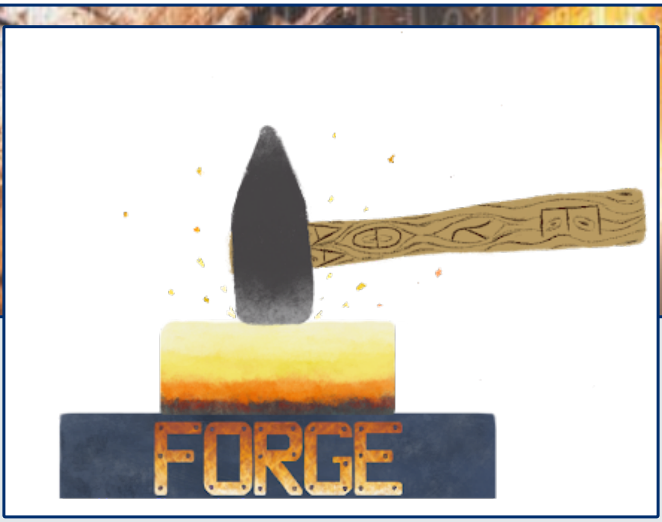
Teaching Formal Methods with Forge



Ben Greenman

Northwestern

2024-09-09

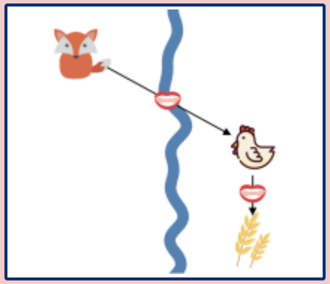
Forge = a solver-aided modeling language

Forge = a solver-aided modeling language

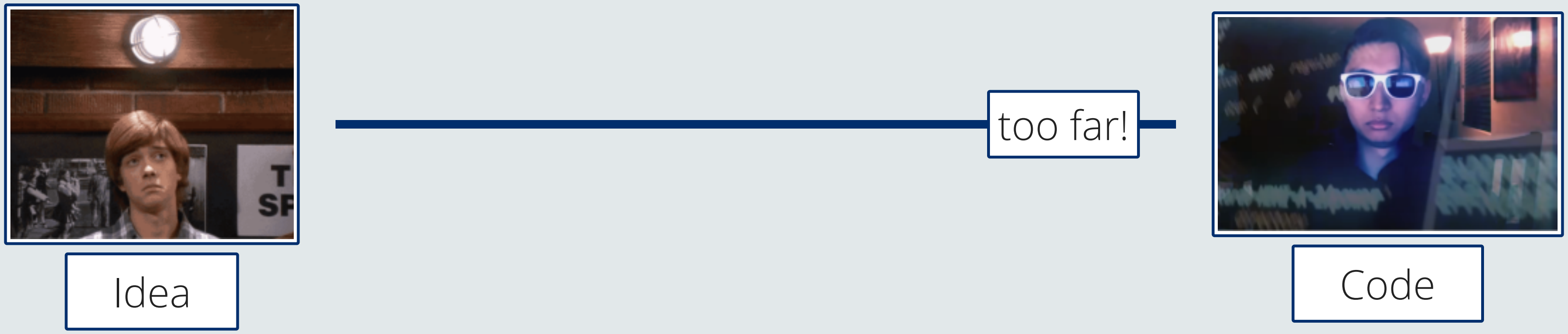
inspired by Alloy

Forge = a solver-aided modeling language

inspired by Alloy



The Problem


too far!

