Teaching Alloy with Alloy4Fun

Nuno Macedo and Alcino Cunha

ABZ'23

Who are we?

- Teaching Alloy for several years (besides research)
  - Alcino Cunha, University of Minho, 15 years
  - Nuno Macedo, University of Porto, 5 years
- Mandatory (>100 students) and optional (<20 students) classes
- Last 3 years with Alloy4Fun
- Led the development by MSc students

Universidade do Minho
Escola de Engenharia

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Program

- Overview of the Alloy4Fun platform
- Defining logical challenges in Alloy4Fun
- Analyzing student submission data

Writing and executing models

Alloy4Fun overview

- Web-platform to specify, analyze and share Alloy models
- Doesn’t support all features of the Analyzer, best suited for simple models
- Additional features allow the creation of challenges to be used in classes:
    - Ability to *share* models and instances, accompanied by themes
    - Ability to define *secrets* in the model, allows the definition of exercises
    - All *data* is collected, to monitor progress and identify bottlenecks

Alloy4Fun overview

Alloy4Fun

```
module network

some sig Node {
  adj : set Node,
}
sig Router in Node {}

run good_example {
  all n : Node | some n.adj
} for exactly 3 Node

run bad_example {
  no Node
} for 3 Node
```

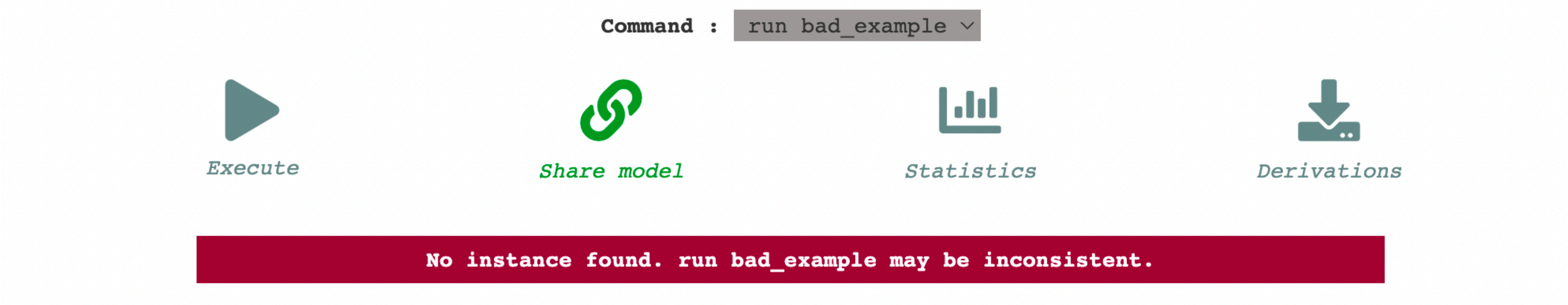
Command : run bad_example

Execute *Share model* *Statistics* *Derivations*

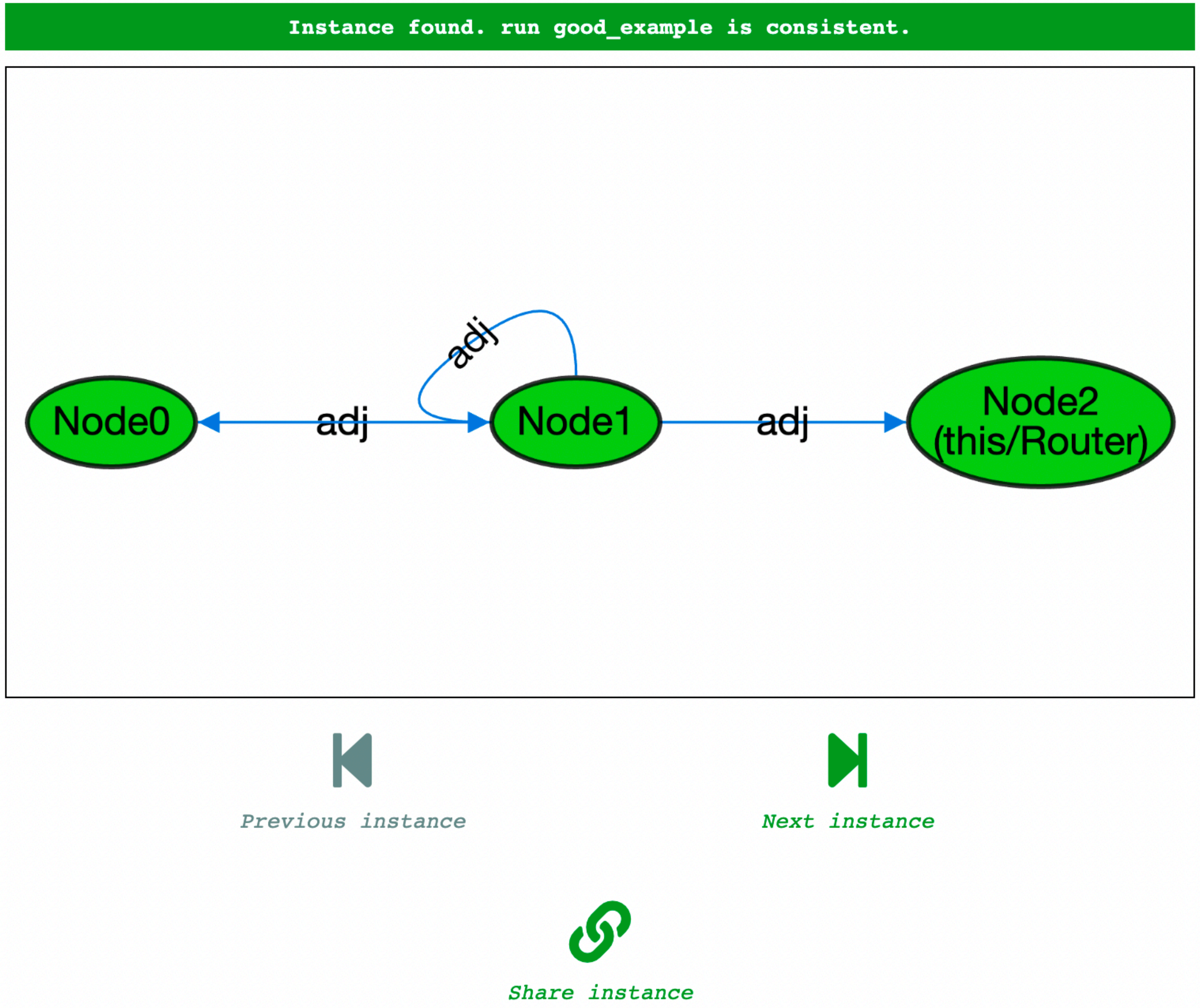
Model execution

- ▶ launches the analysis of a command likewise the Analyzer
- If multiple commands defined, combobox allows selection
- If satisfiable, an instance/counter-example is graphically shown
- Execution disabled until changes are made (or other command selected)

Model execution



Instance visualization



Instance visualization

- If the instance is a trace, → and ← allow state navigation (↺ for last state)
- Single state shown at a time
- Current state identified in bottom-right corner

Alloy4Fun overview

Alloy4Fun

```
module network

abstract sig State {}
one sig Active, Inactive extends State {}
sig Message {}

some sig Node {
  adj : set Node,
  var inbox : set Message,
  var state : one State
}
sig Router in Node {}

run good_example {
  all n : Node | some n.adj
  inbox' != inbox
} for exactly 2 Node, 2 Message

run bad_example {
  no Node
} for 3 Node, 2 Message
```

Command : run good_example

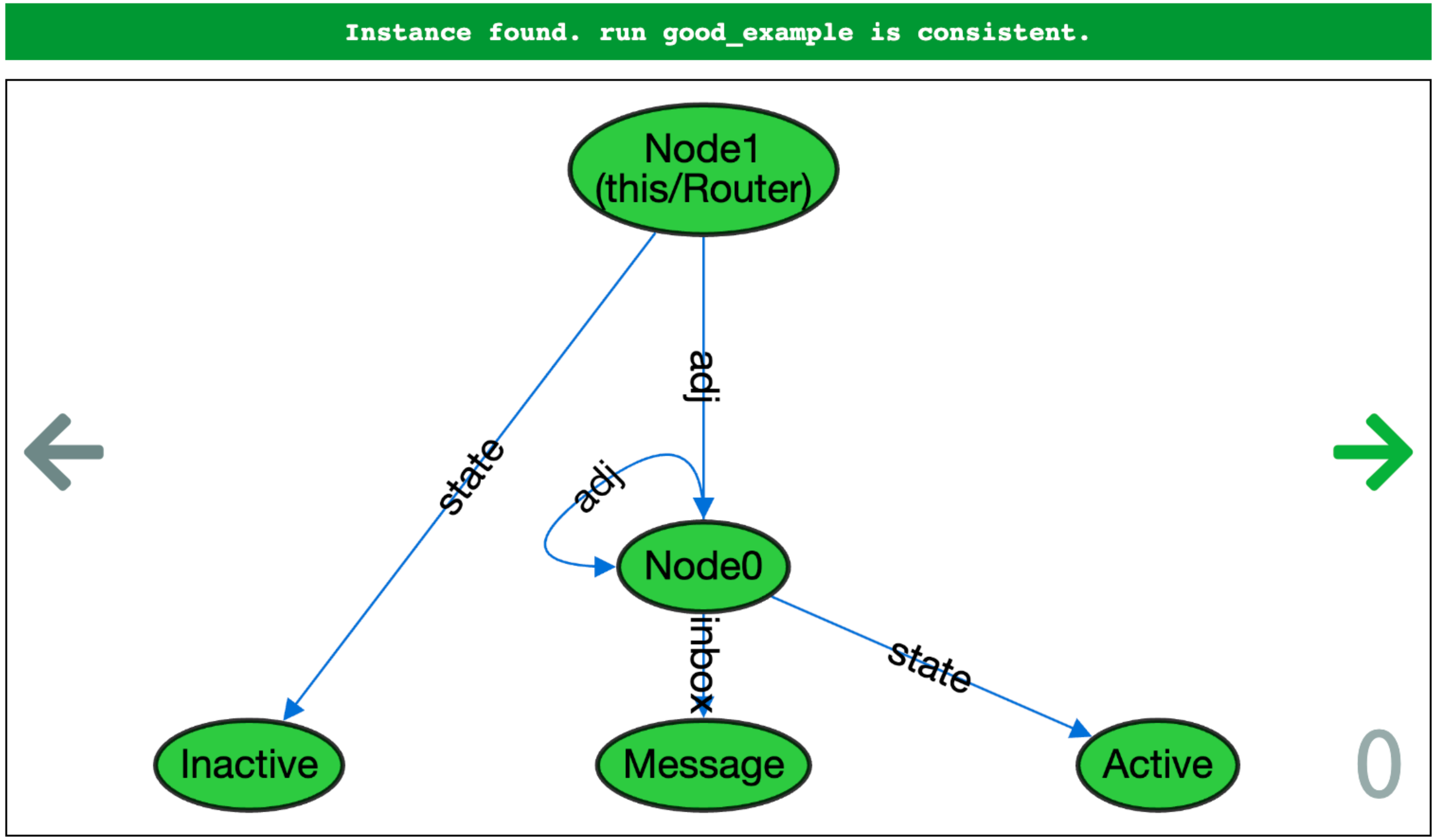
Execute

Share model

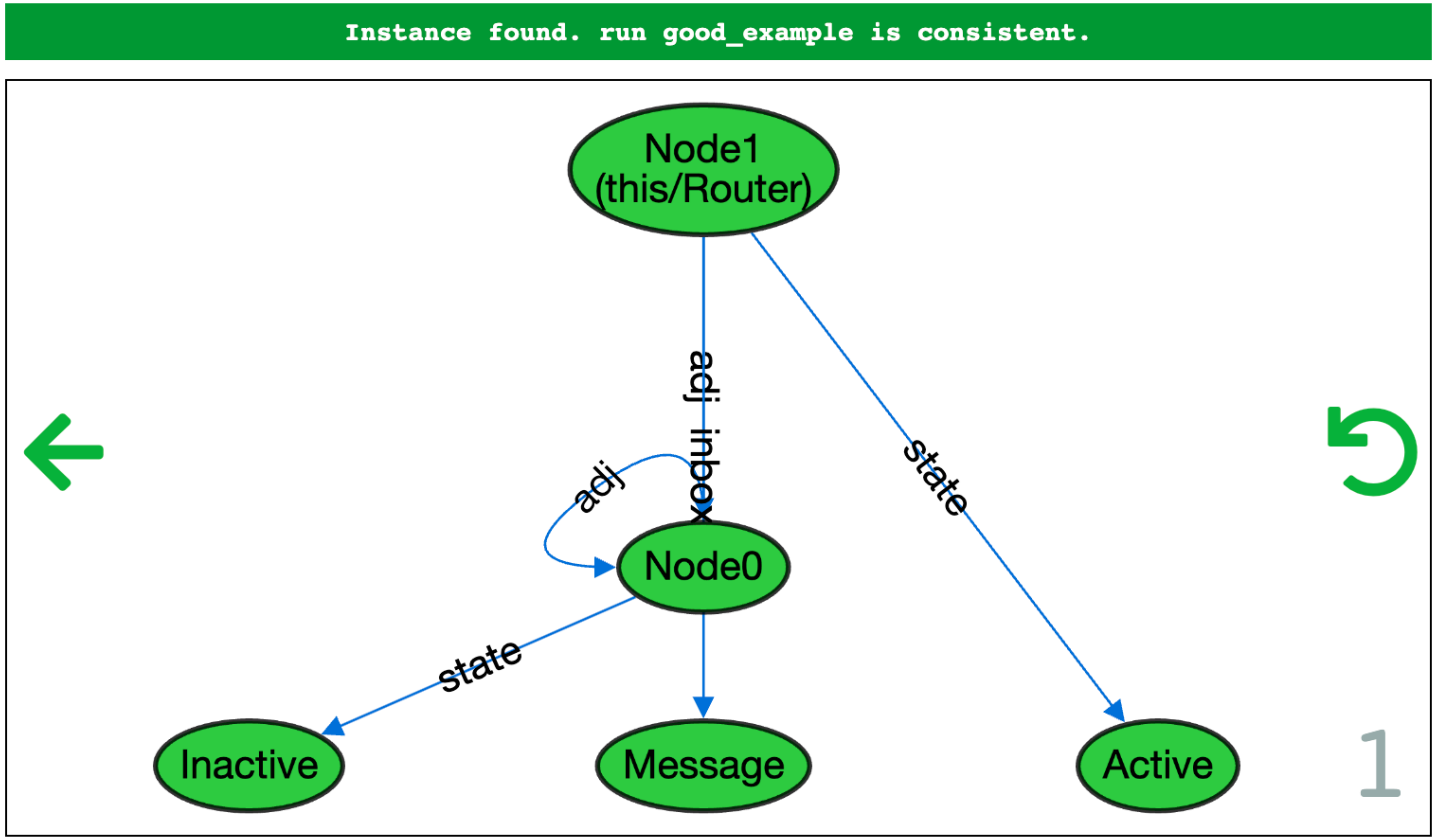
Statistics

Derivations

Instance visualization



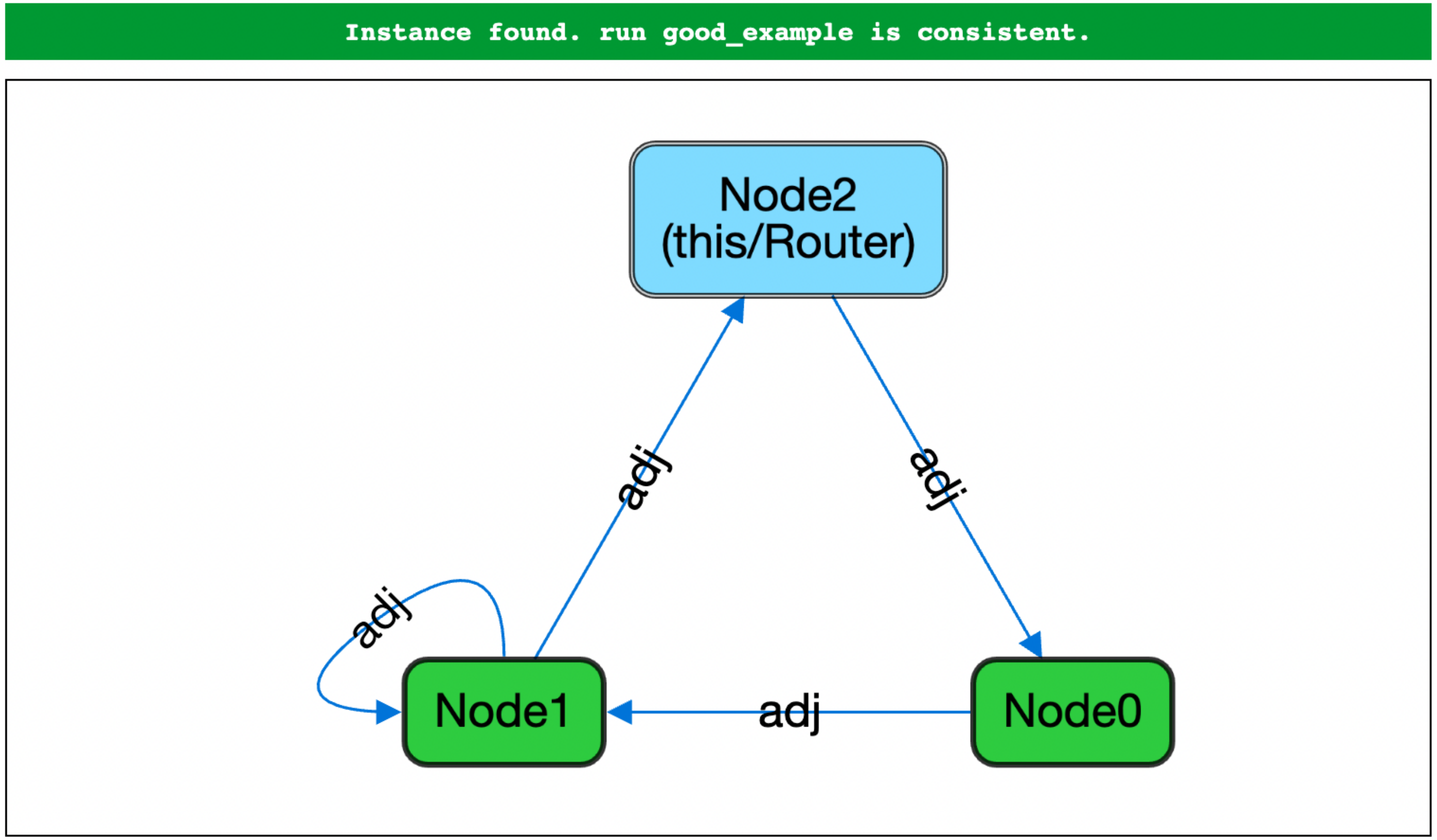
Instance visualization



Customizing visualization

- Likewise the Analyzer, visualization can be customized
- Accessible by right-clicking elements in the visualizer
- Not all the options available, only essential ones
- Layout not as strict, alternative algorithms and manual positioning

