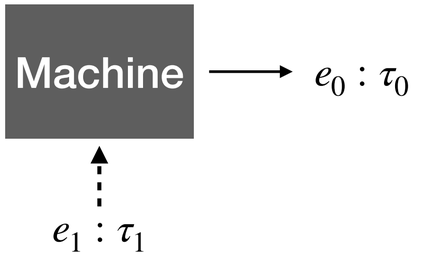
$\forall e_0 : \tau_0, e_1 : \tau_1$

Receive-then-send

$\forall e_0 : \tau_0$
$\exists e_1 : \tau_1$

Conditions under which e_0 can be sent
(e.g. received quorum of e_1)

Send-then-listen



$\forall e_0 : \tau_0$
$\exists e_1 : \tau_1$

May capture liveness properties
(e.g. for each request e_0, there exists a response e_1)

Search space construction

Generating *Terms* and *Predicates*

Vars	$v \in V$
Constants	$c \in C$
Event Vars	$e \in E$
Event Types	τ_i
Terms	$T ::= T_E \mid T_F$
Event Terms	$T_E ::= \texttt{indexof}(E)$
Field Terms	$T_F ::= C \mid E.v \mid f(T_F^+)$
Preds	$P ::= P_E \mid P_F \mid \neg P_F$
Event Preds	$P_E ::= T_E = T_E \mid T_E < T_E \mid T_E > T_E$
Field Preds	$P_F ::= T_F = T_F \mid T_F < T_F$
	$\mid T_F \leq T_F \mid uP(uf(T_F^+)^+)$
SetCardinality	$SC(m) ::= m \geq 1 \mid m = C \mid m \leq C \mid m \geq C$

$(e_i : \tau_i)^+$
$(e_j : \tau_j)^*$

Set of terms $\mathbb{M}$

Set of (atomic) predicates $\mathbb{P}$

Bottom-up grammar expansion
(For a given event combination)

Running example: Ring Leader Election

Generating *Terms* and *Predicates*

$$\forall e_0 : eElectedAsLeader$$
$$\exists e_1 : eNominate$$

Set of terms $\mathbb{M}$

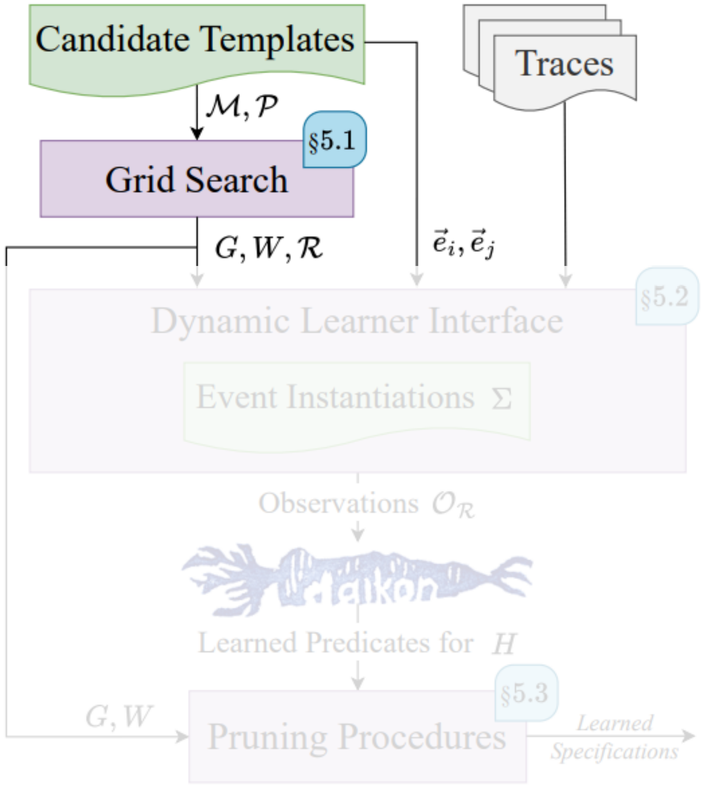
$$e_0 \qquad e_1$$
$$e_0 \,.\, nodeId$$
$$e_1 \,.\, vote$$

Set of (atomic) predicates $\mathbb{P}$

$$e_0 \prec e_1 \qquad e_1 \prec e_0$$
$$e_0 \,.\, nodeId = e_1 \,.\, vote$$
$$e_0 \,.\, nodeId < e_1 \,.\, vote$$
$$e_0 \,.\, nodeId > e_1 \,.\, vote$$
$$\vdots$$

