Protocol verification

Hard problem

Separable from software

I4: inductive invariant.

Reachable state.

Hard problem.

Async. network.

- Delay
- Drop
- Dup
- Corrupt.

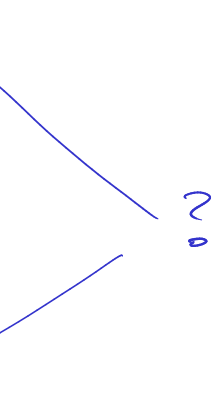
Node failures.

- Crash
- Restart

Inconsistencies.

Open systems: membership.

- Misbehave
- "Byzantine".

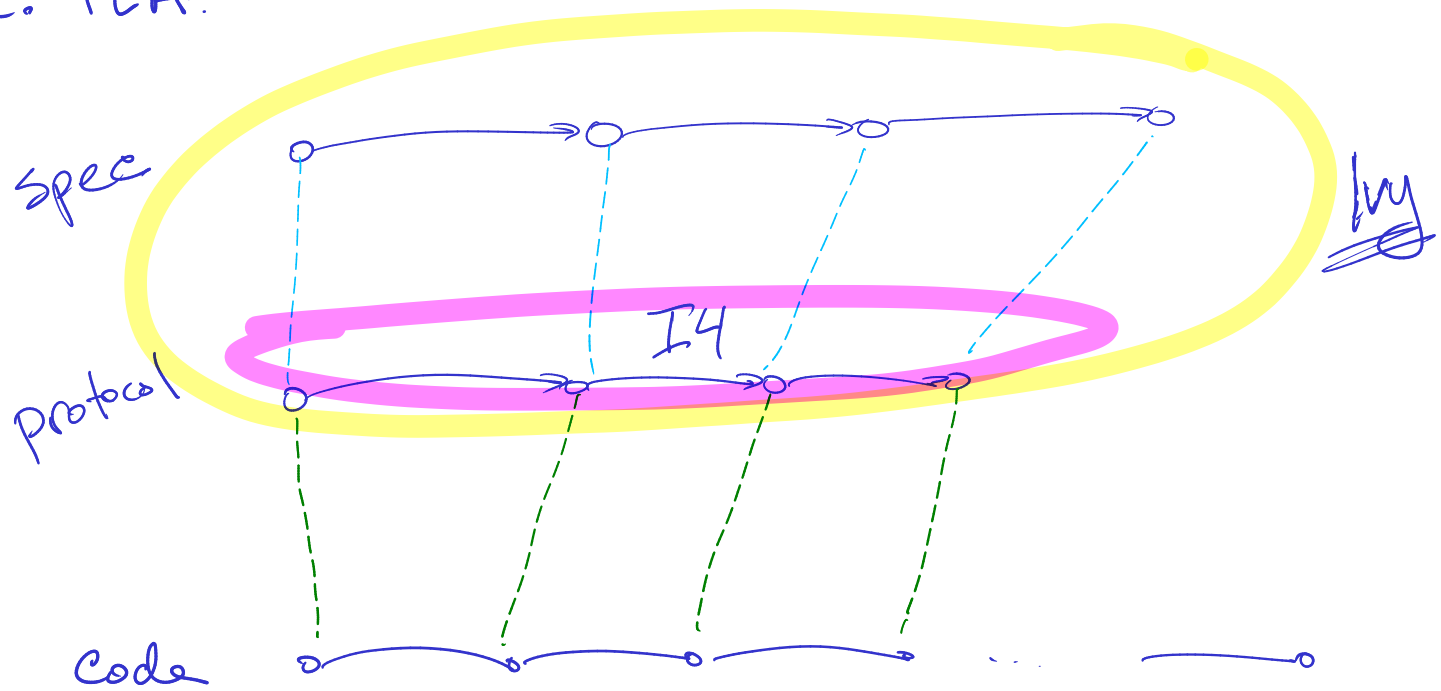


Ivy: protocol verif. tool

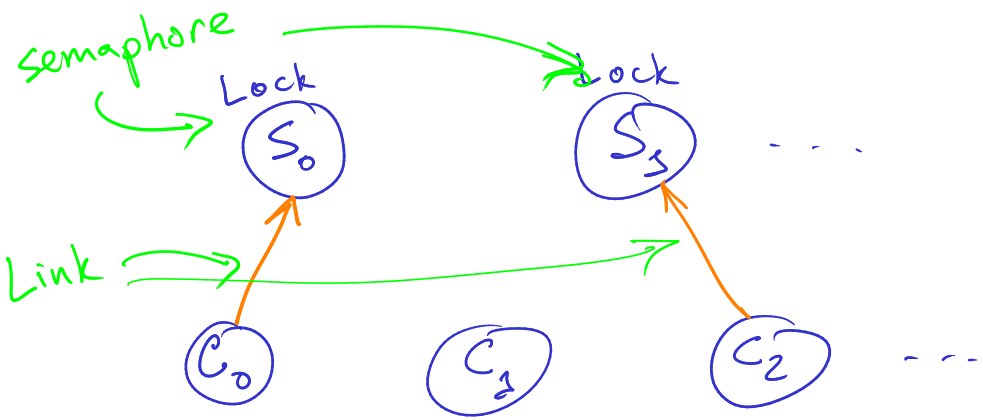
Protocol reasoning.

Close analogue: TLA.