Customizing visualization



Instance enumeration

- Only basic enumeration is provided, ⏭ for a new instance
- Can navigate back to previously seen instances ⏮

Sharing models and instances

Sharing models

- The current state of the model can be shared through a permalink 🔗
- Visualization customization shared alongside the model
- Useful when interacting with students to discuss attempts

Sharing models

Execute

Share model

Statistics

Derivations

public link

http://alloy4fun.inesctec.pt/zqdPWk6gpPyKaQvgG

Copy to clipboard

Sharing instances

- Instances can also be shared with a similar mechanism 🔗
- Visualization is fully preserved, including the position of the elements

Sharing instances

Share instance

instance link

http://alloy4fun.inesctec.pt/ehTGR6s66iPxLRpcF

Copy to clipboard

Managing links

- Alloy4Fun is completely *anonymous* by design
- No accounts for users to track their permalinks
- Users must maintain their permalinks through external means

Model secrets

Secret annotations

- Secrets in a model are introduced with annotation `//SECRET`
- Will make succeeding paragraph a secret
  - Declaration of signature, predicate, command, …
- Comments, retro-compatible with Analyzer

Public and private views

- Once secrets are introduced in the model, there are two views available:
  - *Private*: shows the complete model with secrets
  - *Public*: hides the secret declarations
    - They can still be used if name is known
    - Secret signatures still shown (unless hidden by theme)
    - Secret commands can be selected for execution
- Different permalinks 🔗 generated for models with secrets

Public and private views

public link

http://alloy4fun.inesctec.pt/kNnE7E2BzeJjdf8Ly

Copy to clipboard

private link

http://alloy4fun.inesctec.pt/dXkFAkxFMzEprwfiP

Copy to clipboard

Private view

```
module network

some sig Node {
  adj : set Node,
}
sig Router in Node {}

//SECRET
run good_example {
  all n : Node | some n.adj
} for exactly 3 Node

//SECRET
run bad_example {
  no Node
} for 3 Node
```

Command : run good_example

Execute

Share model

Statistics

Derivations

Public view

```
module network

some sig Node {
  adj : set Node,
}
sig Router in Node {}
```

Command : run good_example

Execute *Share model* *Statistics* *Derivations*

Public and private views

- Interface changes slightly
- Public view: warns user of hidden paragraphs
- Private view: and enable the extraction of submission statistics

Evolving views

- Public views can be changed and re-shared, preserving original secrets
- But secrets only inherited from a single model
- Introducing secrets in a public view breaks the connection with the original
- When evolving a private view, shared links will still point to the original, must re-generate

Defining challenges

Defining challenges

- Secret paragraphs can be used to create *challenges* for students
- Students will be asked to fill the body of predicates
- Alloy's solving engine will test student attempts against the lecturer oracle
- Will provide automatic feedback regarding correction of attempt

Logical challenges

- Alloy4Fun is best suited for simple challenges to train the formal specification of properties
- Students are expected to only complete a predicate
- Secret checks compare student predicates with oracles behind the scenes
- Checks logical *equivalence*: student solution may be syntactically different
- Not expected to change the structure: would undermine automatic tests

Relational logic challenges

- Recipe for challenge *N* within a model:
  - Specify the correct specification as predicate `oracleN`
  - Mark `oracleN` as *`//SECRET`*
  - Declare a header predicate `specN` for the student submission, leave it empty
  - Annotate predicate `specN` with the description of the expected property
  - Declare a **`check`** command `specN` that verifies the equivalence of `oracleN` and `specN` (take care with the scope)
  - Mark command `specN` as *`//SECRET`*

Relational challenge: private

```
//SECRET
pred oracle1 {
  ~adj = adj
}
pred spec1 {
  // the network is undirected

}
//SECRET
check spec1 {
  spec1 iff oracle1
} for 4
```

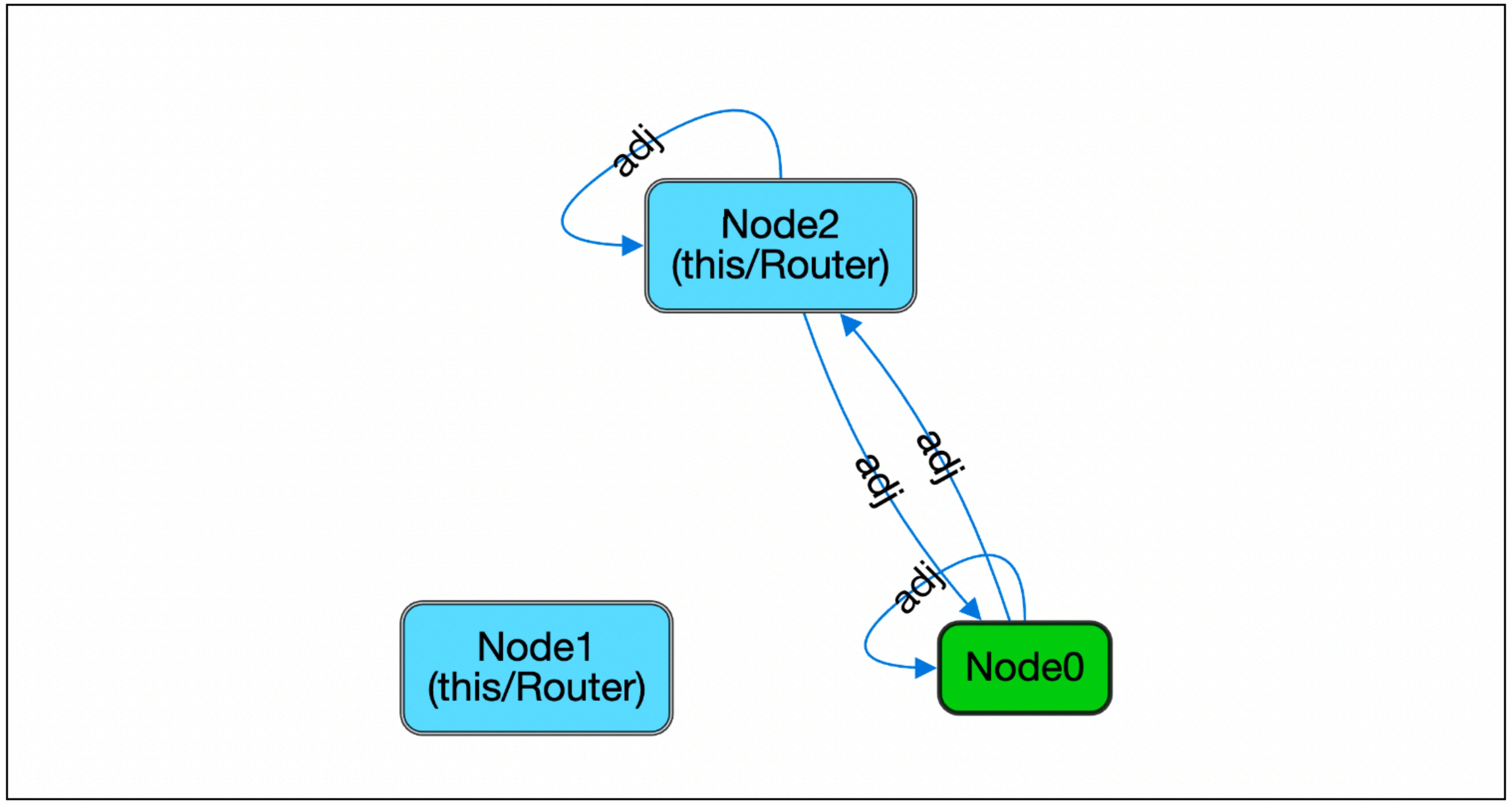
Relational challenge: public

```
pred spec1 {
  // the network is undirected

}
```

Relational challenge: public

```
pred spec1 {
  // the network is undirected
  all n : Node | n->n in adj
}
```

Counter-example found. check spec1 is invalid.



Improving challenge feedback

- Challenge check commands show a counter-example whenever the submission and oracle are not equivalent
- May be due to under-specification, over-specification (or both)
- Determining which is the case from counter-examples is challenging for students
- We can add extra information in the visualization to aid students

Improving challenge feedback

- Recipe for header:
  - Declare an abstract singleton signature `RejectedBy` to represent the feedback
  - Declare extensions of `RejectedBy` for the cases `ThisShouldBeRejected` and `ThisShouldBeAccepted`
  - Customize the visualization of `RejectedBy` atoms as seen fit
- Recipe for challenge *N*:
  - Keep predicates `oracleN` and `specN` as before
  - Add a precondition to command `specN` to only consider counter-examples where the correct `RejectedBy` atom is present

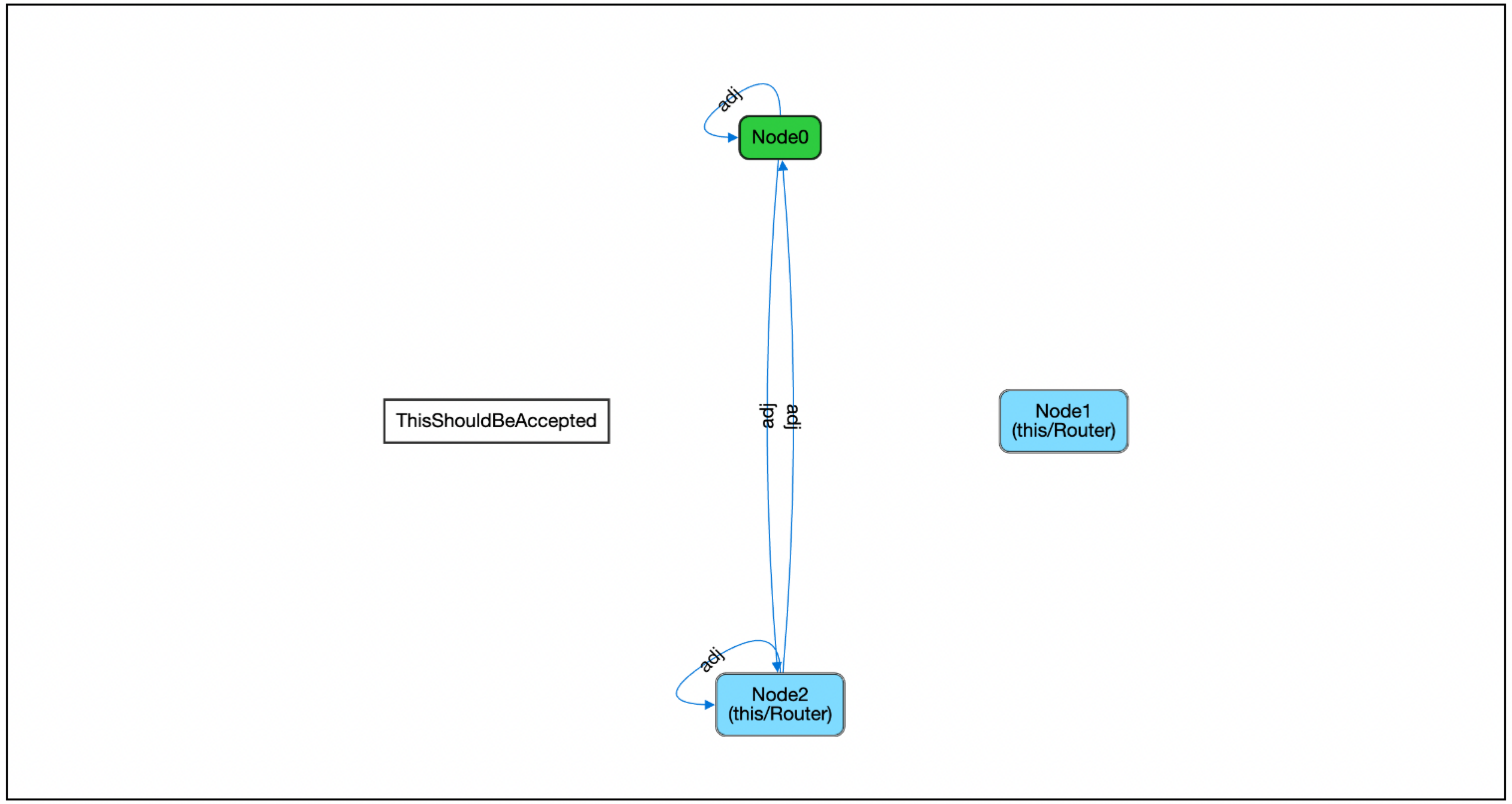
Improving feedback: private

```
//SECRET
abstract one sig RejectedBy {}
//SECRET
sig ThisShouldBeRejected, ThisShouldBeAccepted extends RejectedBy {}

//SECRET
pred oracle1 {
  ~adj = adj
}
pred spec1 {
  // the network is undirected

}
//SECRET
check spec1 {
  (some ThisShouldBeRejected iff (spec1 and not oracle1)) implies
    (spec1 iff oracle1)
} for 4
```

Improving feedback: public

```
pred spec1 {
  // the network is undirected
  all n : Node | n->n in adj
}
```

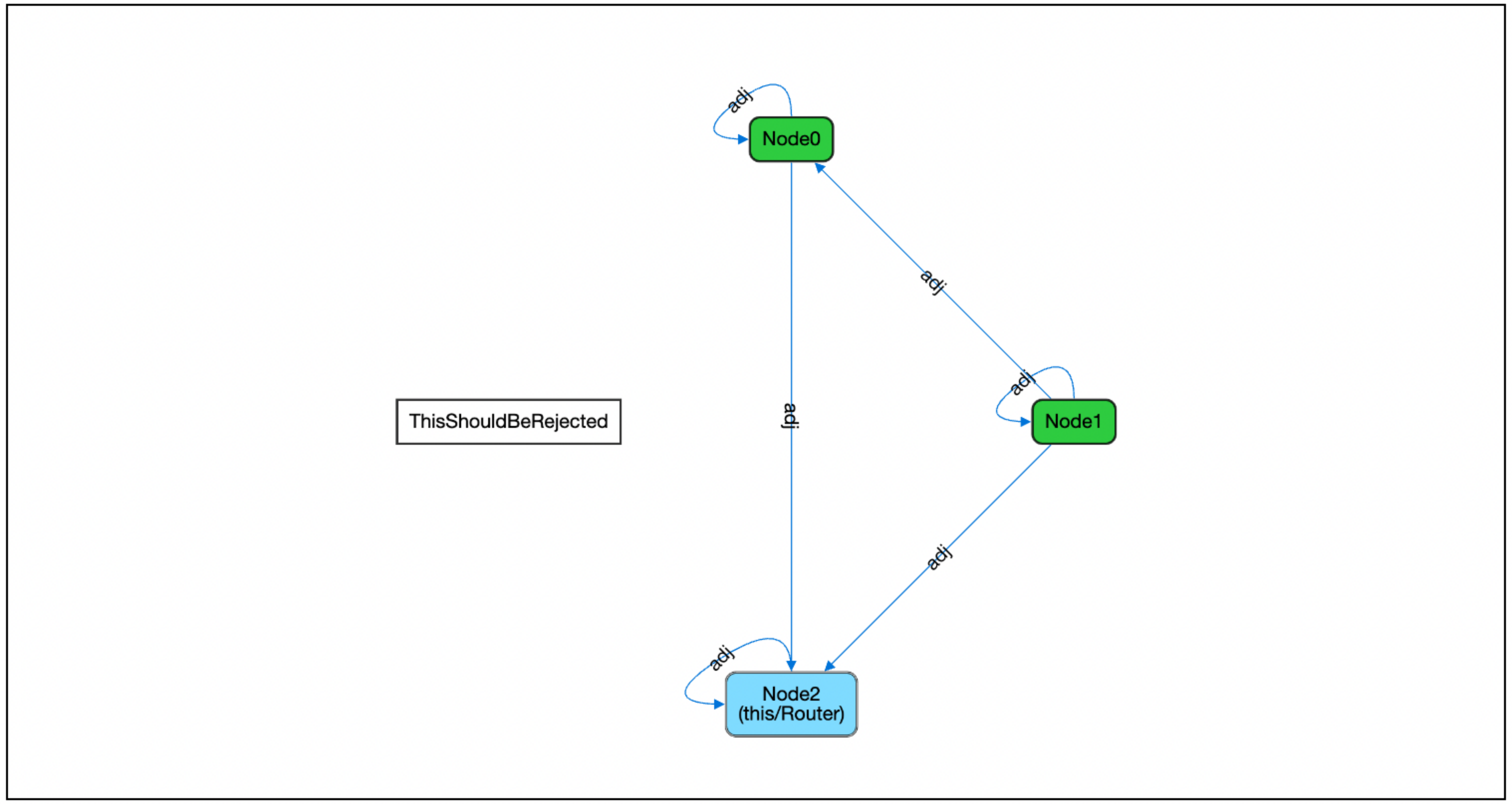
Counter-example found. check spec1 is invalid.