Idea

Code

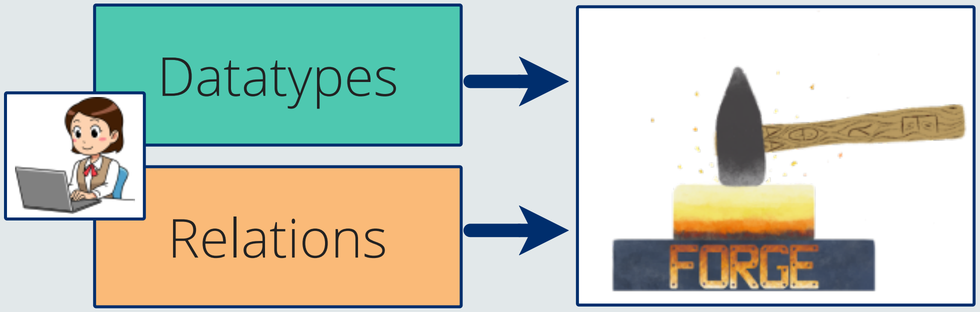
The Problem

Idea

Debug your designs

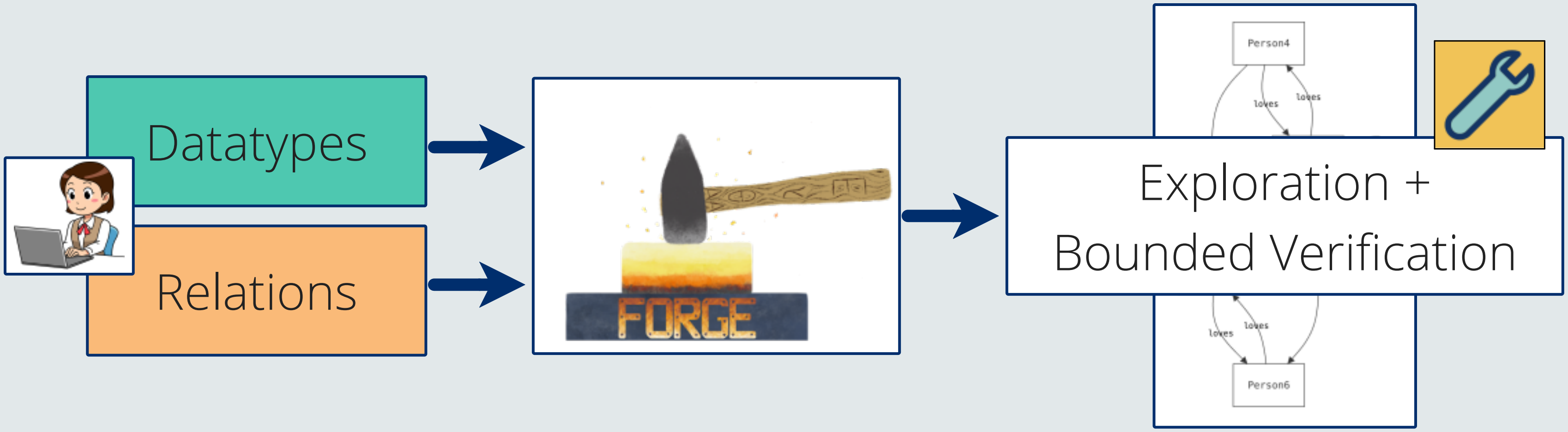
too far!