Enumerative Grid Search

Enumerates G W



Given: A set of terms $\mathbb{M}$

A set of predicates $\mathbb{P}$

At a grid point (g, w, r), Specy enumerates

G Conjunctions of g predicates from $\mathbb{P}$

W Conjunctions of w predicates from $\mathbb{P}$

$\mathbb{R}$ Subsets of $\mathbb{M}$ with size r, called *Relate Sets*

over which the dynamic learner learns predicates in H

Dynamic Learner Interface

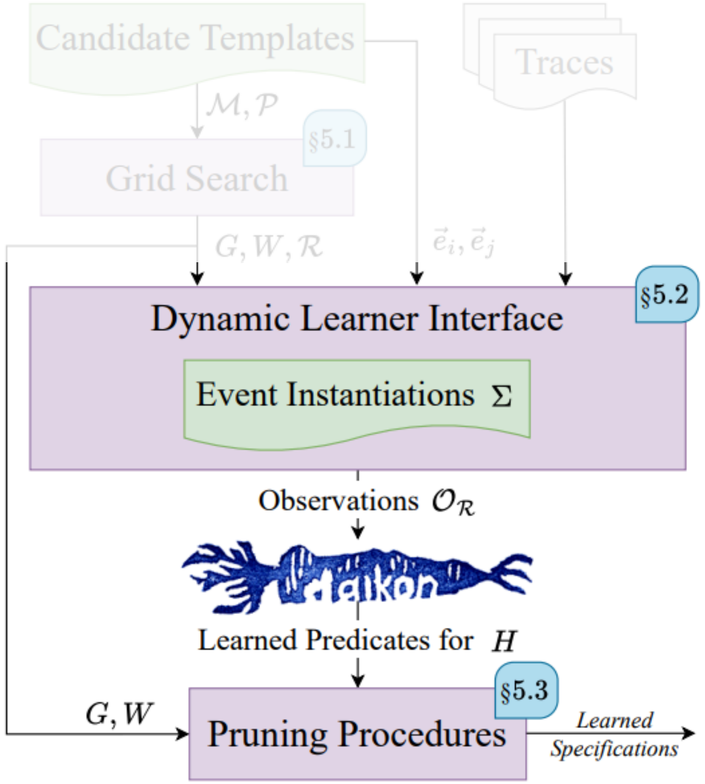
Compute Event Instantiations and Observations

Given: $\begin{matrix}\forall(e_i : \tau_i)^+ \\ \exists(e_j : \tau_j)^*\end{matrix}$, enumerated G W $\mathbb{R}$ and event traces

1. Compute Event instantiations Σ that satisfy G W

2. Compute *Observations* of $\mathbb{R}$, $\mathcal{O}_{\mathbb{R}}$, under each substitution

3. *Learn* predicates for H that hold for *all* observations in $\mathcal{O}_{\mathbb{R}}$



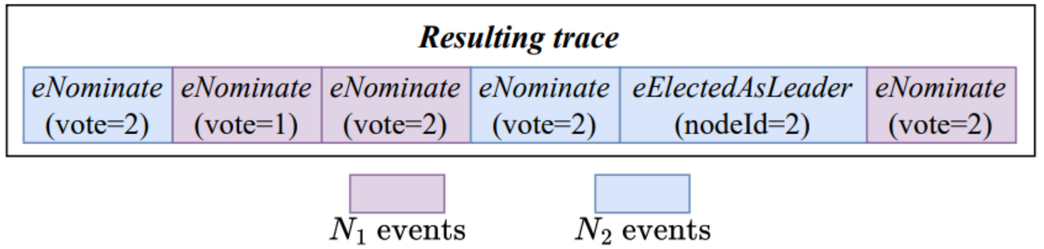
Examples

Learning formulas of $\phi_\forall$

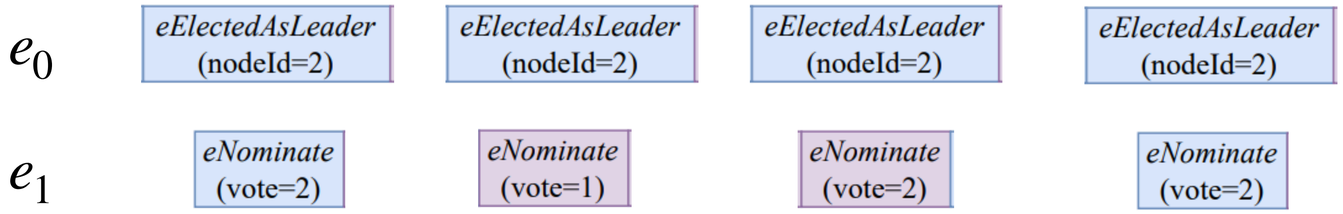
Running example:

$\forall e_0 : eElectedAsLeader, e_1 : eNominate$

$e_1 \prec e_0$ | $\top$ | $\mathbb{R} = \{e_0 . nodeId, e_1 . vote\}$



Substitutions of e_0, e_1 that satisfy $e_1 \prec e_0$



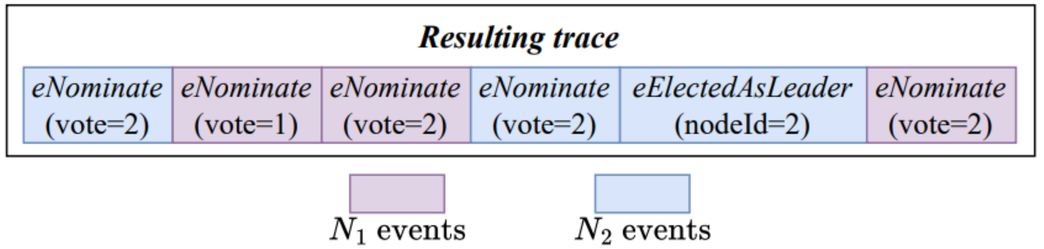
Examples

Learning formulas of $\phi_\forall$

Running example:

$\forall e_0 : eElectedAsLeader, e_1 : eNominate$

$e_1 \prec e_0$ $\quad$ $\top$ $\quad$ $\mathbb{R} = \{e_0.nodeId, e_1.vote\}$



Substitutions of e_0, e_1 that satisfy $e_1 \prec e_0$

e_0	*eElectedAsLeader* (nodeId=2)	*eElectedAsLeader* (nodeId=2)	*eElectedAsLeader* (nodeId=2)	*eElectedAsLeader* (nodeId=2)
e_1	*eNominate* (vote=2)	*eNominate* (vote=1)	*eNominate* (vote=2)	*eNominate* (vote=2)
$\mathcal{O}_\mathbb{R}$	{2,2}	{2,1}	{2,2}	{2,2}

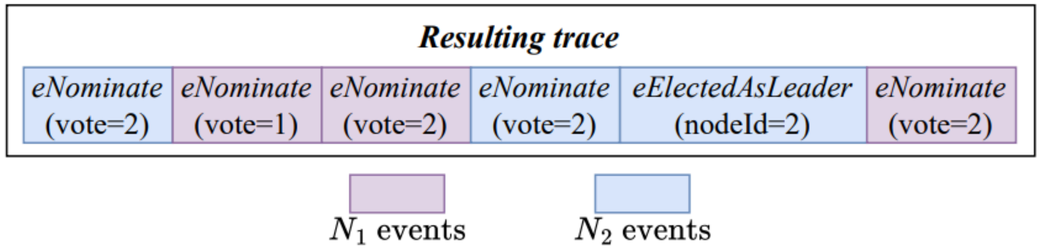
Examples

Learning formulas of $\phi_\forall$

Running example:

$\forall e_0 : eElectedAsLeader, e_1 : eNominate$

$e_1 \prec e_0$ $\quad$ $\top$ $\quad$ $\mathbb{R} = \{e_0 . nodeId, e_1 . vote\}$



$\mathcal{O}_\mathbb{R}$	{2,2}	{2,1}	{2,2}	{2,2}

Learned H $\quad$ $e_0 . nodeId \geq e_1 . vote$

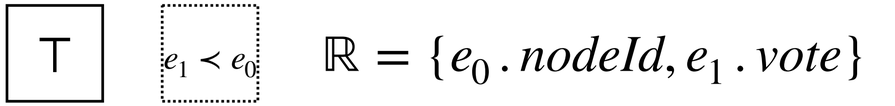
Learned specification $\forall e_0 : eElectedAsLeader, e_1 : eNominate$ $e_1 \prec e_0 \rightarrow e_0 . nodeId \geq e_1 . vote$