Improving feedback: public

```
pred spec1 {
  // the network is undirected
  all n : Node | n->n in adj
}
```

Counter-example found. check spec1 is invalid.



Providing partial feedback

- The same strategy can be used to give feedback about challenges with sub-specifications
- Or even check all challenges at once and have feedback about which one is failing

Partial feedback: private

```
//SECRET
abstract sig Subspec {}
//SECRET
lone sig FailsSubspec1, FailsSubspec2 extends Subspec {}
```

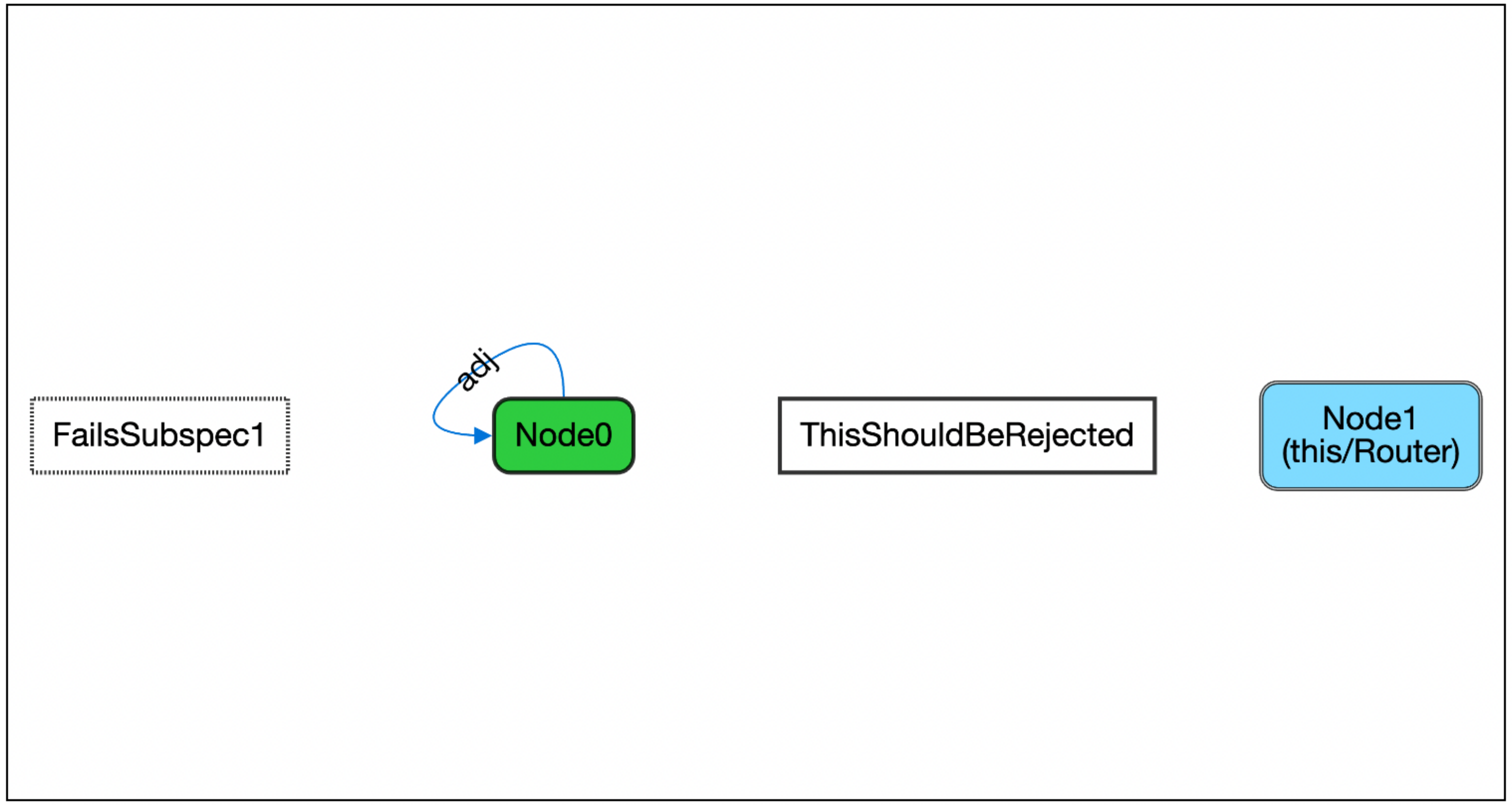
Partial feedback: private

```
//SECRET
pred oracle2 {
  oracle2a and oracle2b }
//SECRET
pred oracle2a {
  all r:Router | some n1 , n2 : r.adj | n1 != n2 }
//SECRET
pred oracle2b {
  all r:Router | r not in r.adj }
//SECRET
check spec2 {
  { some ThisShouldBeRejected iff (spec2 and not oracle2)
    some FailsSubspec1 iff (spec2 and not oracle2a)
    some FailsSubspec2 iff (spec2 and not oracle2b)
  } implies (spec2 iff oracle2)
}
```

Partial feedback: public

```
pred spec2 {
  // 1) router nodes have more than one adjacent node

  // 2) router nodes are not adjacent to themselves
  no iden & adj :> Router }
```

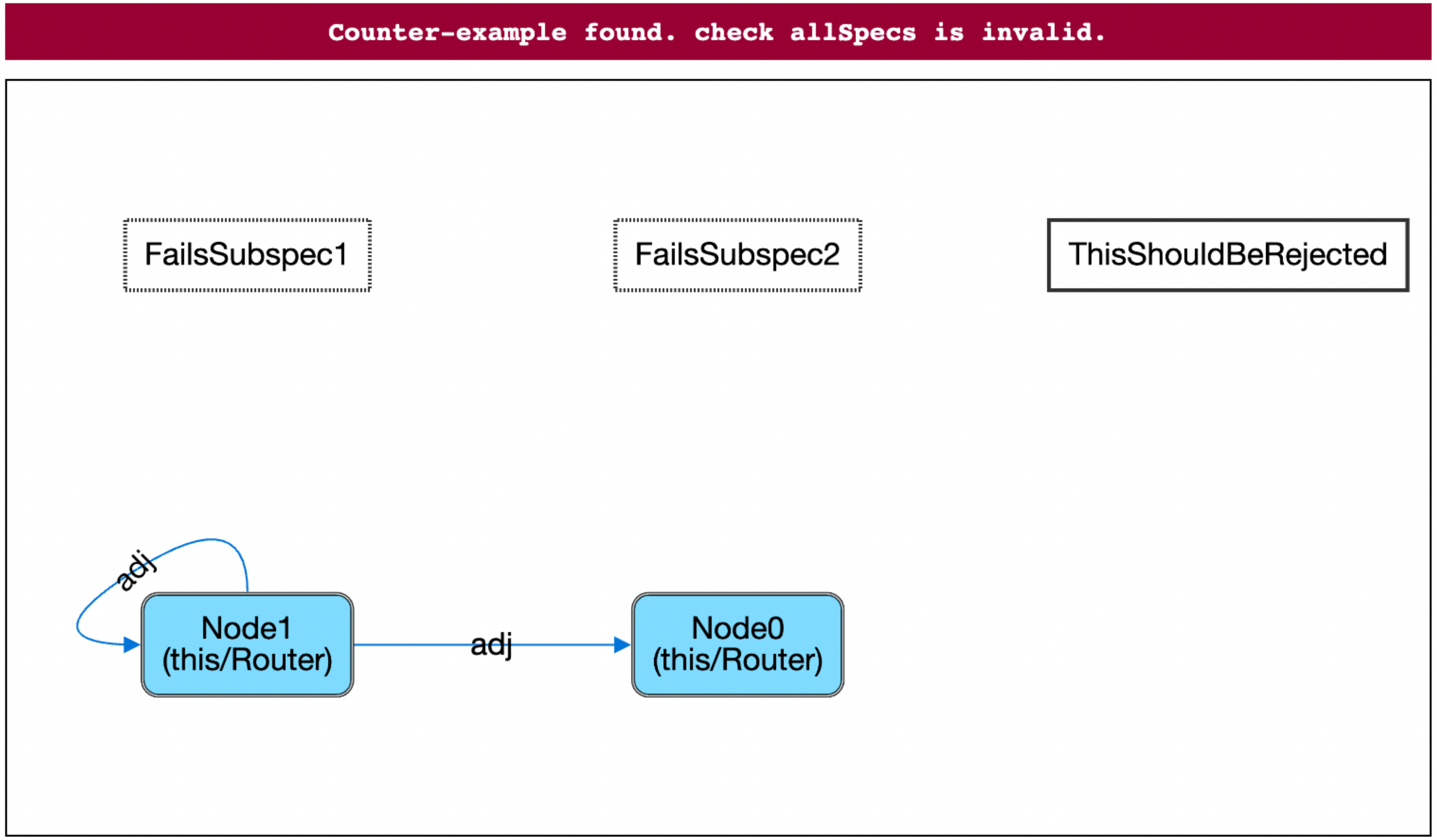
Counter-example found. check spec2 is invalid.



Partial feedback: private

```
//SECRET
check allSpecs {
  let specs = spec1 and spec2
  { some ThisShouldBeRejected iff
      (() and not (oracle1 and oracle2))
    some FailsSubspec1 iff (spec1 and not oracle1)
    some FailsSubspec2 iff (spec2 and not oracle2)
  } implies
    ((spec1 and spec2) iff (oracle1 and oracle2))
}
```

Partial feedback: public



Incremental challenge assumptions

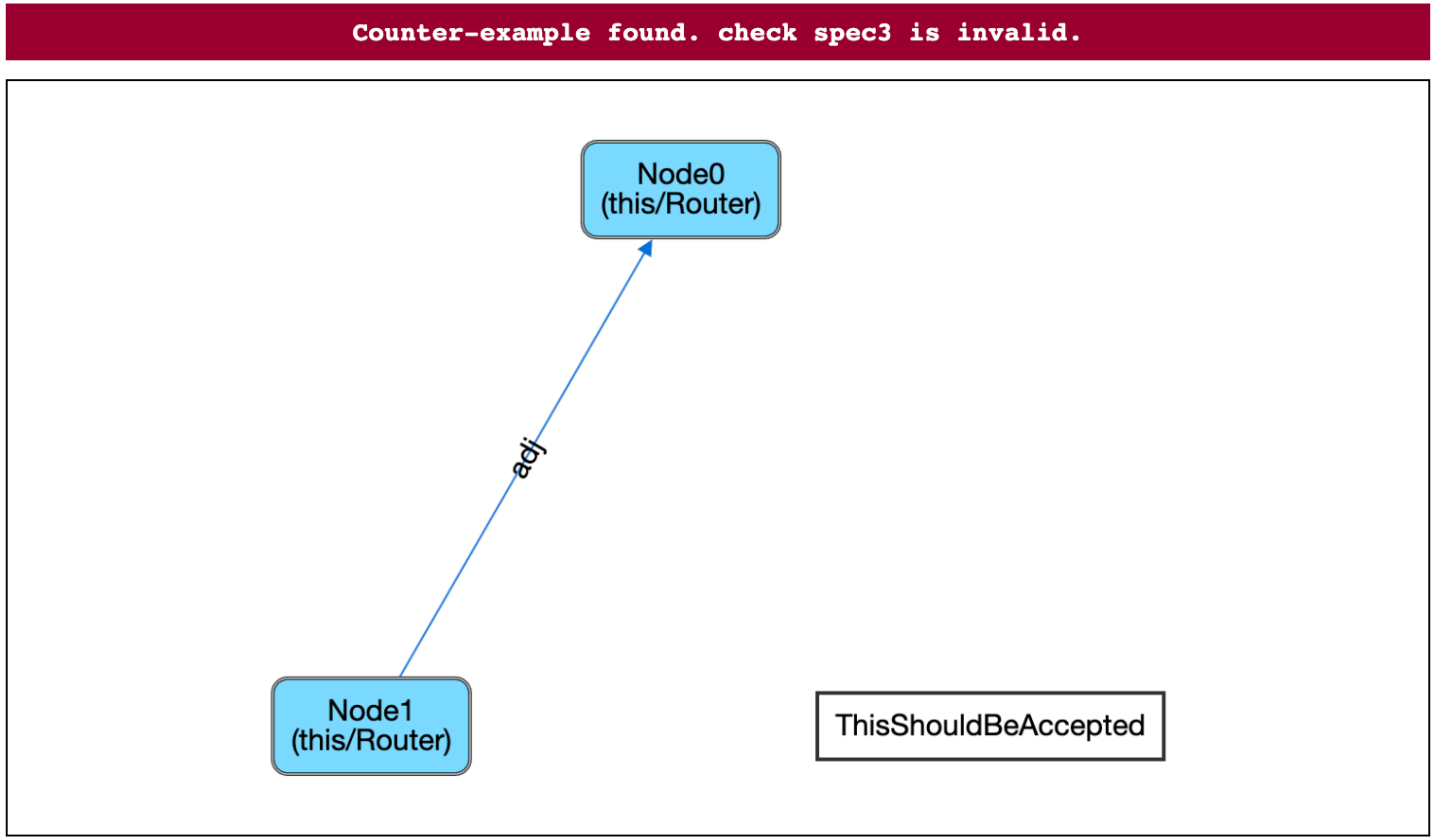
- When exercises have several challenges, students struggle to disregard previously specified properties
- We found it best to just assume previous specifications to hold
- Assumed regardless of whether the student got them right, uses the oracles

Incremental challenge assumptions

```
//SECRET
pred oracle3 {
  all n : Node | Node in n.*(~adj+adj)
}
pred spec3 {
  // the network is connected

}
//SECRET
check spec3 {
  spec3 iff oracle3
} for 4
```

Incremental challenge assumptions

```
pred spec3 {
  // the network is connected
  all n:Node | Node in n.*adj
}
```



Incremental challenge assumptions

- Recipe for challenge *N*:
  - Keep predicates `oracleN` and `specN` as before
  - Add a precondition to command `specN` to only consider counter-examples where `oracleI` holds for all *I* < *N*

Incremental assumptions: private

```
//SECRET
pred oracle3 {
  all n : Node | Node in n.*(~adj+adj)
}
pred spec3 {
  // the network is connected

}
//SECRET
check spec3 {
  oracle1 implies (spec3 iff oracle3)
} for 4
```

Incremental assumptions: private

```
//SECRET
pred oracle3 {
  all n : Node | Node in n.*(~adj+adj)
}
pred spec3 {
  // the network is connected

}
//SECRET
check spec3 {
  { some ThisShouldBeRejected iff (spec3 and not oracle3)
    oracle1
  } implies
    (spec3 iff oracle3)
}
```

Incremental assumptions: public

```
pred spec3 {
  // the network is connected
  all n:Node | Node in n.*adj
}
```