IronFleet:



Ivy example




```
type client
type server

relation link(X:client, Y:server)
relation semaphore(X:server)

after init {
  semaphore(W) := true;
  link(X,Y) := false;
}

action connect(x:client,y:server) = {
  require semaphore(y);
  link(x,y) := true;
  semaphore(y) := false;
}

action disconnect(x:client,y:server) = {
  require link(x,y);
  link(x,y) := false;
  semaphore(y) := true;
}

export connect
export disconnect

invariant [1] link(C1, S) & link(C2, S) -> C1 = C2
```

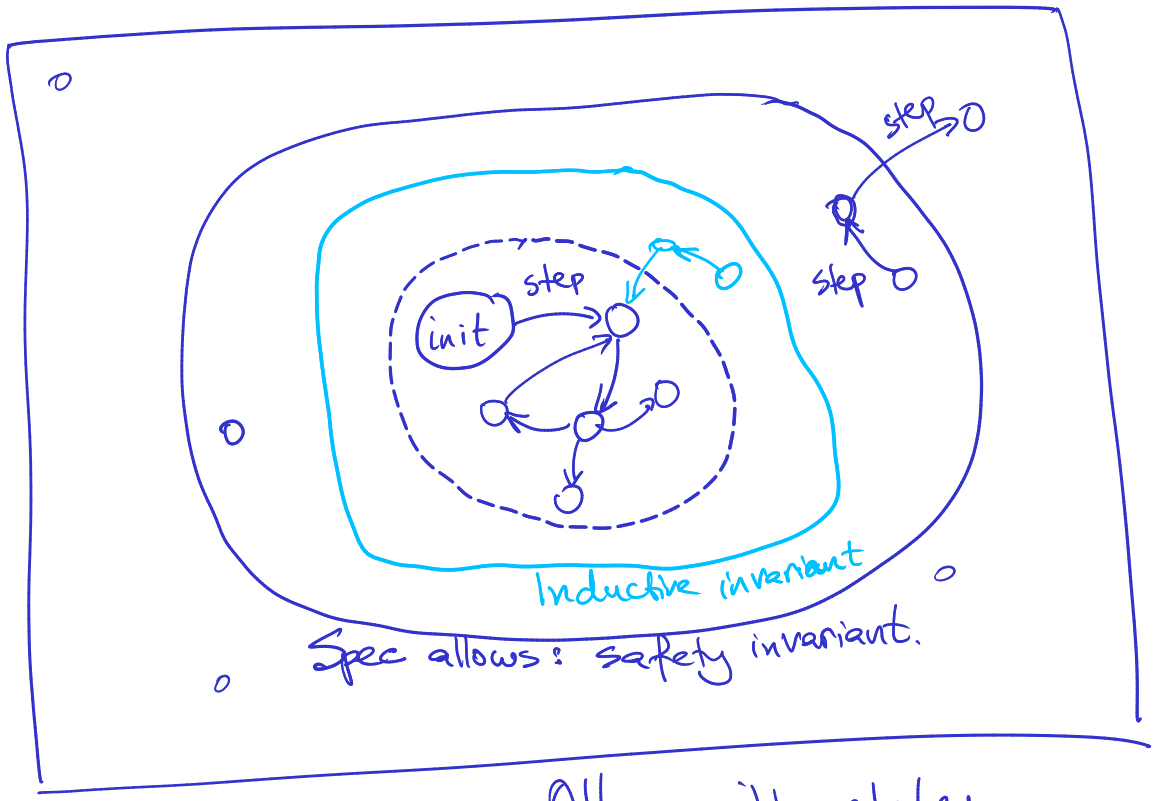
State

Init

Transitions
Actions

Spec
safety predicate

Safety of reachable states



Hard?

- Reachable states defined transitively.
- Spec set: well defined but not inductive.
- Proof plan: inductive invariant

Ivy proof: safety of reach states

1.) $\forall s \in init, \quad Inv(s)$.

2.) $\forall s, s', \quad Inv(s) \wedge step(s, s') \rightarrow Inv(s')$

Inv = over-approx of reachable states.

Why automation?

Ivy code → Z3 constraints.

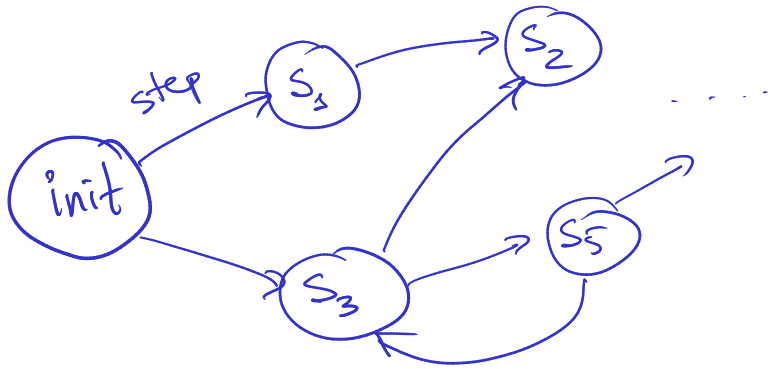
Restricted logic

Limited $\exists$, predicates, ...

Z3 check ①, ②

Unbounded ∀ proof state

Model checking: safety of reach. states



Bounded state size

	s	c_0	c_1
	sem(s)	link(c_0, s)	link(c_1, s)
init:	T	F	F
	F	T	F
	F	F	T

safety spec

AVR: synthesize compact predicate for reachable states.

I4's trick:

AVR predicate ⇝ Ivy invariant.

I4's workflow