Examples

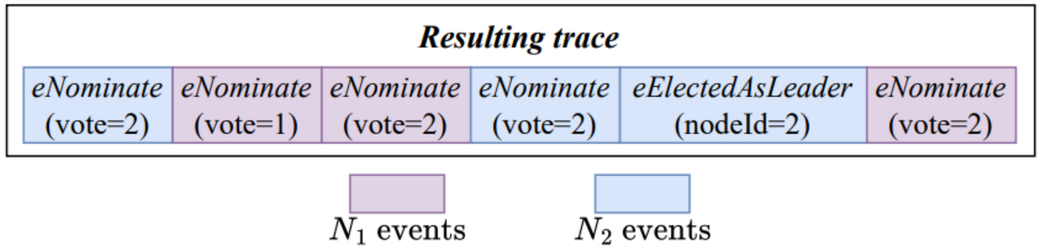
Learning formulas of $\phi_{\forall\exists}$

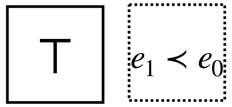
Running example:

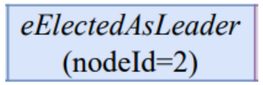
$\forall e_0 : eElectedAsLeader \exists e_1 : eNominate$

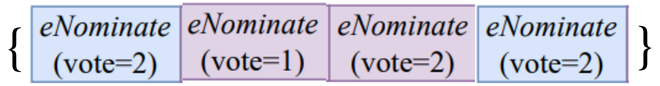


$\mathbb{R} = \{e_0 . nodeId, e_1 . vote\}$



Substitutions of e_0, e_1 that satisfy 

e_0 

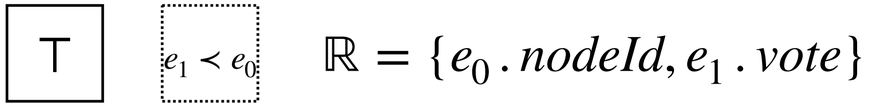
e_1 

Examples

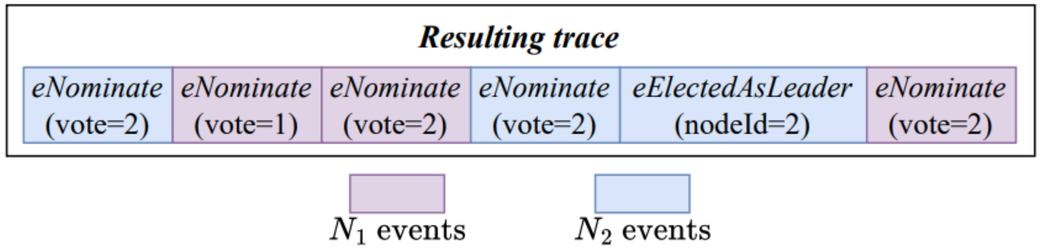
Learning formulas of $\phi_{\forall\exists}$

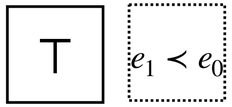
Running example:

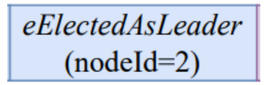
$\forall e_0 : eElectedAsLeader \exists e_1 : eNominate$

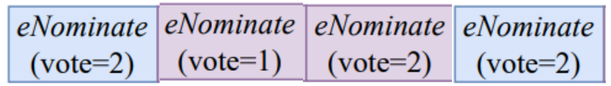


$\mathbb{R} = \{e_0 . nodeId, e_1 . vote\}$



Substitutions of e_0, e_1 that satisfy 

e_0 

e_1 { 

}

$\mathcal{O}_{\mathbb{R}}$ $\{2,\{2,1,2,2\}\}$

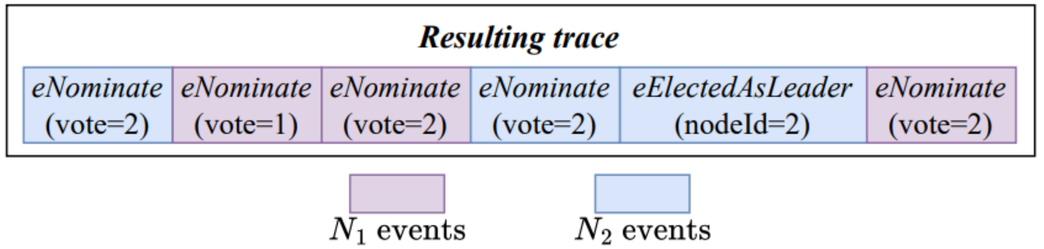
Examples

Learning formulas of $\phi_{\forall\exists}$

Running example:

$\forall e_0 : eElectedAsLeader \exists e_1 : eNominate$

T $\quad e_1 \prec e_0 \quad \mathbb{R} = \{e_0 . nodeId, e_1 . vote\}$



$\mathcal{O}_{\mathbb{R}}$ $\quad \{2,\{2,1,2,2\}\}$

$2 \in \{2,1,2,2\}$

Learned H $\quad e_0 . nodeId = e_1 . vote$

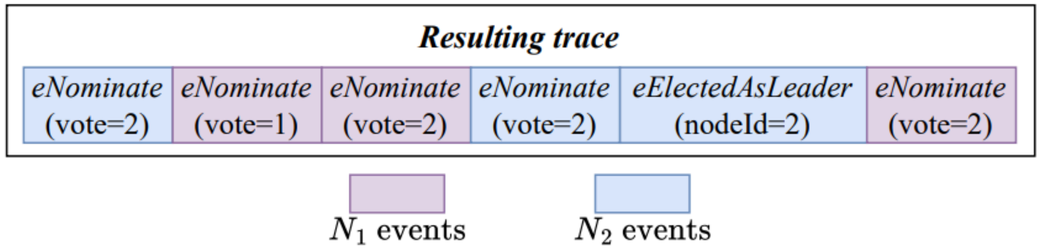
Examples

Learning formulas of $\phi_{\forall\exists}$

Running example:

$\forall e_0 : eElectedAsLeader \exists e_1 : eNominate$

$\boxed{\top}$ $\boxed{e_1 \prec e_0}$ $\mathbb{R} = \{e_0 . nodeId, e_1 . vote\}$



Learned specification $\forall e_0 : eElectedAsLeader \exists e_1 : eNominate \boxed{e_1 \prec e_0} \wedge \boxed{e_0 . nodeId = e_1 . vote}$

Evaluations

1. Learning specifications identified by previous literatures and developers
2. Benefit of Specy in practice
3. Learning inductive invariant for deductive verification

Evaluations

Table 4. Results for learning protocol specifications (RQ1). The LoC column reports the lines of code in the P model. S_{goals} shows the number of protocol specifications in the challenge set, and a ✓ indicates that all are learned by Specy. The next column provides a description (and if ∀∃ is needed). The last column shows the end-to-end run time of Specy (in seconds).

Benchmark	LoC	S_{goals}	Description of Specifications	Time (s)
Ring Leader Election [9]	47	3 ✓	Unique leader, Leader Highest ID, Leader Voted	352
Consensus$^{\top}$[26]	61	1 ✓	Safety (Unique Decision)	925
2PC [3]	238	2 ✓	Atomicity ($\forall\exists_n$), Safety (Commit or Abort)	3935
Sharded KV$^{\top}$*[26]	48	1 ✓	Safety (Unique Shard Owner)	481
Paxos [31]	164	2 ✓	Safety (Unique Decision), B3($\mathcal{B}$)†	1155
Distributed Lock$^{\top}$[37]	73	1 ✓	Safety (Mutual Exclusion)	902
Vertical Paxos [33]	241	2 ✓	Safety (Unique Decision), B3($\mathcal{B}$)†	2589
Firewall [38]	80	1 ✓	Whitelisted Safety (∀∃)	753
Lock Server$^{\top}$[37]	102	1 ✓	Safety (Mutual Exclusion)	620
Chain Replication [49]	452	6 ✓	Update Propagation^{+},Inprocess Requests^{+},Linearizability (∀∃)$^{\dagger\#}$	984
Raft*[40]	1191	5 ✓	Election Safety, State Machine Safety, Log Matching^{+} Leader Append-Only, Leader Completeness^{+}	4223
GlobalClock	177	3 ✓	Clock Monotonicity, Real-time ordering$^{\#}$	805
DBLeaderElection	1178	5 ✓	Monotonic Commits, Unique Leader, Read Consistency$^{\dagger\#}$	697
MVCC-2PC	1135	10 ✓	Atomicity ($\forall\exists_n$)$^{\#}$,Snapshot Isolation (∀∃)$^{\ddagger\#}$	6642