Code



Datatypes

Relations

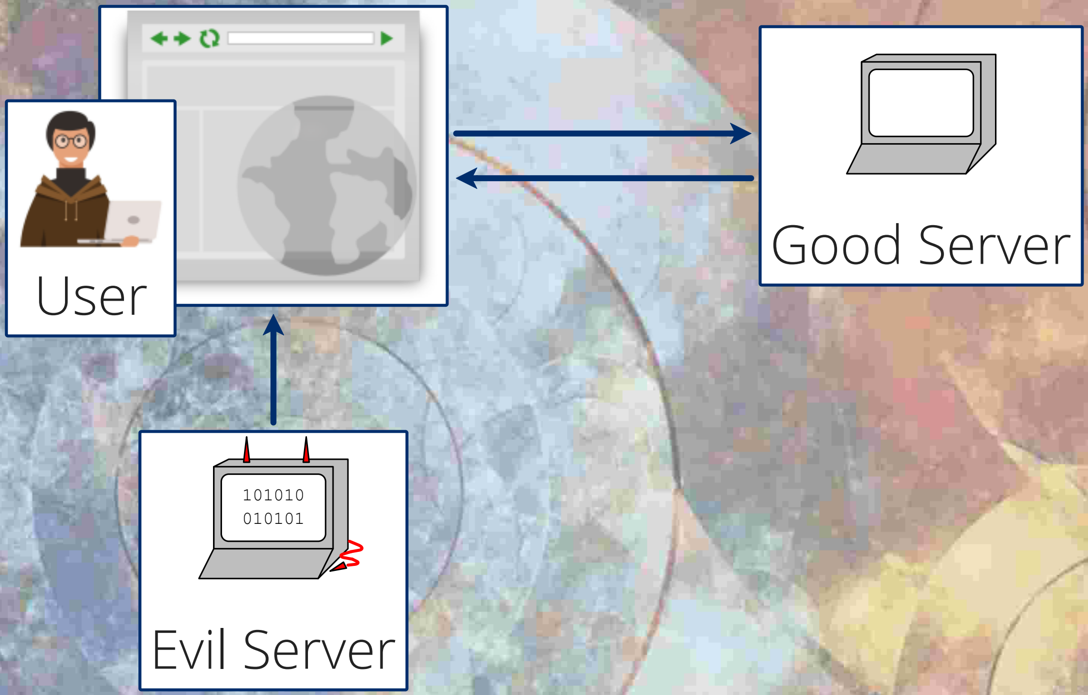


Datatypes

Relations

Exploration +
Bounded Verification

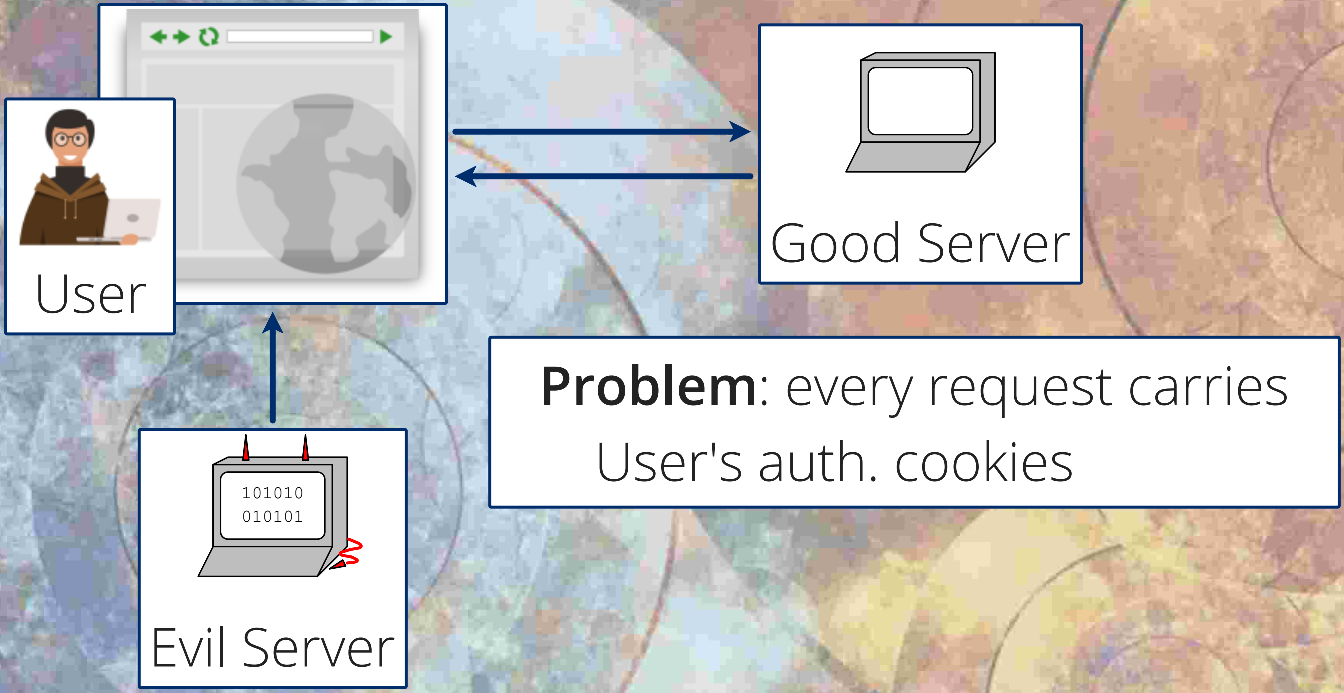
Cross-Site Request Forgery



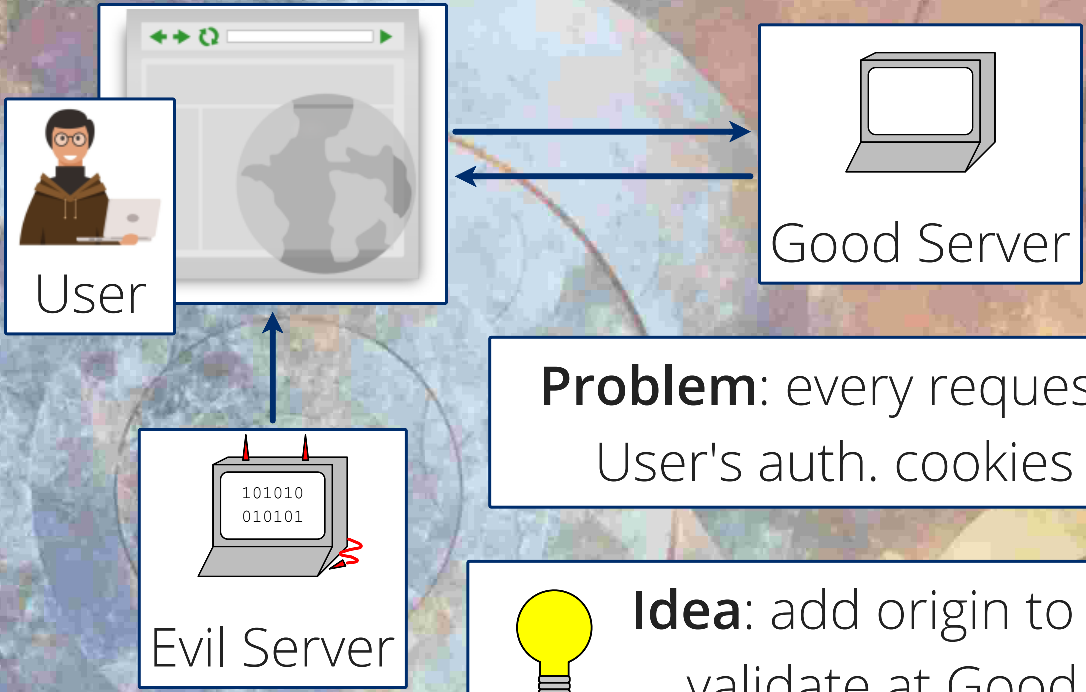
User

Good Server

Evil Server



Problem: every request carries User's auth. cookies



Problem: every request carries User's auth. cookies

Idea: add origin to requests, validate at Good Server