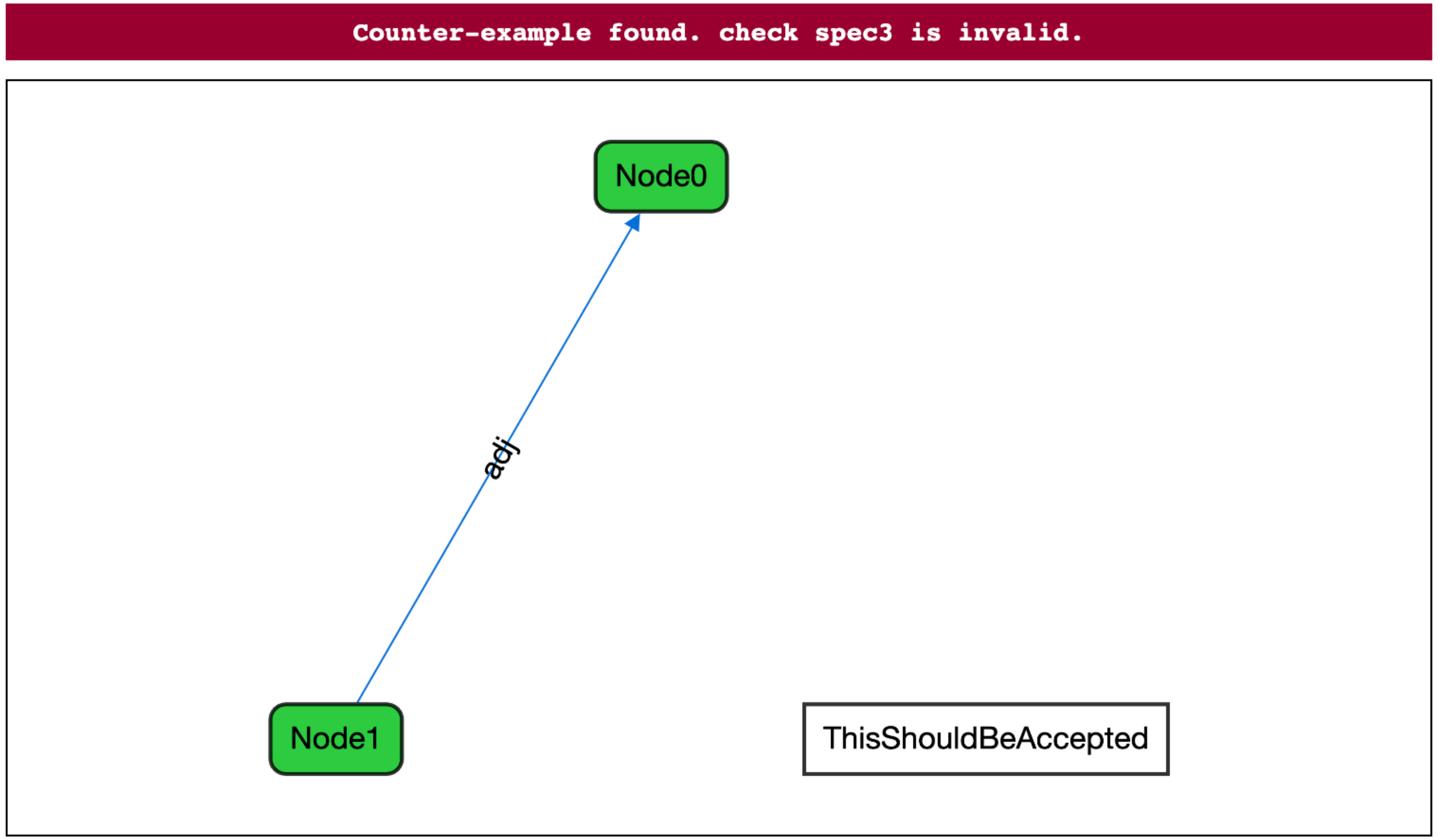
No counter-example found. check spec3 may be valid.

Incremental assumptions: private

```
//SECRET
pred oracle3 {
  all n : Node | Node in n.*(~adj+adj)
}
pred spec3 {
  // the network is connected

}
//SECRET
check spec3 {
  { some ThisShouldBeRejected iff (spec3 and not oracle3)
    oracle1
    no Router
  } implies
    (spec3 iff oracle3)
}
```

Partial feedback: public

```
pred spec3 {
  // the network is connected
  all n:Node | Node in n.*adj
}
```



Improving maintainability

- In models with several challenges, commands can be difficult to maintain
- Cannot refactor out to predicates (arguments would be formulas and not relations)
- Alloy supports `let`-macros, replaced directly during preprocessing
- No type-checking, use with care

Macro for simple challenges

```
//SECRET
let verify[s,o] {
  { some ThisShouldBeRejected iff (s and not o)
    no Subspec
  } implies (s iff o)
}

//SECRET
check spec1 {
  verify[spec1,oracle1]
}
```

Macro for simple challenges

```
//SECRET
let verifypre[p,s,o] {
   { some ThisShouldBeRejected iff (s and not o)
     no Subpsec
     p
   } implies (s iff o)
}

//SECRET
check spec3 {
   verifypre[oracle3 and no Router,spec3,oracle3]
}
```

Macro for partials

```
//SECRET
let verifypresub[p,s,o,s1,s2,s3] {
  { some ThisShouldBeRejected iff (s and not o)
    some FailsSubspec1 iff (s and not s1)
    some FailsSubspec2 iff (s and not s2)
    some FailsSubspec3 iff (s and not s3)
    p
  } implies (s iff o)
}
```

Macro for partials

```
//SECRET
check spec2 {
  verifypresub[no none,spec2,oracle2,oracle2a,oracle2b,no none]
}

//SECRET
check allSpecs {
  let oracles = spec1 and spec2 and spec3,
      specs = spec1 and spec2 and spec3 |
  verifypresub[no none,specs,oracles,oracle1,oracle2,oracle3]
}
```

Relational challenges

- Private view: http://alloy4fun.inesctec.pt/qCwrMjA9W7cuv4amm
- Public view: http://alloy4fun.inesctec.pt/mtf27hGfbwgdhxhZZ

Temporal logic challenges

- Challenges for temporal logic are particularly difficult for students
- Struggle to ignore internals of events to focus on abstract properties over traces
- To keep problems well-defined, two classes of challenges:
  - Pure temporal logic reasoning over abstract *traces*
  - Predicates relating two states encoding individual *events*

Concurrency models

- Control how events occur in the trace
  - *Interleaved execution*: only one node acts at a time, global stutter if no node acts
  - *True concurrency*: all nodes may act at the same time, each node stutters independently
- Must be made clear since expected properties depend on it

Trace property challenges

- Ask for temporal specifications over traces of abstract events
- Internals of events irrelevant: only focus on their occurrence
- Counter-examples simply show sequences of occurring events

Trace property challenges

- Create a mutable abstract `Event` for events occurring in each state
- Create sub-signatures of `Event` for each class of events
- Add parameters of events as mutable fields
- For each event *M*:
  - Define a mutable signature `EventM` extending the respective `Event` signature
  - Define predicate `EventM` to test the occurrence of event with given parameters, checks the occurrence of event atom
- Define a fact `Trace` encoding the desired concurrency model
- Mark all signatures and predicates as *`//SECRET`*
- Describe the available event API through comments
- Hide everything from the visualization leaving only the events

Trace property challenges

- Recipe for challenge *N* within a model:
  - Follow the same strategy as the relational logic challenges
  - Take care for the scope of `Event`: depends on concurrency model

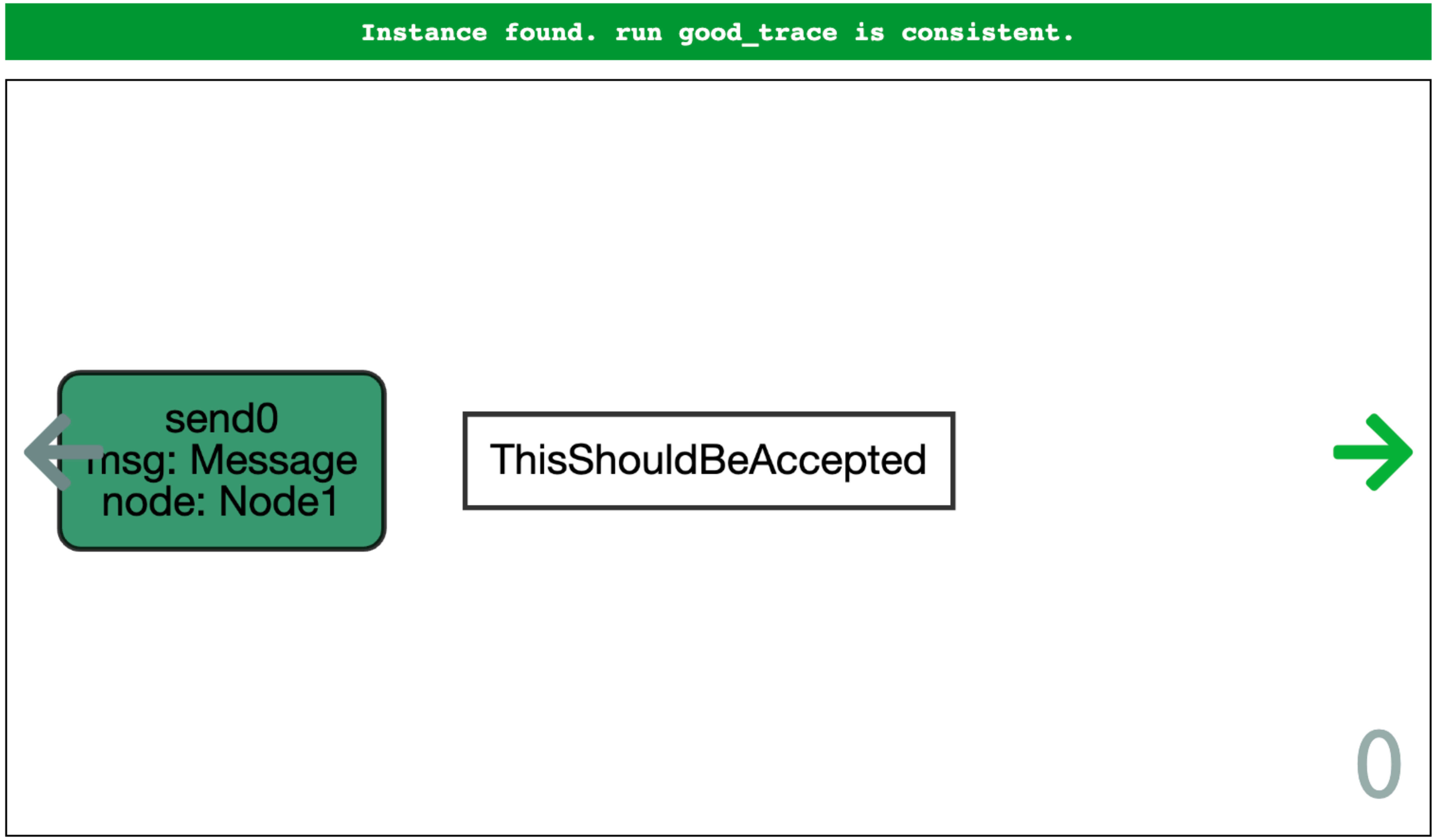
Trace challenges: interleaved

```
sig Message {}
//SECRET
var abstract sig Event {}
//SECRET
var abstract sig Action extends Event {
  var node : one Node,
  var msg  : one Message }
//SECRET
var sig send, receive extends Action {}
//SECRET
var lone sig stutter extends Event {}
//SECRET
pred send[n : Node, m : Message] {
  some a : send | a.node = n and a.msg = m }
//SECRET
pred receive[n : Node, m : Message] {
  some a : receive | a.node = n and a.msg = m }
//SECRET
pred stutter {
  some stutter }
```

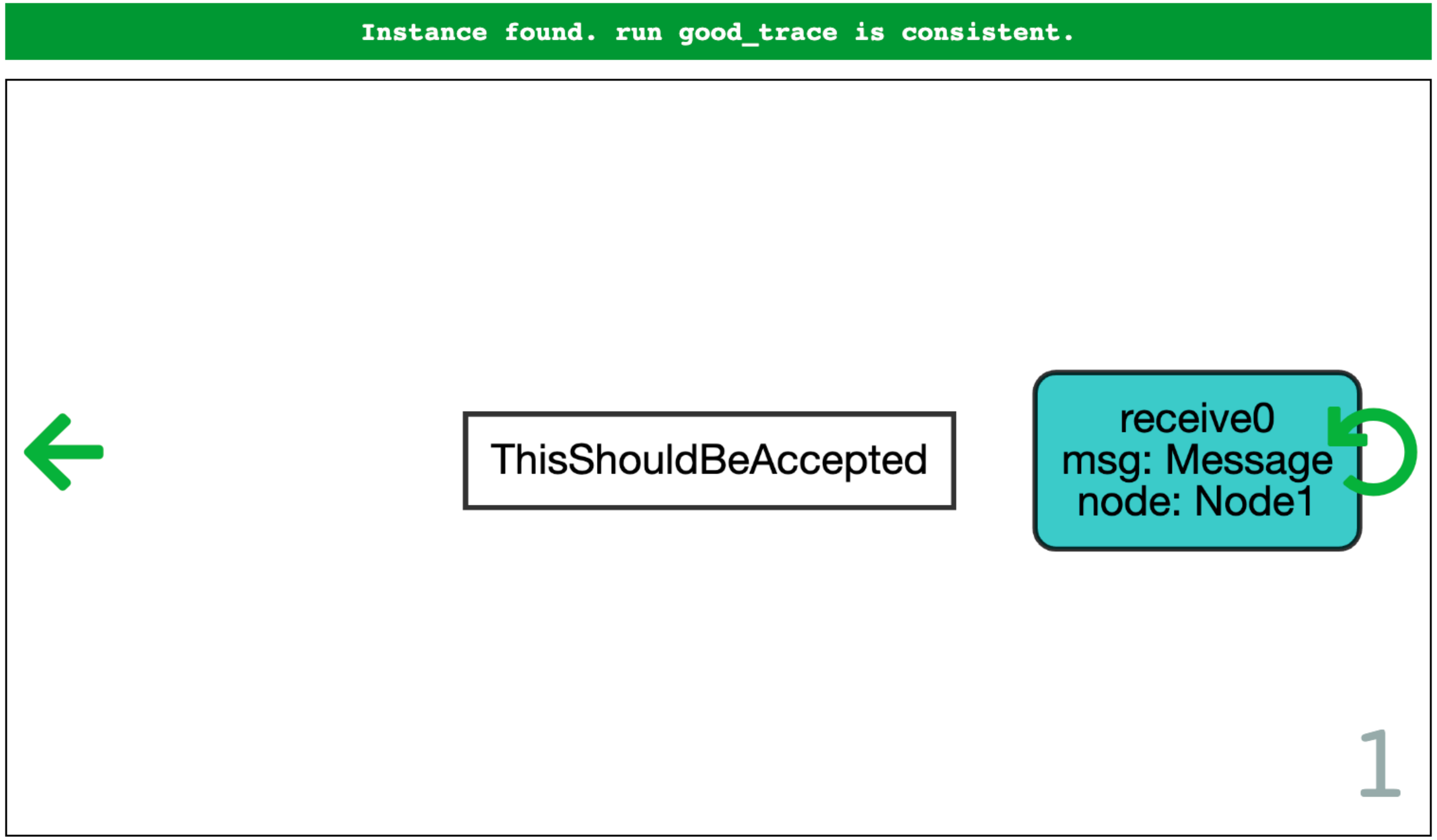
Trace challenges: interleaved

```
//SECRET
fact Trace {
  always one Event
}

/* Assume the existence of the following events, and that
   only one may happen at each state:
   pred send[n : Node, m : Message] {...}
   pred receive[n : Node, m : Message] {...}
   pred stutter {...} */
```