$^{\top}$ P model translated from an IVy model
* Sharded KV models key transfers but not modifications, Raft omits cluster reconfiguration
† Requires user-specified event combinations (UG1)
$^{+}$ Requires user-defined predicate over log entries exposed to traces via events (UG2 & UG3)
‡ Requires events that expose commit logs to traces $^{\#}$ Implied by a subset of the learned specifications

Learned all specifications from well-known and proprietary protocols identified by previous works (28 fully automatically, 15 with user guidance)

Evaluations

Table 5. Number of specifications remaining after each pruning step. S_{raw} and S_{sem} columns show the number of specifications before and after all the pruning procedures, where the reduction ratio (S_{raw}/S_{sem}) is shown in *RR* column.

Benchmark	S_{raw}	S_{syn}	S_{sem}	*RR*	S_{false}
Ring Leader Elect.	479	52	30	15×	0
Consensus	361	28	28	13×	1
2PC	887	60	46	19×	9
Sharded KV	289	23	19	15×	0
Paxos	850	52	49	17×	5
Distributed Lock	740	91	77	9×	0
Vertical Paxos	1650	92	76	21×	5
Firewall	606	46	40	19×	4
Lock Server	399	44	35	11×	0
Chain Replication	533	40	37	14×	2
Raft	1080	73	58	18×	10
GlobalClock	368	37	37	9×	0
DBLeaderElect.	577	50	47	12×	9
MVCC-2PC	5789	334	236	24×	0

Pruning steps reduced by 15x on average while keeping the important specifications

For the proprietary benchmark MVCC-2PC, Specy learned two specifications that are missed by the developers

Evaluations

Table 6. Results for learning inductive invariants using SPECY (RQ3). We use a subset of benchmarks with proofs in PVerifer [38]. I_e and I_s columns show the numbers of event-based and state-based inductive invariants, respectively.

Benchmark	I_e	I_s	All learned
Ring Leader Election*	1	2	✓
Consensus*	1	5	✓
Distributed Lock*	4	4	✓
Lock Server*	4	4	✓
Firewall	1	0	✓
Sharded KV*	1	1	✓

All the event-based inductive invariants are learned automatically
State-based inductive invariants are learned with user guidance

More Details in Our Paper

SPECY: Learning Specifications for Distributed Systems from Event Traces

MIKE HE, Princeton University, USA
ANKUSH DESAI*, Snowflake, USA
AISHWARYA JAGARAPU, Amazon Web Services, USA
DOUG TERRY*, LinkedIn, USA
SHARAD MALIK, Princeton University, USA
AARTI GUPTA, Princeton University, USA

Reasoning about the correctness of distributed systems is a significant challenge, with precise correctness specifications serving as an essential prerequisite to verification. However, identifying and formulating specifications remains a major hurdle for developers in practice. SPECY addresses this challenge by automatically learning specifications from observable *event traces* generated by message exchanges in distributed systems. The system employs a specialized grammar tailored for event-based specifications, incorporating support for quantifiers over events – a capability essential for capturing the complex behavioral patterns inherent in distributed protocols. SPECY utilizes a novel learning procedure that combines grammar-based enumerative search with dynamic learning from event traces, providing effective control over the specification search. We evaluated SPECY on established distributed protocols and industrial case studies, demonstrating its ability to successfully learn important protocol specifications. SPECY can discover previously unidentified specifications overlooked by developers, automatically derive inductive invariants that were previously constructed manually for verification purposes, and, through run-time monitoring in production systems, reveal gaps in testing coverage – highlighting opportunities to leverage specifications in practice.

CCS Concepts: • **Theory of computation** → **Program specifications**; • **Information systems** → *Parallel and distributed DBMSs*; • **Software and its engineering** → **Software verification and validation**.

Additional Key Words and Phrases: specification inference, distributed systems, automated reasoning

ACM Reference Format:
Mike He, Ankush Desai, Aishwarya Jagarapu, Doug Terry, Sharad Malik, and Aarti Gupta. 2026. SPECY: Learning Specifications for Distributed Systems from Event Traces. *Proc. ACM Program. Lang.* 10, OOPSLA1, Article 101 (April 2026), 29 pages. https://doi.org/10.1145/3798209

- **Semantics of event-base specifications**
- **Formalization of the learning procedure**
- **Proof of correctness**