Datatypes

```
abstract sig EndPoint {}

sig Client
 extends EndPoint {}
```

Datatypes

```
abstract sig EndPoint {}

sig Client
 extends EndPoint {}

sig Server
 extends EndPoint {
  causes: set HTTPEvent
}
```

multiplicity

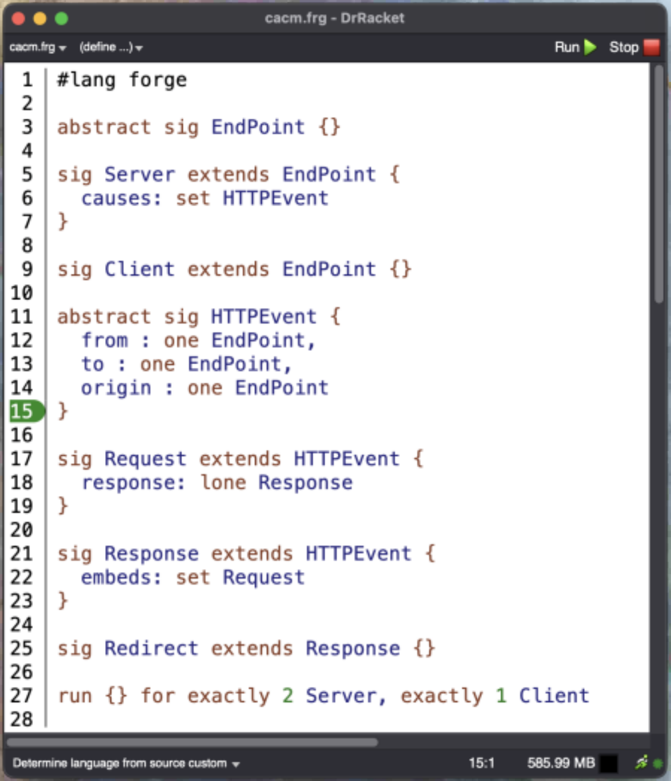
Datatypes

```
abstract sig EndPoint {}

sig Client
 extends EndPoint {}

sig Server
 extends EndPoint {
  causes: set HTTPEvent
}
```