Trace challenges: public



Trace challenges: public

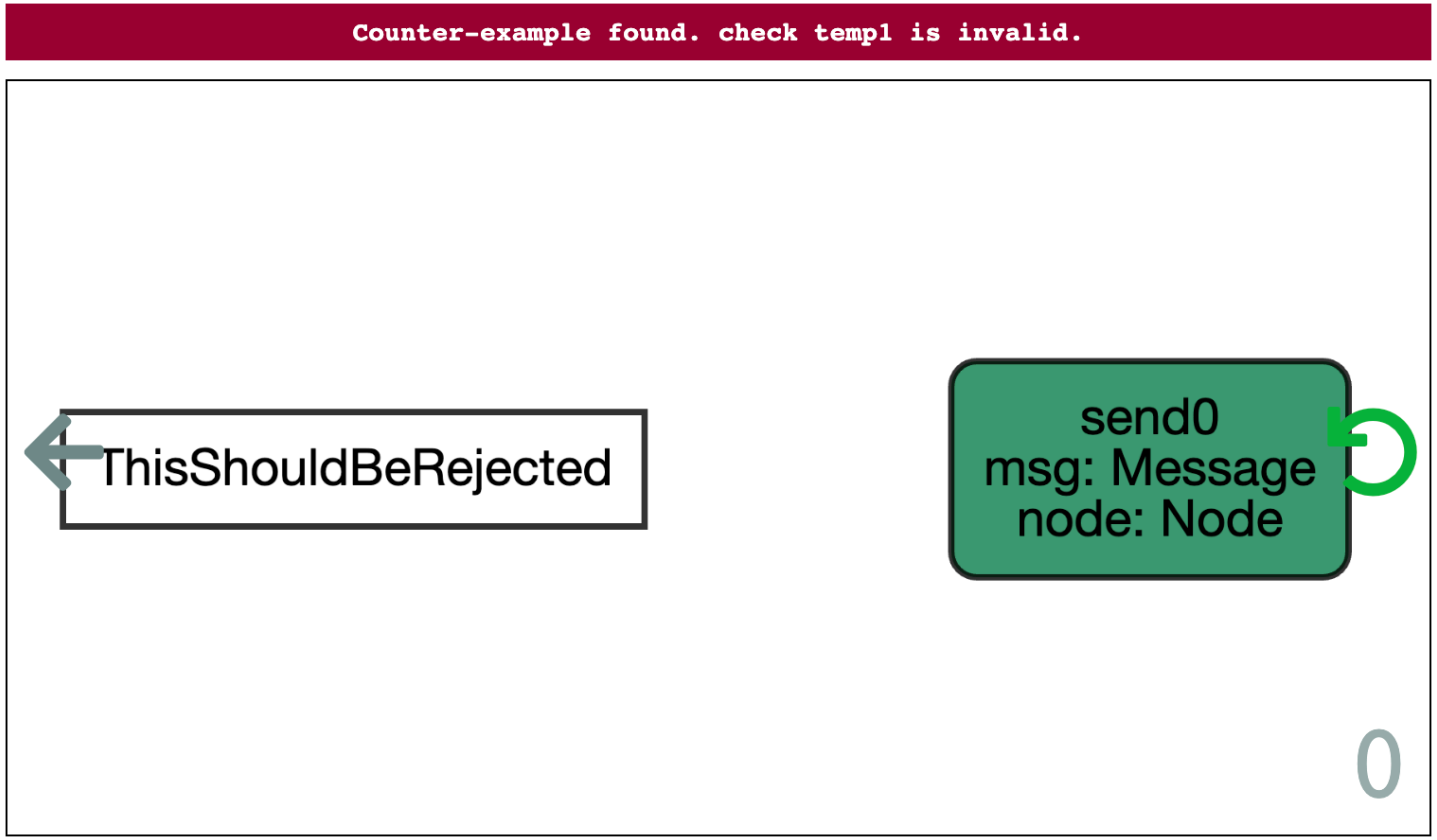


Trace challenges: interleaved

```
//SECRET
pred toracle1 {
  always stutter
}
pred temp1 {
  // nothing will ever happen

}
//SECRET
check temp1 {
  verify[temp1,toracle1]
} for 2 but 3 Event
```

Trace challenges: public

```
pred temp1 {
  // nothing will ever happen

}
```



Trace challenges: interleaved

```
//SECRET
pred toracle2 {
  all n : Node, m : Message |
    always (receive[n,m] implies before once some f : Node | send[f,m])
}
pred temp2 {
  // any received message must have been sent before

}
//SECRET
check temp2 {
  verify[temp2, toracle2]
} for 2 but 3 Event
```

Trace challenges: public

```
pred temp2 {
  // any received message must have been sent before
  all n : Node, m : Message |
    always (receive[n,m] implies before once some f : Node | send[f,m])
}
```

No counter-example found. check temp2 may be valid.

Trace challenges: public

```
pred temp2 {
  // any received message must have been sent before
  all n : Node, m : Message |
    always (receive[n,m] implies once some f : Node | send[f,m])
}
```

No counter-example found. check temp2 may be valid.

Trace challenges: interleaved

- Private view: http://alloy4fun.inesctec.pt/GxKTndgdDTxewzX8X
- Public view: http://alloy4fun.inesctec.pt/M65cdRJE4Jci2nnKY

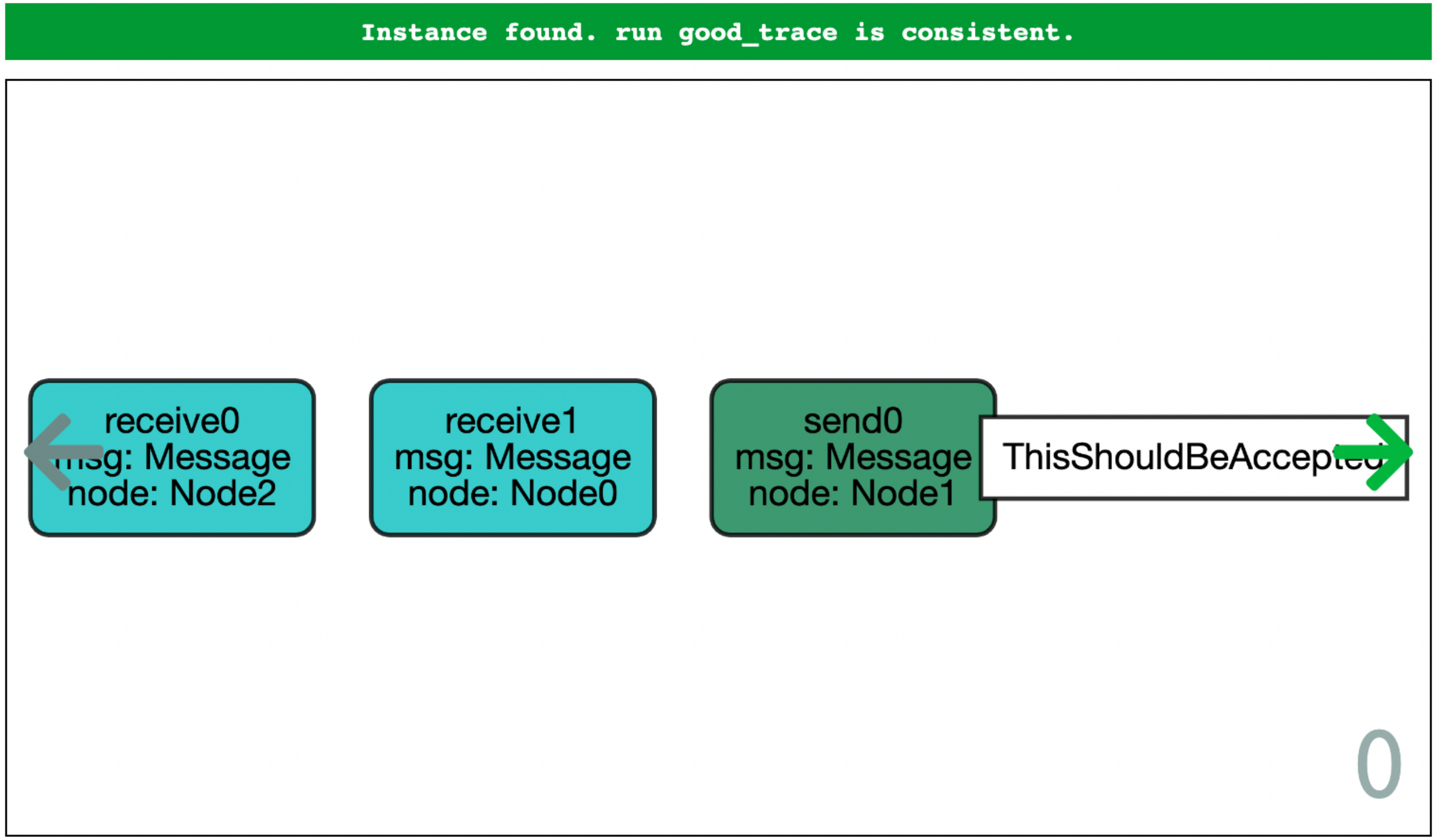
Trace challenges: concurrent

```
sig Message {}
//SECRET
var abstract sig Event {
  var node : one Node }
//SECRET
var abstract sig Action extends Event {
  var msg  : one Message }
//SECRET
var sig send, receive extends Action {}
//SECRET
var lone sig stutter extends Event {}
//SECRET
pred send[n : Node, m : Message] {
  some a : send | a.node = n and a.msg = m }
//SECRET
pred receive[n : Node, m : Message] {
  some a : receive | a.node = n and a.msg = m }
//SECRET
pred stutter[n : Node] {
  some a : stutter | a.node = n }
```

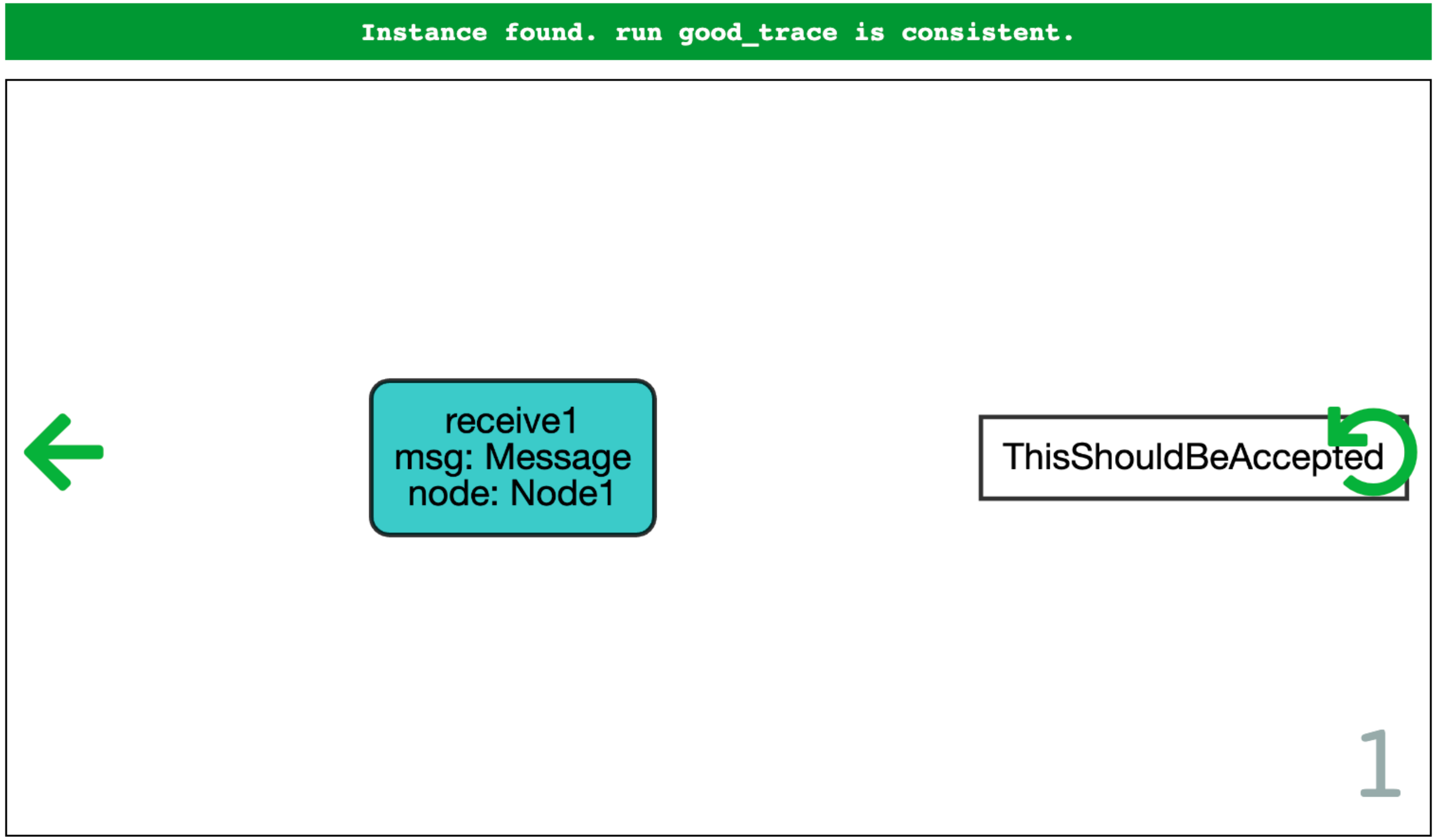
Trace challenges: concurrent

```
//SECRET
fact Trace {
  always all n : Node | one node.n
}

/* Assume the existence of the following events, and that
   for each node one event happens at each state:
   pred send[n : Node, m : Message] {...}
   pred receive[n : Node, m : Message] {...}
   pred stutter[n : Node] {...} */
```

Trace challenges: public



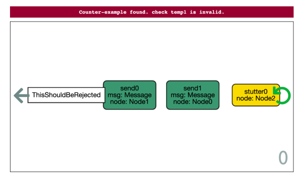
Trace challenges: public



Trace challenges: concurrent

```
//SECRET
pred toracle1 {
  all n : Node | always stutter[n]
}
pred temp1 {
  // nothing will ever happen

}
//SECRET
check temp1 {
  verify[temp1,toracle1]
} for 2 but 6 Event
```

Trace challenges: public



Trace challenges: concurrent

```
//SECRET
pred toracle2 {
  all n : Node, m : Message |
    always (receive[n,m] implies before once some f : Node | send[f,m])
}
pred temp2 {
  // any received message must have been sent before

}
//SECRET
check temp2 {
  verify[temp2, toracle2]
} for 2 but 6 Event
```

Trace challenges: public

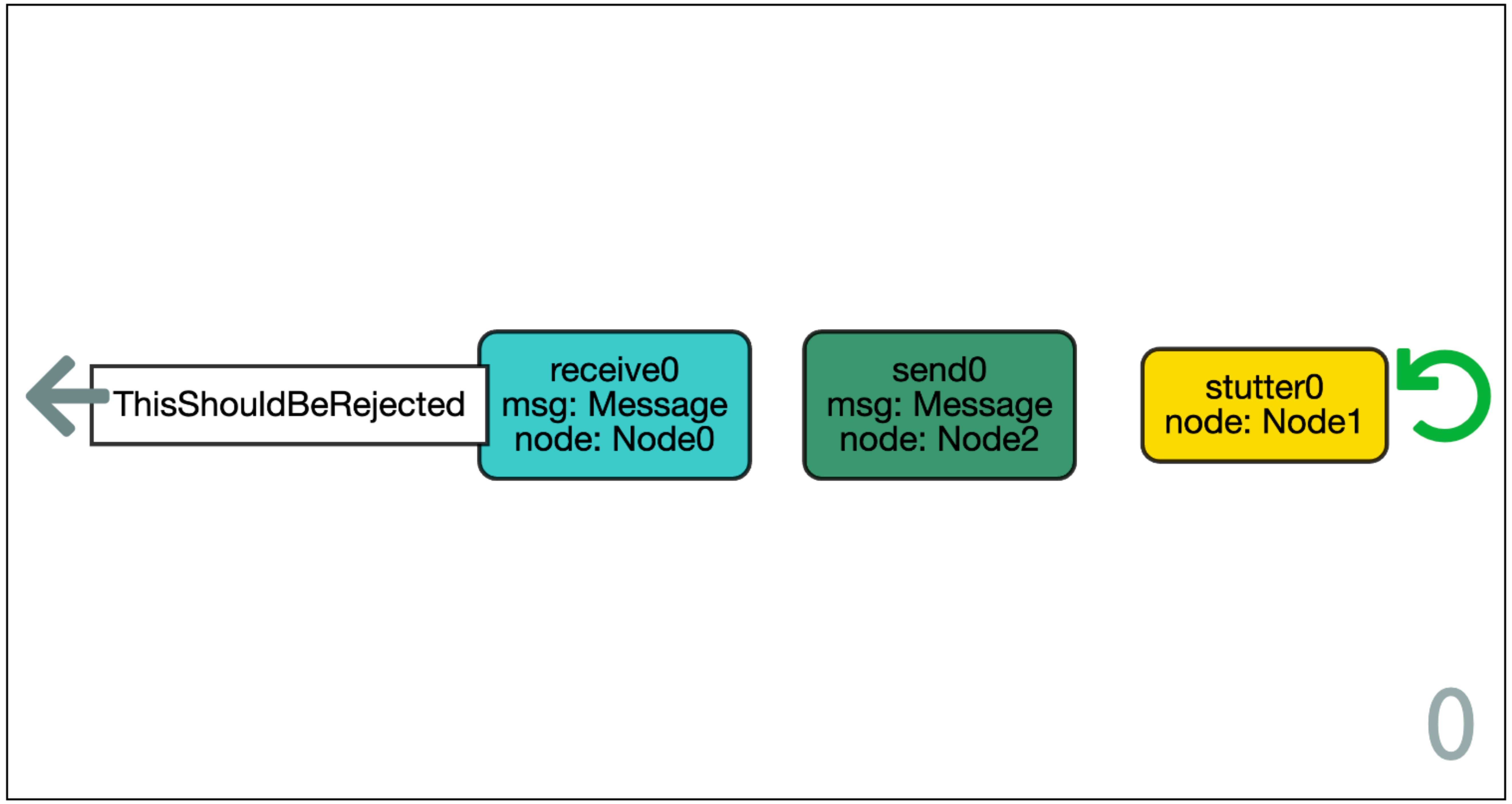
```
pred temp2 {
  // any received message must have been sent before
  all n : Node, m : Message |
    always (receive[n,m] implies before once some f : Node | send[f,m])
}
```

No counter-example found. check temp2 may be valid.