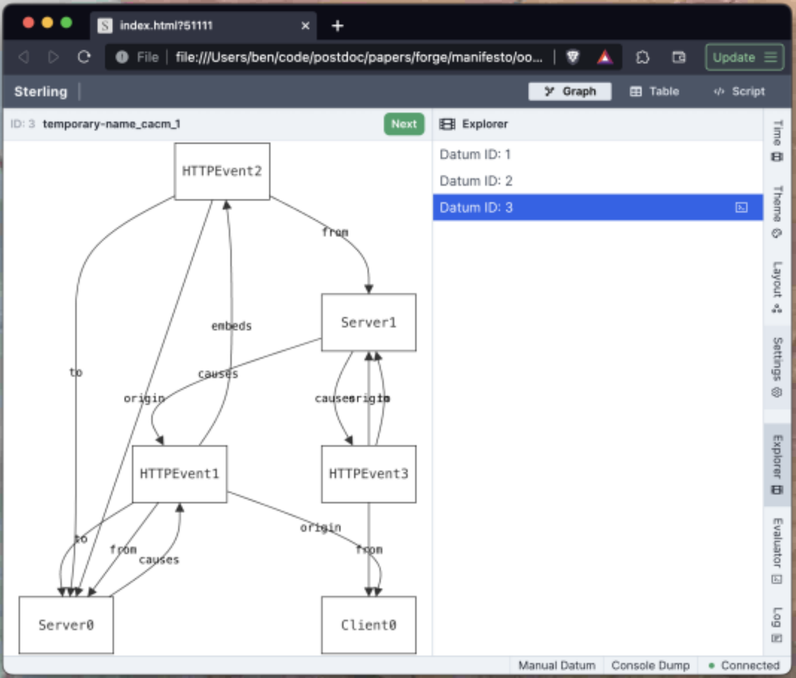
```
abstract sig HTTPEvent {
  from : one EndPoint,
  to : one EndPoint,
  origin : one EndPoint
}

// Request, Response, Redirect
//  extends HTTPEvent
```

Redirect ==> auto-retry

Bounded Exploration

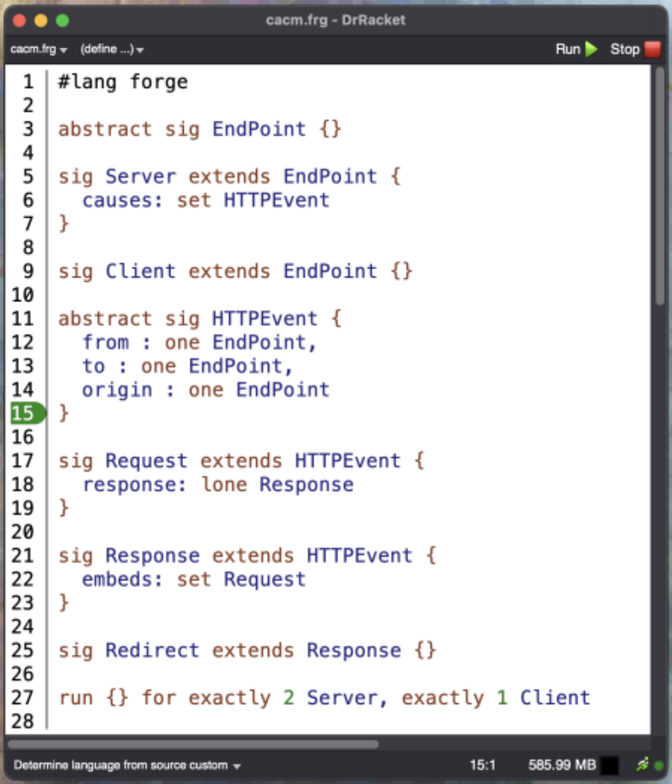


```
#lang forge

abstract sig EndPoint {}

sig Server extends EndPoint {
  causes: set HTTPEvent
}

sig Client extends EndPoint {}

abstract sig HTTPEvent {
  from : one EndPoint,
  to : one EndPoint,
  origin : one EndPoint
}

sig Request extends HTTPEvent {
  response: lone Response
}

sig Response extends HTTPEvent {
  embeds: set Request
}

sig Redirect extends Response {}

run {} for exactly 2 Server, exactly 1 Client
```

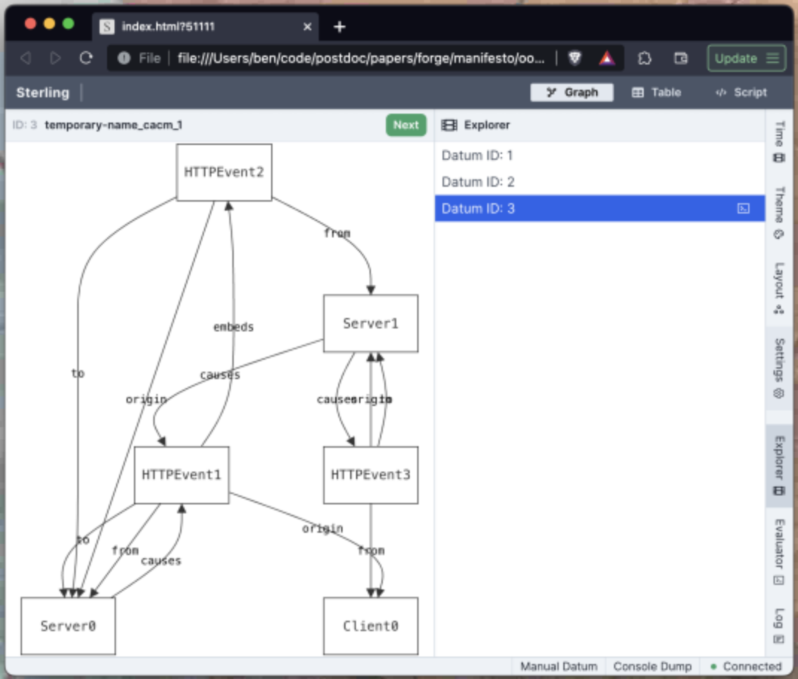


Bounded Exploration

redex-check ?



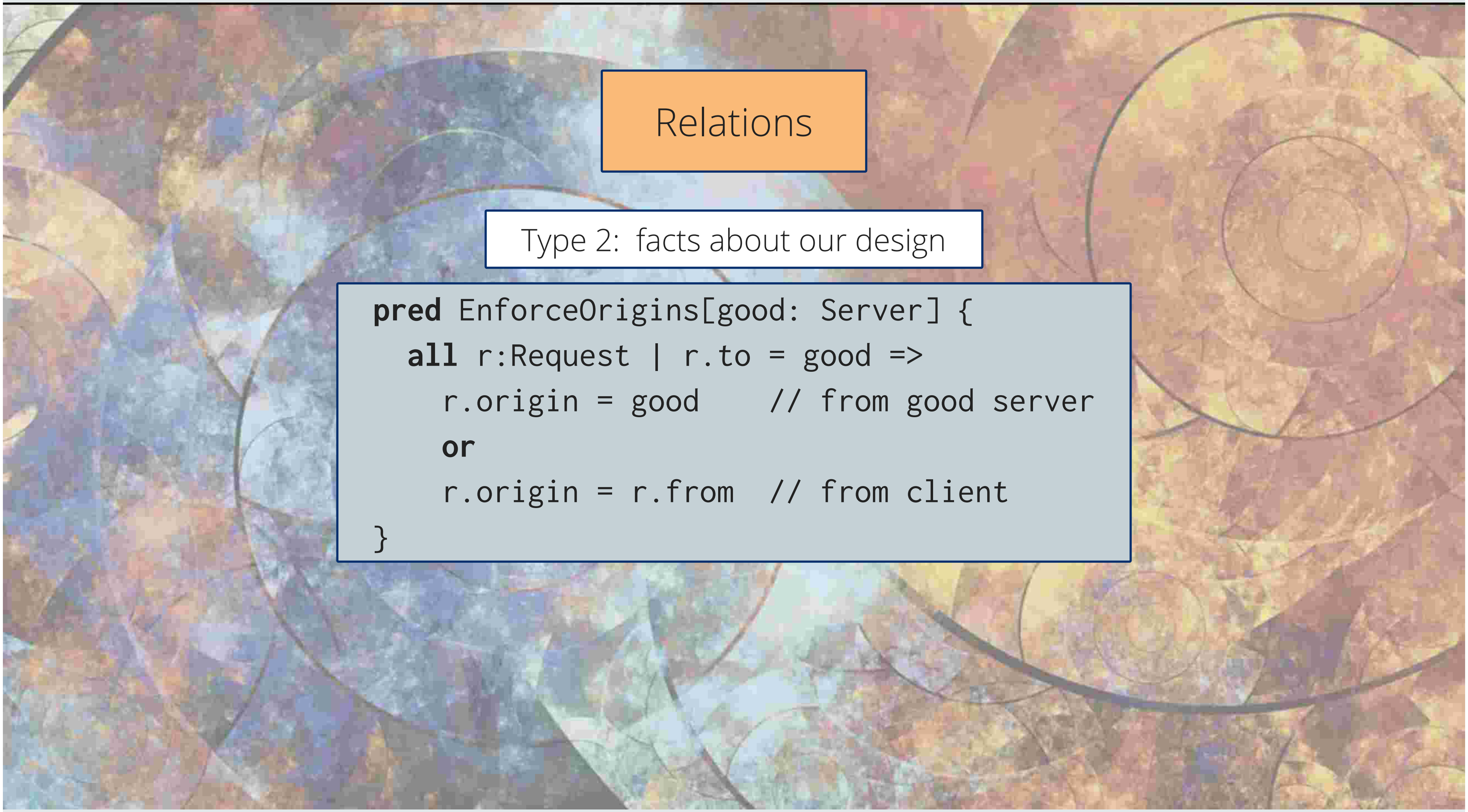
```
#lang forge

abstract sig EndPoint {}

sig Server extends EndPoint {
  causes: set HTTPEvent
}

sig Client extends EndPoint {}

abstract sig HTTPEvent {
  from : one EndPoint,
  to : one EndPoint,
  origin : one EndPoint
}

sig Request extends HTTPEvent {
  response: lone Response
}

sig Response extends HTTPEvent {
  embeds: set Request
}

sig Redirect extends Response {}

run {} for exactly 2 Server, exactly 1 Client