Trace challenges: public

```
pred temp2 {
  // any received message must have been sent before
  all n : Node, m : Message |
    always (receive[n,m] implies once some f : Node | send[f,m])
}
```

Counter-example found. check temp2 is invalid.



Trace challenges: concurrent

- Private view: http://alloy4fun.inesctec.pt/vEBcedmNSJqhA9kab
- Public view: http://alloy4fun.inesctec.pt/PZSCFT28pREZCQASX

Event definition challenges

- Must now consider the internal *mutable state* of the system
- Contrast to trace challenges: check the specification of a single event
- Valid sequence of events not enforced: must specify an *invariant* characterizing reachable states to avoid meaningless counter-examples
- Distinguished elements declared to represent parameters to help counter-example interpretation

Event definition challenges

- Add internal mutable state to the system's elements
- Define a predicate `inv` that represents valid states of the system
- For each challenge for event *M*:
  - Define the specification and oracle predicates as before, taking into consideration concurrency model
  - The check must now:
    - Consider only pre-states where the invariant holds
    - Declare singletons signatures for the arguments
    - Force the event to occur for those singletons, and all others to stutter
    - Restrict the **`steps`** scope to 2

Event definition challenges

```
enum State { Active, Inactive }
sig Node {
  adj : set Node,
  var inbox : set Message,
  var state : one State
}

pred inv {
  always all n : Node | n.state = Inactive implies no n.inbox
}
```

Event challenges: interleaved

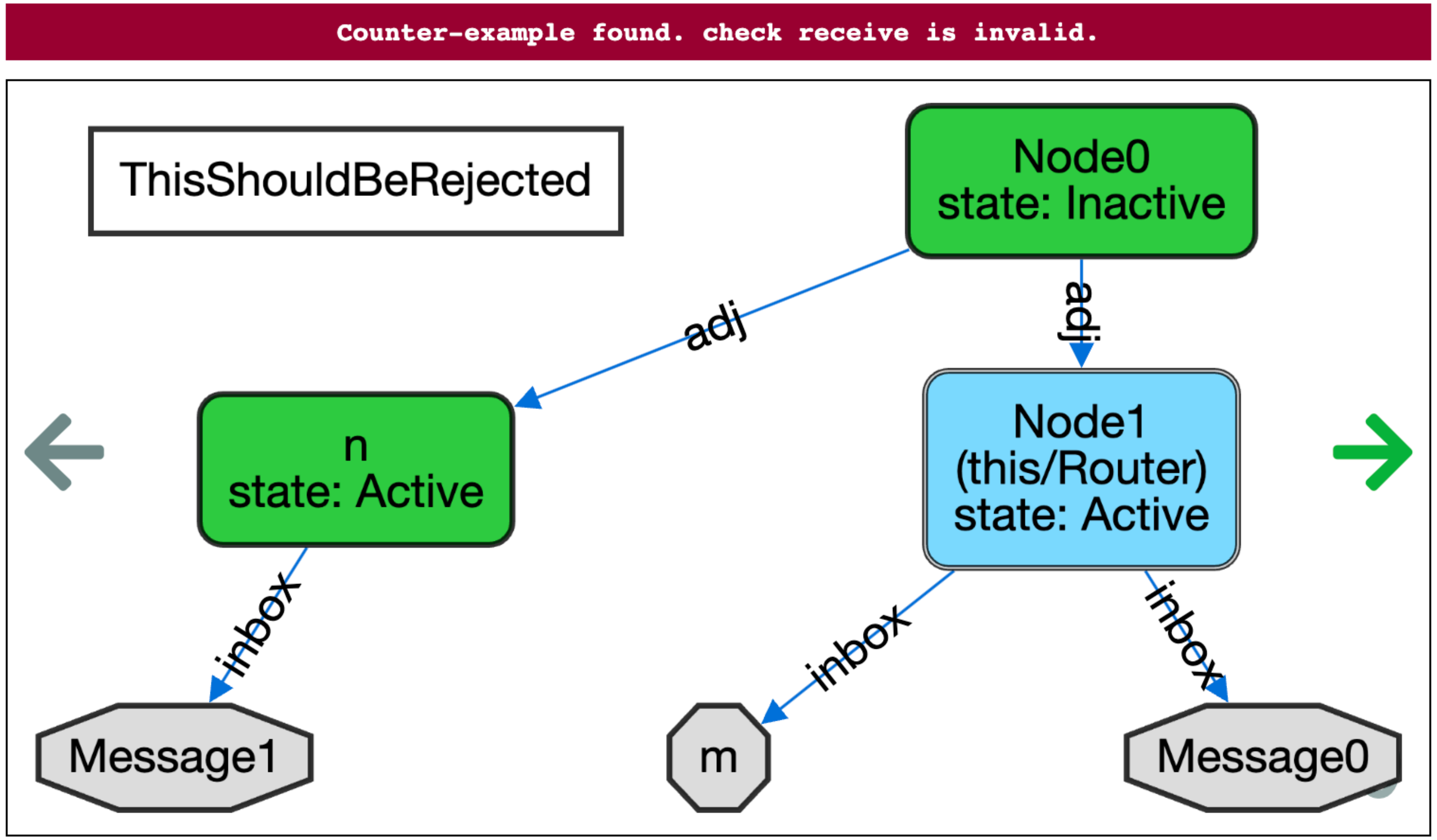
```
//SECRET
pred receiveoracle[n : Node, m : Message] {
  n.state = Active
  inbox' = inbox + n->m
  state' = state
}
pred receive[n : Node, m : Message] {
  // add the message to the inbox if active

}
//SECRET
pred stutter {
  state’ = state
  inbox’ = inbox
}
```

Event challenges: interleaved

```
//SECRET
one sig n extends Node {}
//SECRET
one sig m extends Message {}
//SECRET
check receive {
  verifypre[inv,receive[n,m]receiveoracle[n,m]]
} for 3 but 2 steps
```

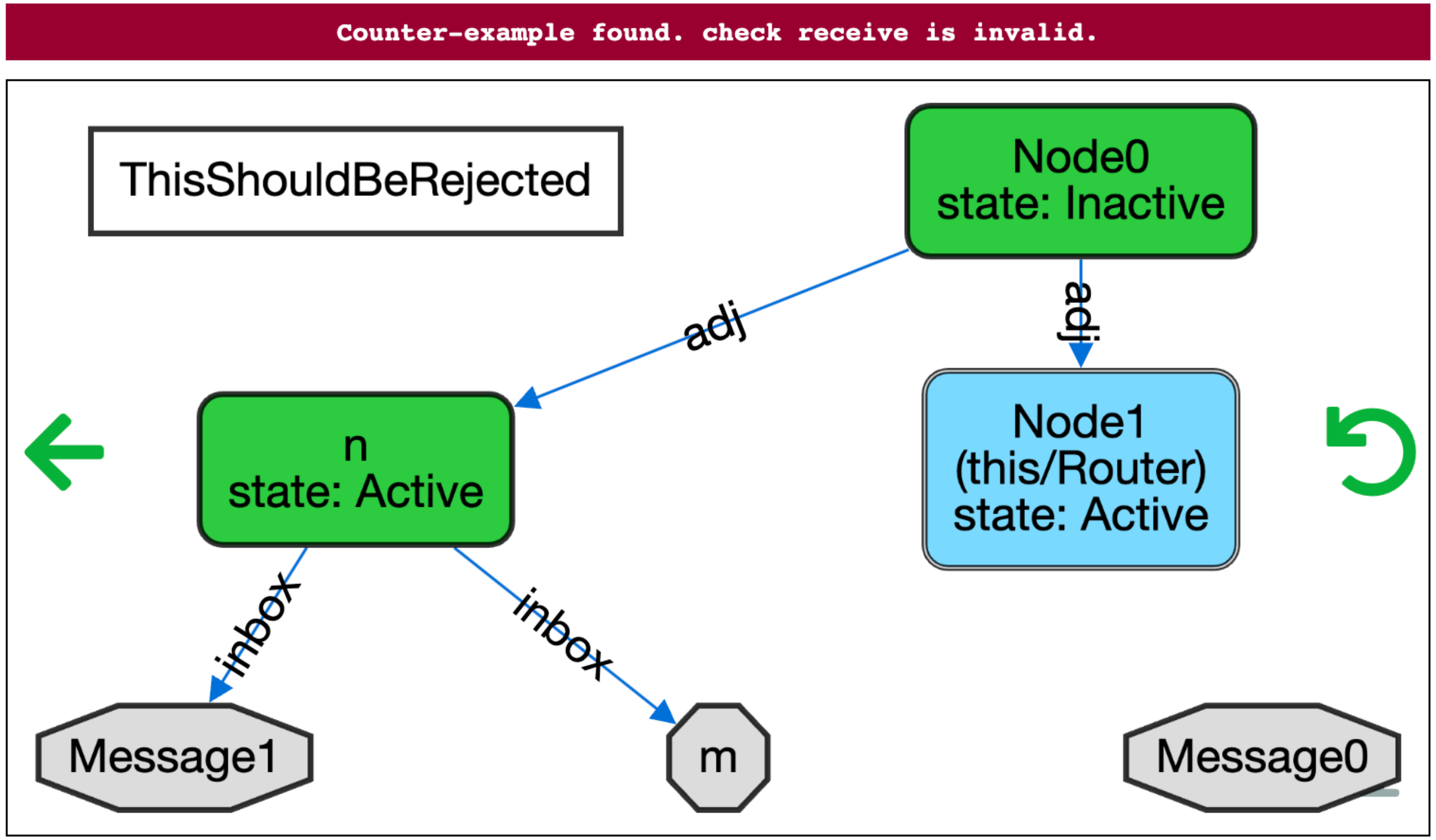
Event challenges: visualization

```
pred receive[n : Node, m : Message] {
  // add the message to the inbox if active
  n.state = Active
  n.inbox' = n.inbox +  m
  n.state' = n.state }
```



Event challenges: visualization

```
pred receive[n : Node, m : Message] {
  // add the message to the inbox if active
  n.state = Active
  n.inbox' = n.inbox +  m
  n.state' = n.state }
```



Event challenges: interleaved

- Private view: http://alloy4fun.inesctec.pt/jbdrBFtb6NibboKPE
- Public view: http://alloy4fun.inesctec.pt/SEYtemwhLRTAzLZEP

Event challenges: concurrent

```
//SECRET
pred receiveoracle[n : Node, m : Message] {
  n.state = Active
  n.inbox' = n.inbox + m
  n.state' = n.state
}
pred receive[n : Node, m : Message] {
  // add the message to the inbox if active

}
//SECRET
pred stutter[n : Node] {
  n.state’ = n.state
  n.inbox’ = n.inbox
}
```

Event challenges: concurrent

```
//SECRET
one sig n extends Node {}
//SECRET
one sig m extends Message {}
//SECRET
check receive {
  verifypre[inv and all f : Node-n | stutter[f],receive[n,m],receiveoracle[n,m]]
} for 3 but 2 steps
```

Event challenges: visualization

```
pred receive[n : Node, m : Message] {
  // add the message to the inbox if active
  n.state = Active
  n.inbox' = n.inbox +  m
  n.state’ = n.state }
```

No counter-example found. check receive may be valid.

Event challenges: concurrent

- Private view: http://alloy4fun.inesctec.pt/TYvixjj36NoW4GBtS
- Public view: http://alloy4fun.inesctec.pt/E4XajuEs5a2u4LLfy

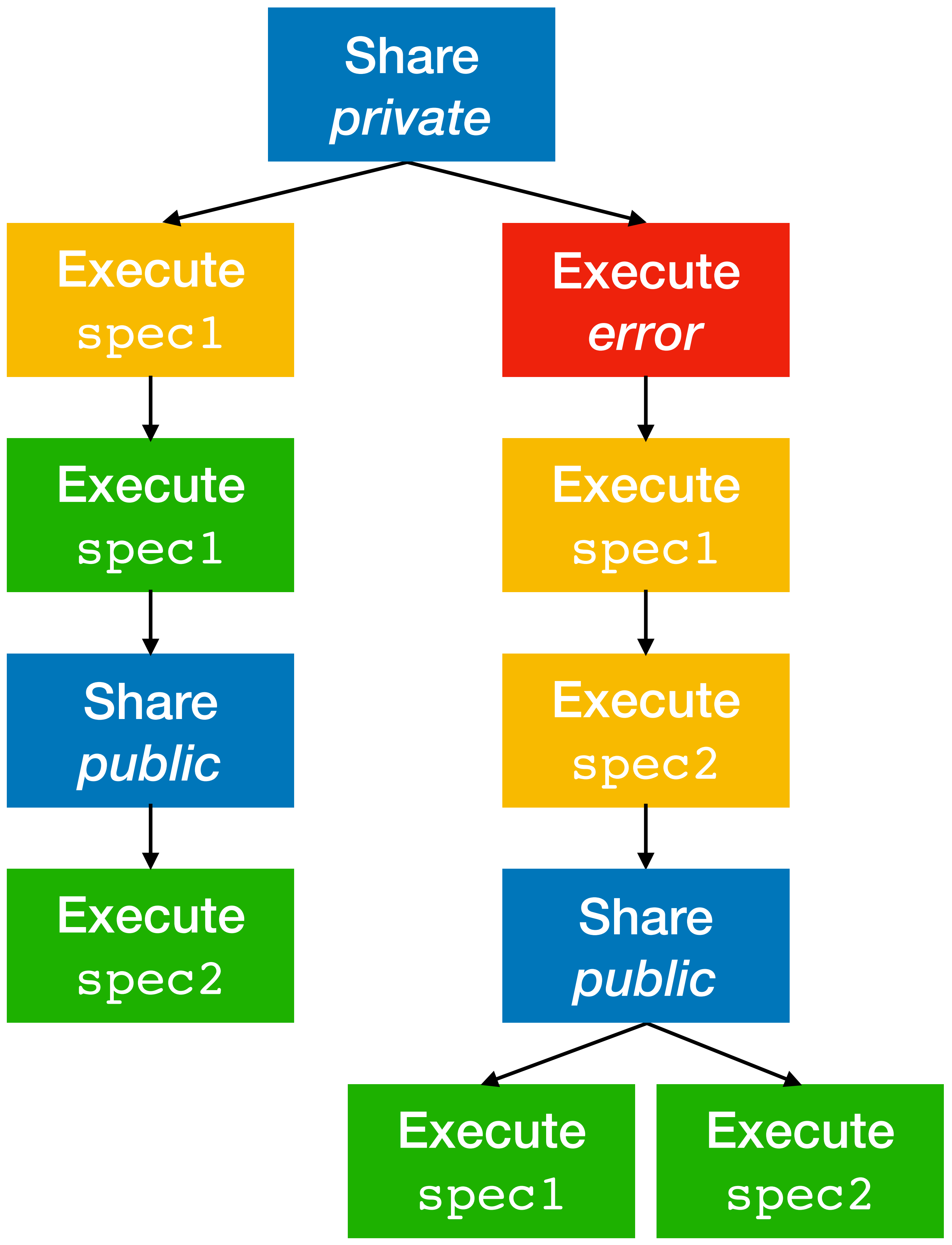
Analyzing results

Data collection

- Alloy4Fun collects (anonymous) information about all interactions
- Owners of a model with secrets can access all submissions to the public permalink
- Useful for
  - *lecturers* to keep track of progress during classes
  - *researchers* to perform studies on formal specification
- Some tools provided to ease analysis of data

Data model

- Alloy4Fun organizes data in *derivation trees*
- Each node is an executed or shared model
- The parent is the previously registered state
- Shared models have a children for each access
- The root is the original model with secrets
- Each children of the root is a *session*, usually a series of attempts by a participant



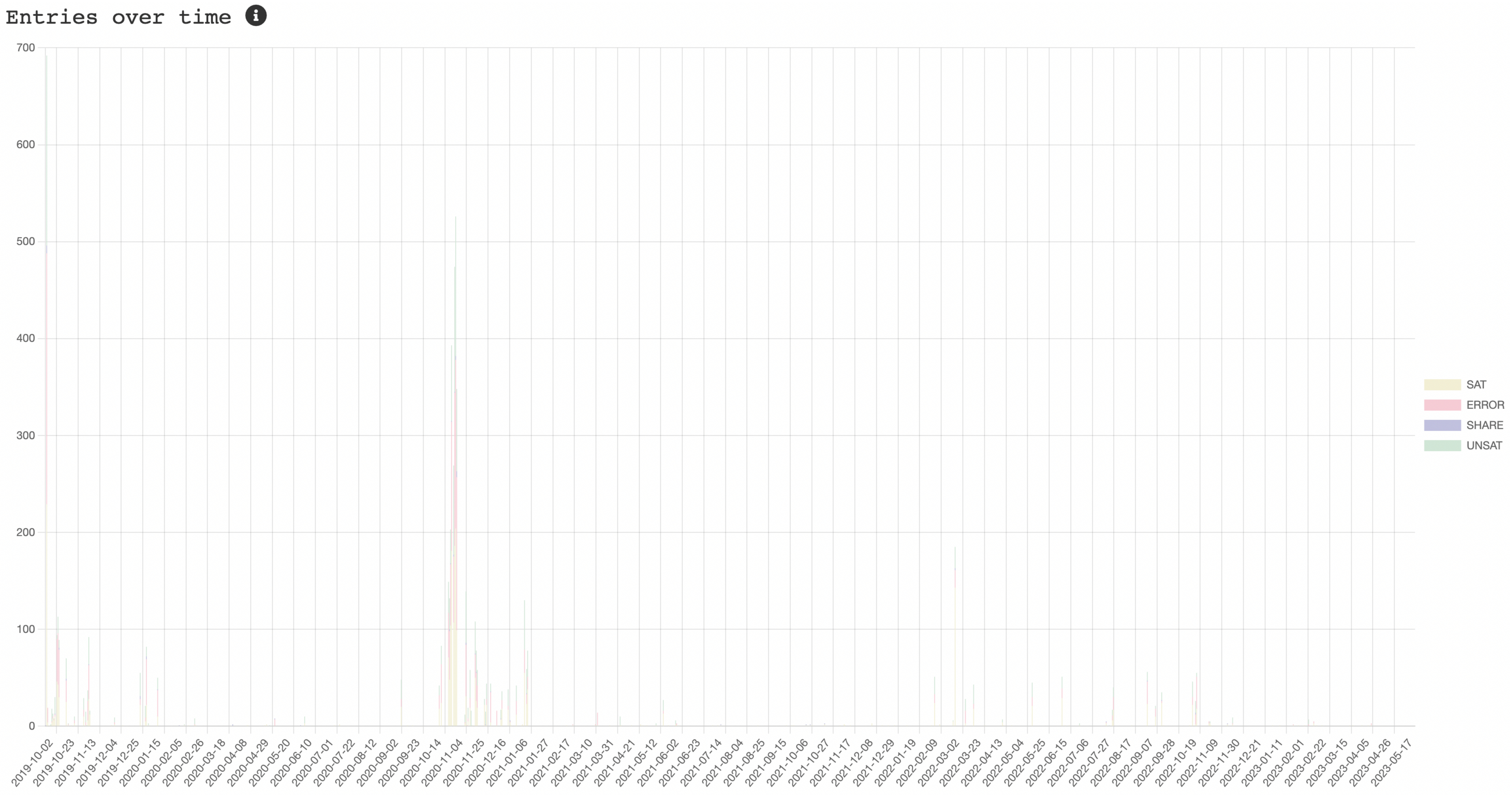
Automatic statistics

- When accessing a private view of a model with secrets, some statistics can be inspected for its derivation tree 📊
- Quick insights about submissions to the model
- “Challenges” automatically detected:
    - check commands which call an empty predicate

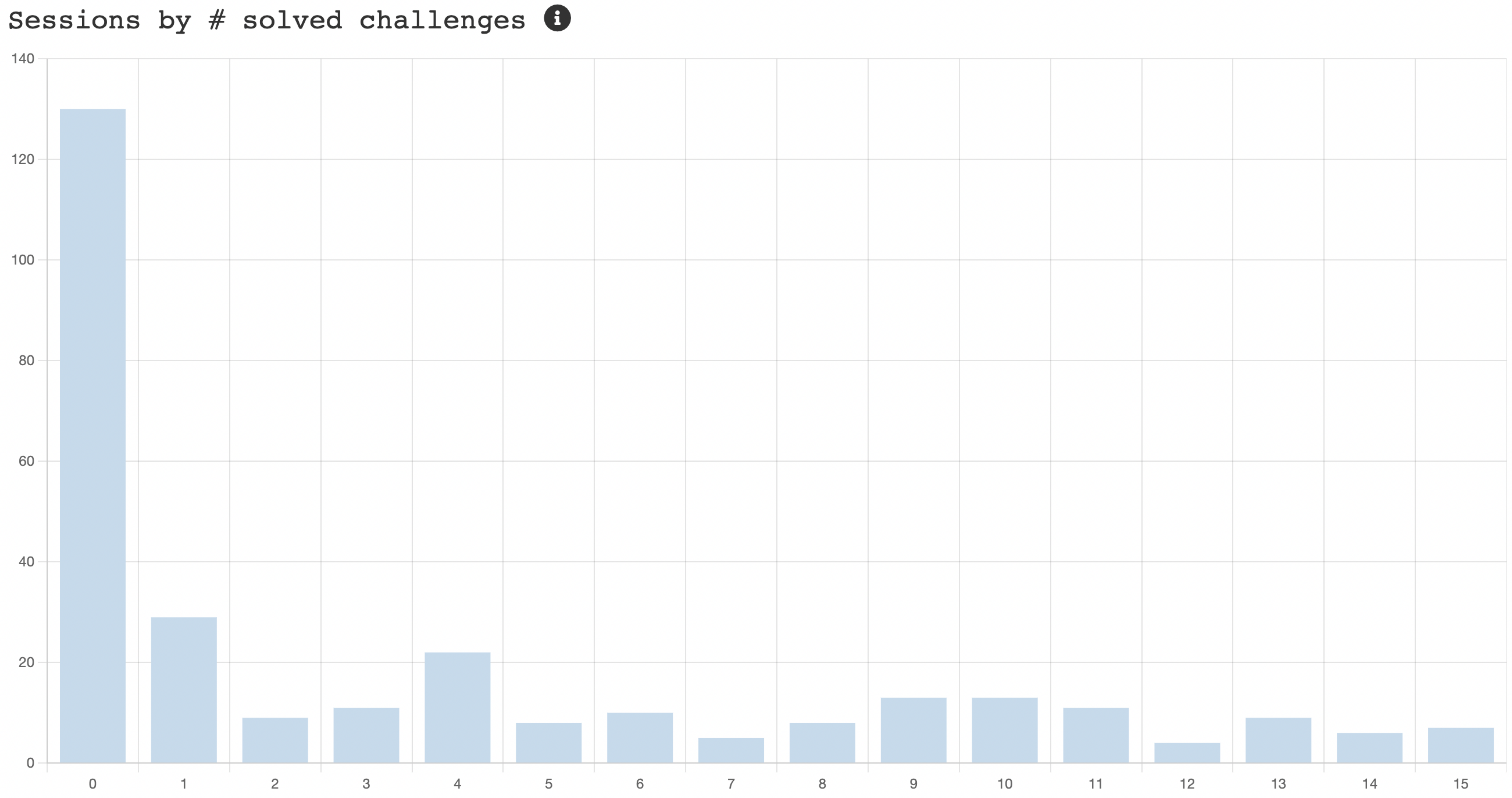
Statistics: overview

Challenge name	courses	**Extraction time**	2023-05-26T15:43:09.109
# sub-challenges	15	**Metric catalog**	Simple online metrics
Total sessions ℹ	295	**Longest session** ℹ	207
Average session ℹ	20.66	**Average % unsatisfiable** ℹ	0.33
Sessions all solved ℹ	7	**Average length all solved** ℹ	68.43
Total executions ℹ	6157	**Challenge executions** ℹ	4164
Sat executions ℹ	2458	**Unsat executions** ℹ	1708
Error executions ℹ	1991	**Compile-time errors** ℹ	1940
Total warnings ℹ	1	**Unsat executions w/ warnings** ℹ	48
Sat executions w/ warnings ℹ	365	**Error executions w/ warnings** ℹ	20
Number of shared sessions ℹ	48	**Total shares** ℹ	65
Shared models ℹ	62	**Shared instances** ℹ	3
Number of iterations ℹ	0	**Average iterations** ℹ	0

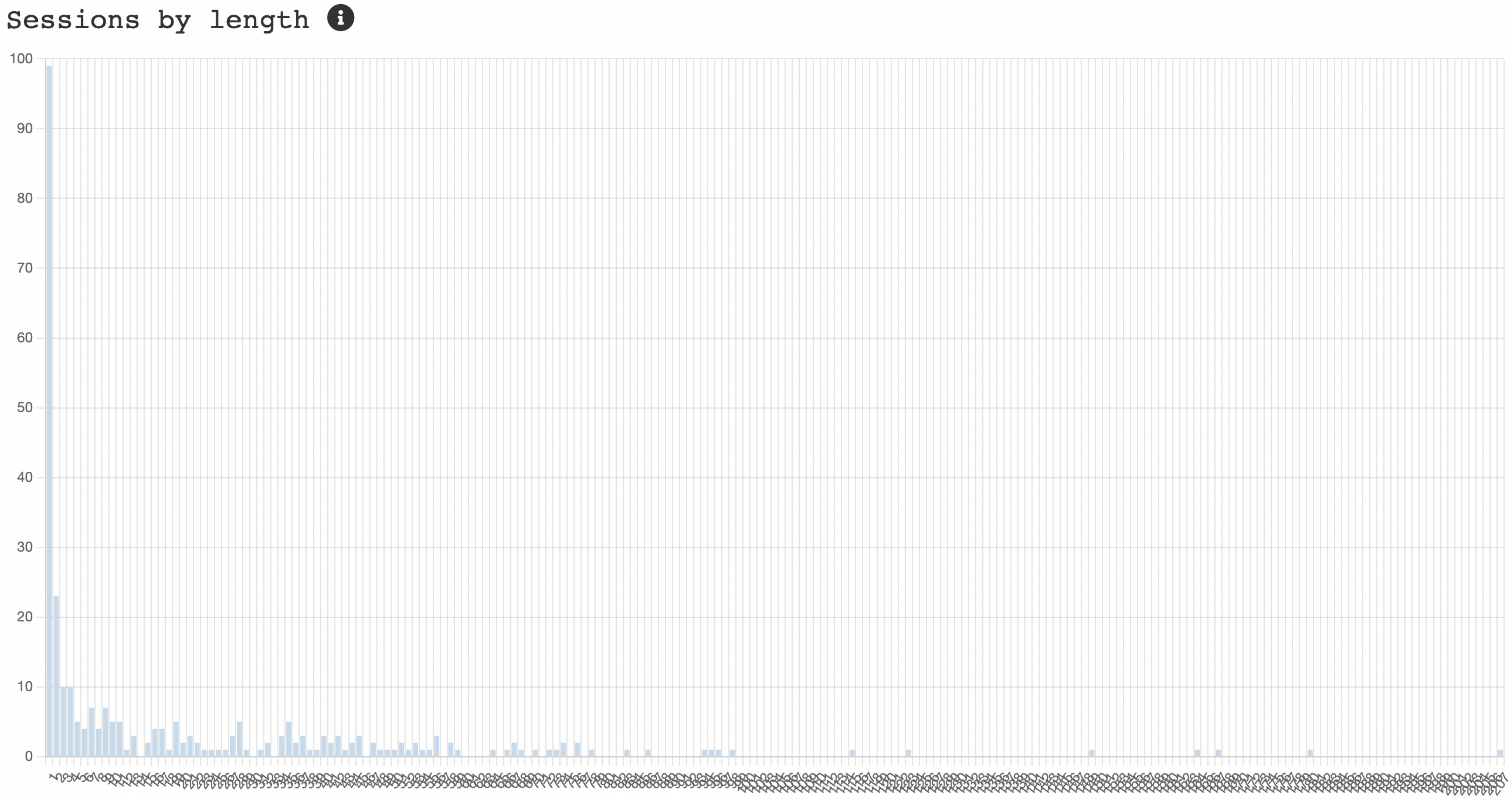
Statistics: results over time



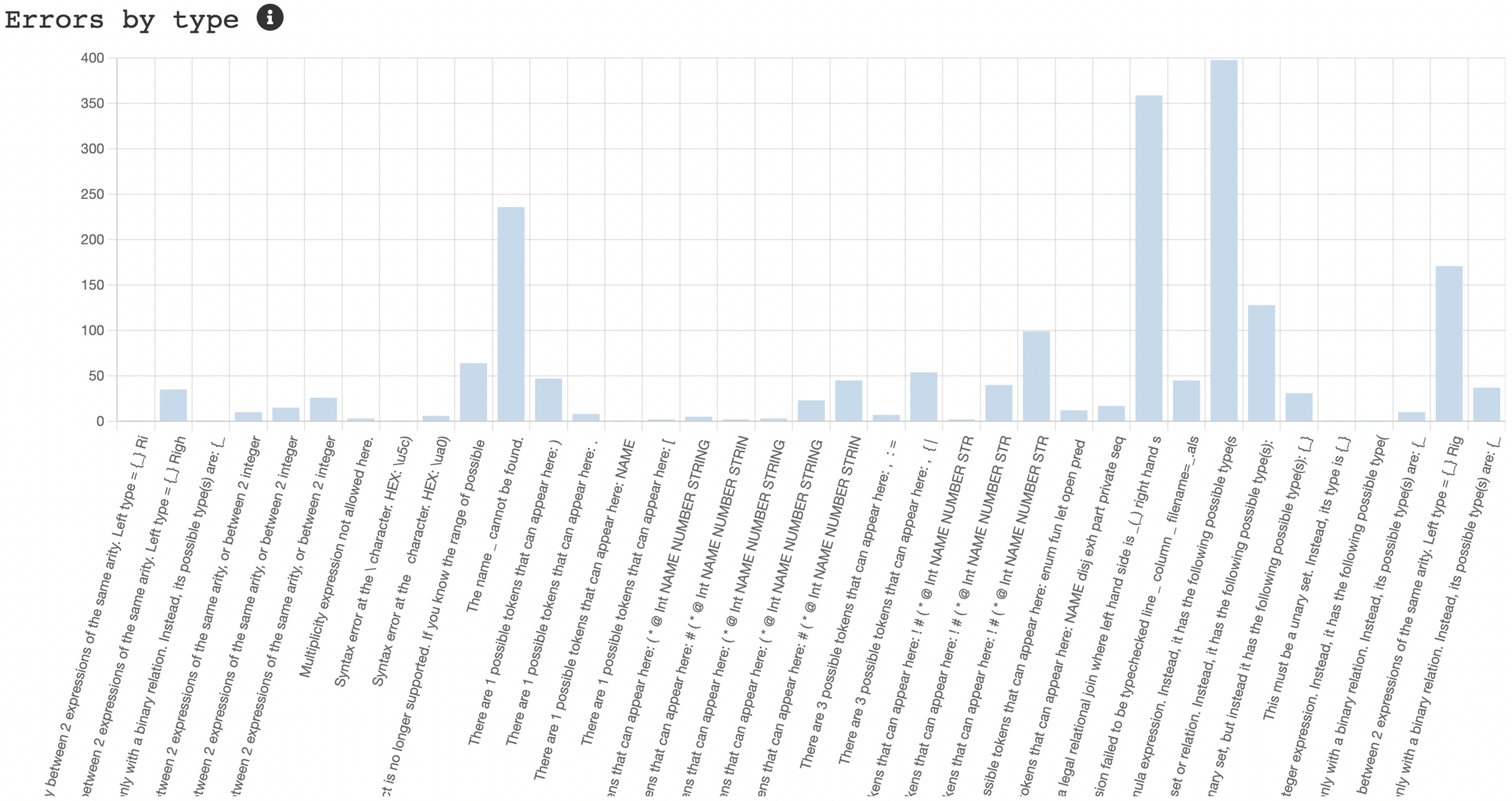
Statistics: sessions



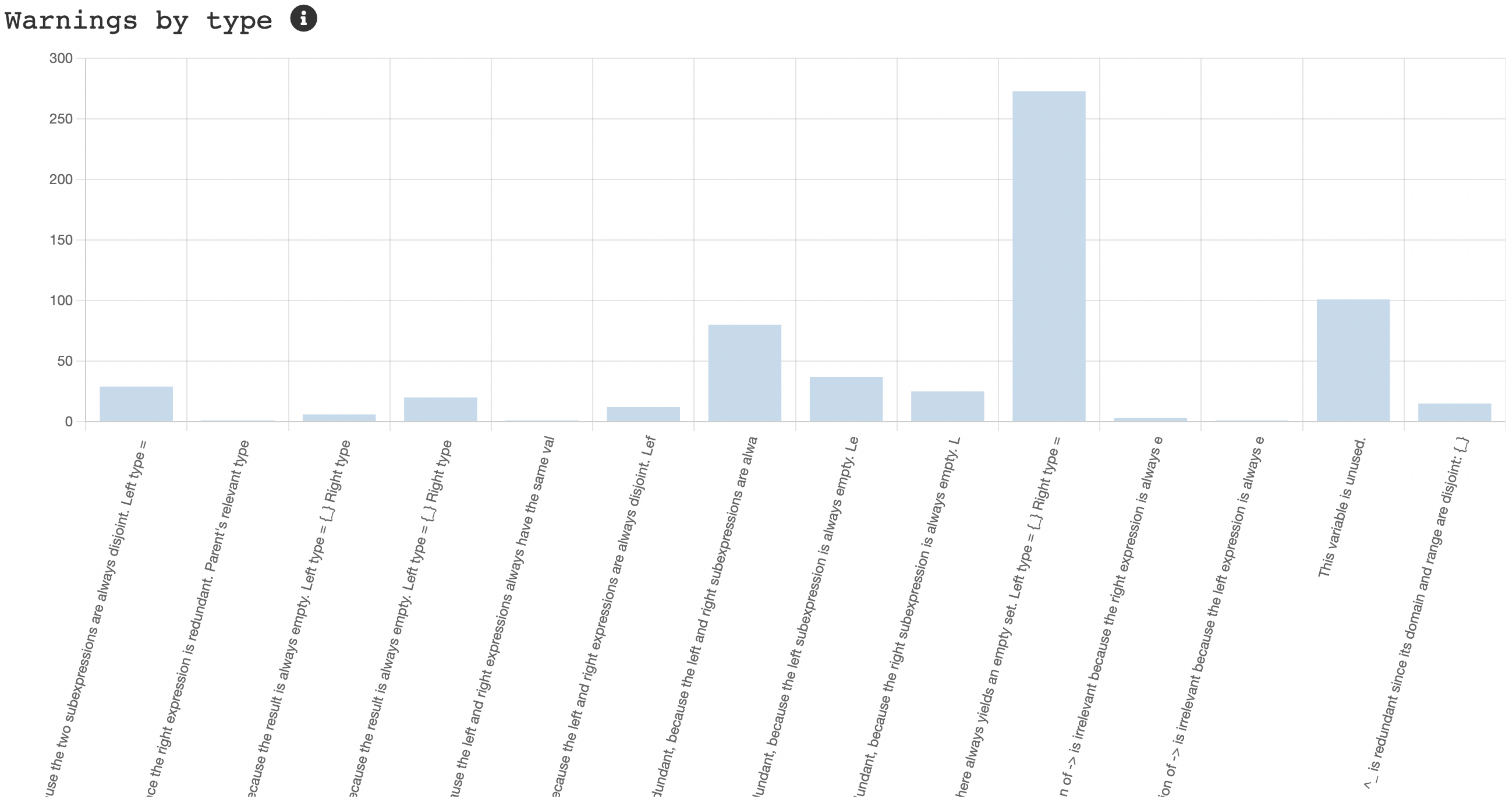
Statistics: sessions



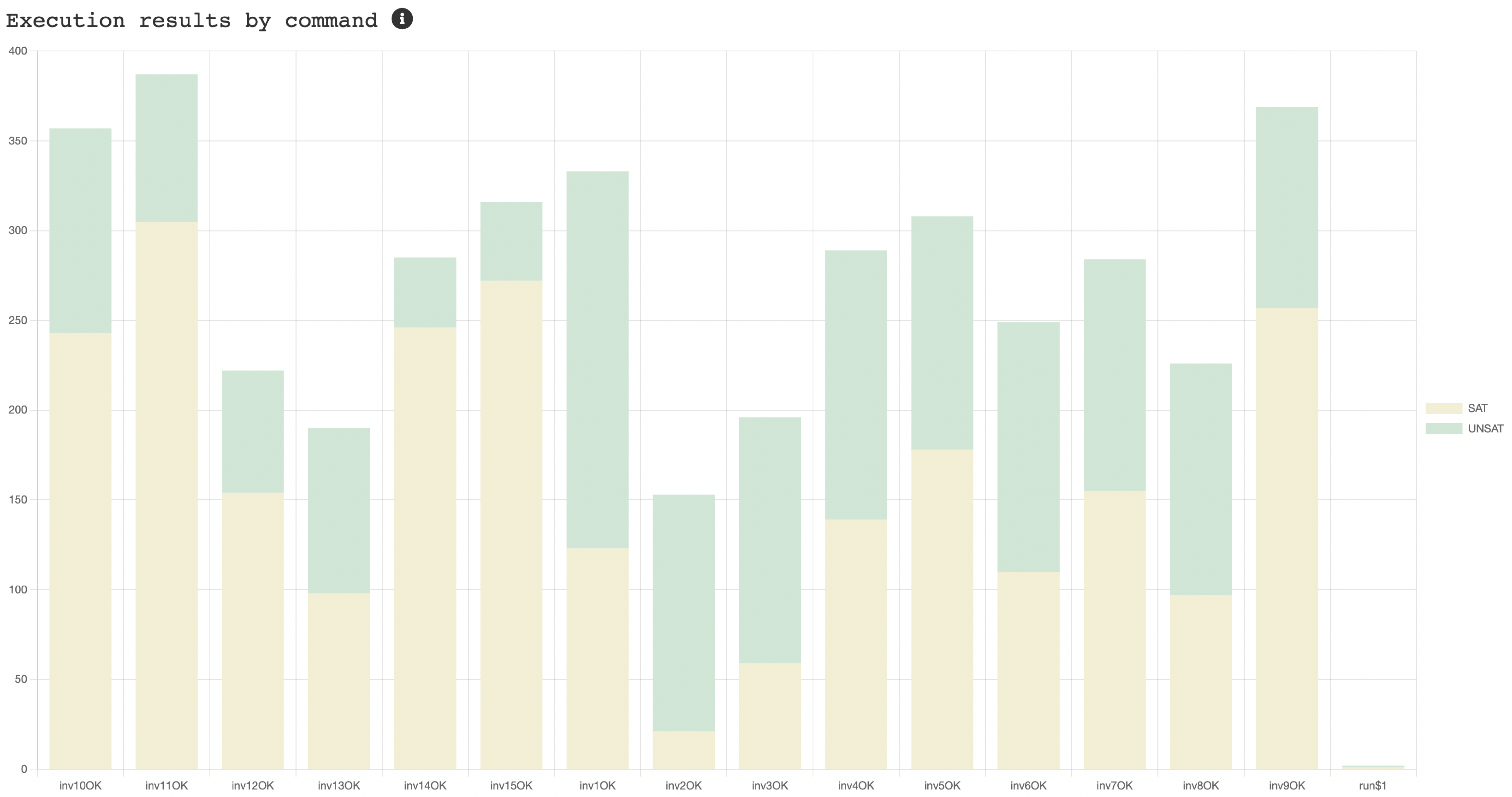
Statistics: errors



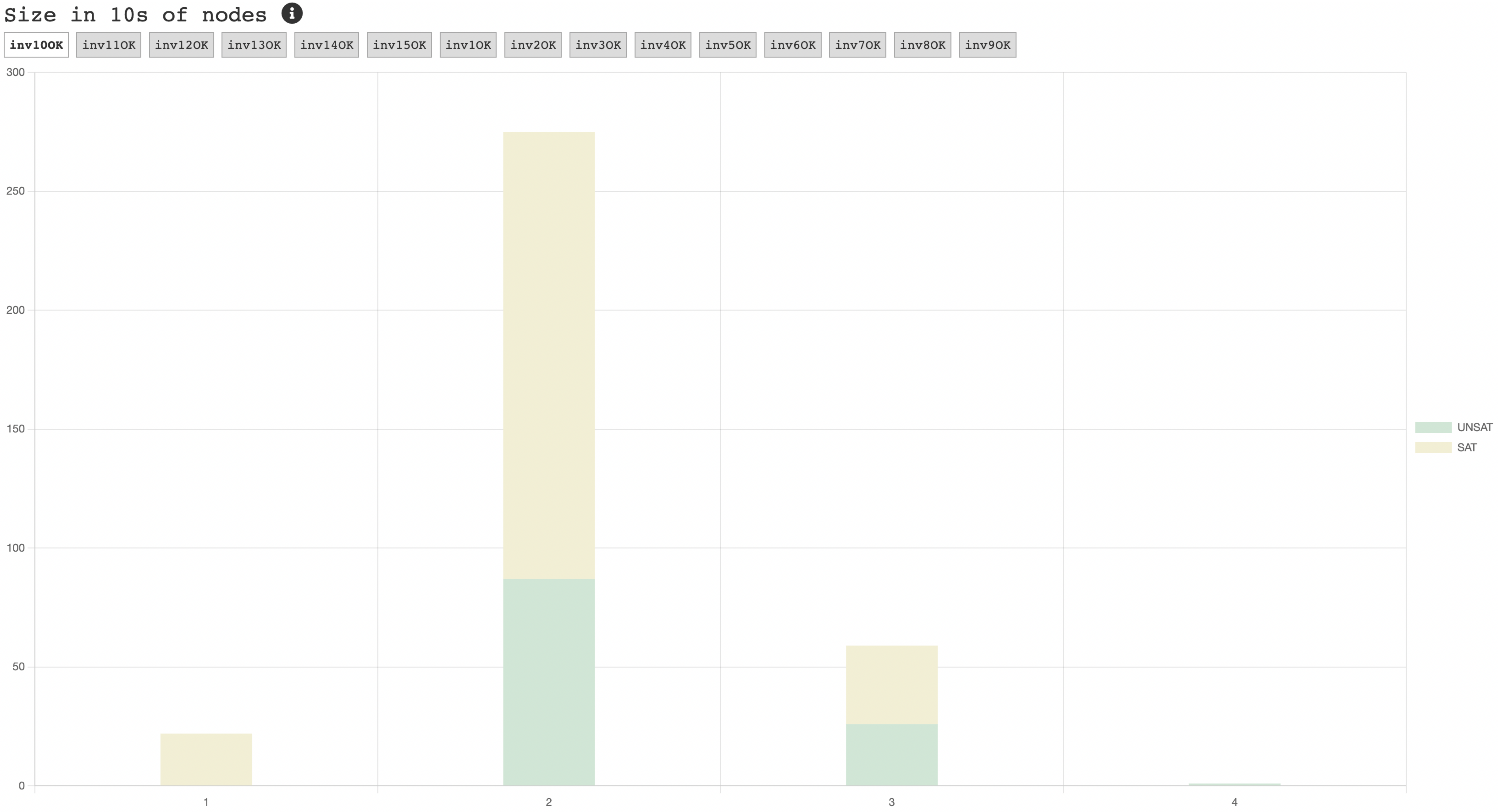
Statistics: warnings



Statistics: outcomes



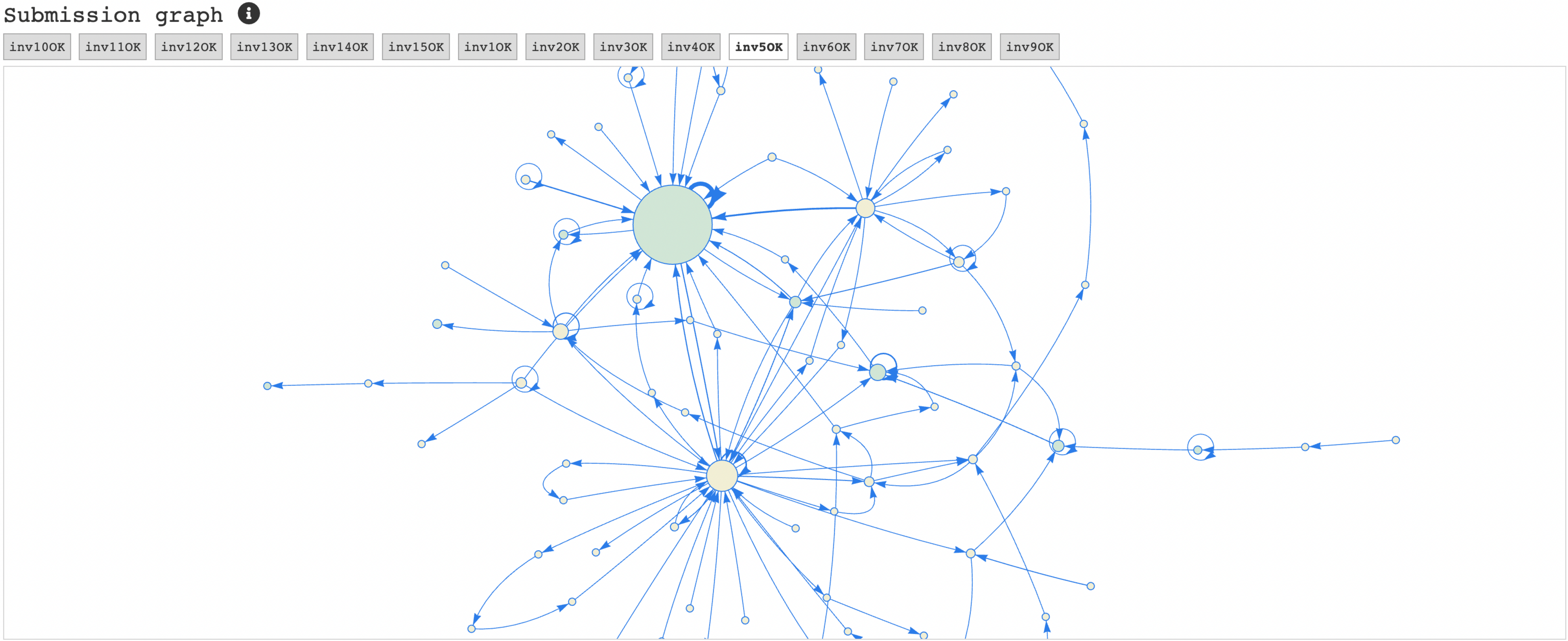
Statistics: by challenge



Statistics: submission graph

- Groups together all syntactically similar submissions and transitions
- Quick interesting insights:
  - Learning bottlenecks
  - Common reasoning steps
  - Popular correct solutions
  - …

Statistics: submission graph



Local analysis

- When accessing a private view of a model with secrets, the derivation tree for that challenge can be downloaded
- JSON file of model derivation tree from original
- Java library provided to ease some tasks (such as the ones for online statistics)

Data model

- Each model contains the following information:
    - **_id**: a unique id for the entry
    - **time**: the timestamp of its creation
    - **derivationOf**: the parent entry
    - **original**: the first ancestor with secrets (always the same within a challenge)
    - **code**: the complete code of the model (excluding the secrets defined in the original entry)
- Additionally for executed models:
    - **sat**: whether the command was satisfiable (counter-example found for checks), or -1 when error thrown
    - **cmd_i**: the index of the executed command
    - **cmd_n**: the name of the executed command for successful executions (no error)
    - **cmd_c**: whether the command was a check for successful executions (no error)
    - **msg**: the error or warning message, if any
- Additionally for shared models:
    - **theme**: the visualization theme

Metrics library

- Java library to support the analysis of Alloy4Fun datasets
  - The statistics shown previously (except graph)
- Provides derivation tree with parsed and analyzed entries
- Supports definition metric suites:
  - Methods annotated with `@MetricMethod` automatically executed
  - Parameter annotations to be executed for all desirable entries

Metrics library

- Entry points:
    - *`@ForAllSessions`*: run for all sessions
    - *`@ForAllModels`*: run for model entries
    - *`@ForAllExecutions`*: run for all execution entries
    - *`@ForAllShares`*: run for share entries
    - *`@ForAllErrors`*: run for all found errors
    - *`@ForAllSolutions`*: run for all solutions (if re-execution enabled)

Metric example

```
@MetricMethod(rule = "Errors by type", description = "The number of
errors by normalized message.")
public static Object[] errorMessages(@ForAllErrors Err err) {
  return new Object[] { MetricRunner.normUpMessages(err.msg) };
}
```

Metric example

```
@MetricMethod(rule = "Entries over time", description = "The number
of model entries by date, classified by type and result.")
public static Object[] resultsTime(@ForAllModels A4FModel entry) {
  LocalDate date = entry.time.toLocalDate();
  if (entry instanceof A4FExecution)
    return new Object[] { date, ((A4FExecution) entry).result() };
  else
    return new Object[] { date, "SHARE" };
}
```

Metric example

```
@MetricMethod(rule = "Size in 10s of nodes", description = "The number of executions, for each
challenge, by the size AST.")
public static Object[] nodeSize10(@A4FDB A4FDatabase db, @ForAllExecutions A4FExecution exe) {
  if (!db.challengeLabels().contains(exe.cmd_name))
    return null;
  AggregateVisitor<Integer> qnt =
    new AggregateVisitor<Integer>((k, l) -> k + l + 1, 1, db.challengPreds()) { };
  return new Object[] { exe.cmd_name, exe.command().formula.accept(qnt) / 10, exe.result() };
}
```

Running catalog

```
java -cp […] pt.haslab.alloy4fun.metrics.MetricHTMLPrinter \
  models.json \
  model_id \
  pt.haslab.alloy4fun.metrics.BasicCatalog
```

Alloy4Fun dataset

- We have released the Alloy4Fun dataset from our classes in Zenodo

  https://zenodo.org/record/4676413

- (Most recent 2021, latest years still pending, ~300k models)
- Can be, and has already been, used to support research
  - E.g., to evaluate model repair, Brida et al, ICSE 2021
- You can also run your own version of Alloy4Fun

  https://haslab.github.io/Alloy4Fun/