```



Relations

Type 1: facts about the world

```
pred RequestResponse {
  all r: Response | one response.r
  // every Response is paired with
  //  a unique request
}

// ...
```



Relations

Type 2: facts about our design

```
pred EnforceOrigins[good: Server] {
  all r:Request | r.to = good =>
    r.origin = good    // from good server
    or
    r.origin = r.from  // from client
}
```

Checks

```
run {
  // can we find (hope not)
  some good, bad: Server {
    EnforceOrigins[good]
    // ...
  }
} for exactly 2 Server,
      exactly 1 Client,
      5 HTTPEvent
```

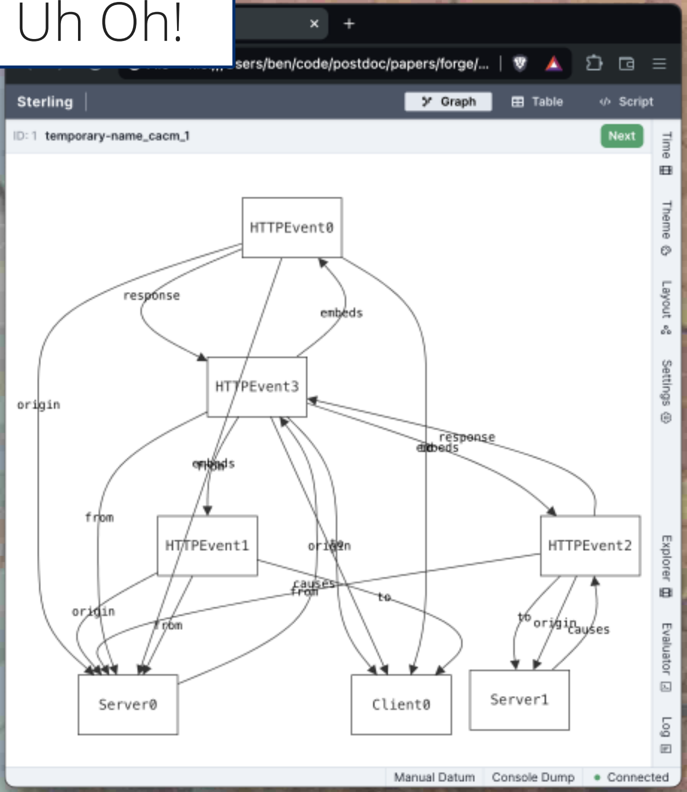
bounds

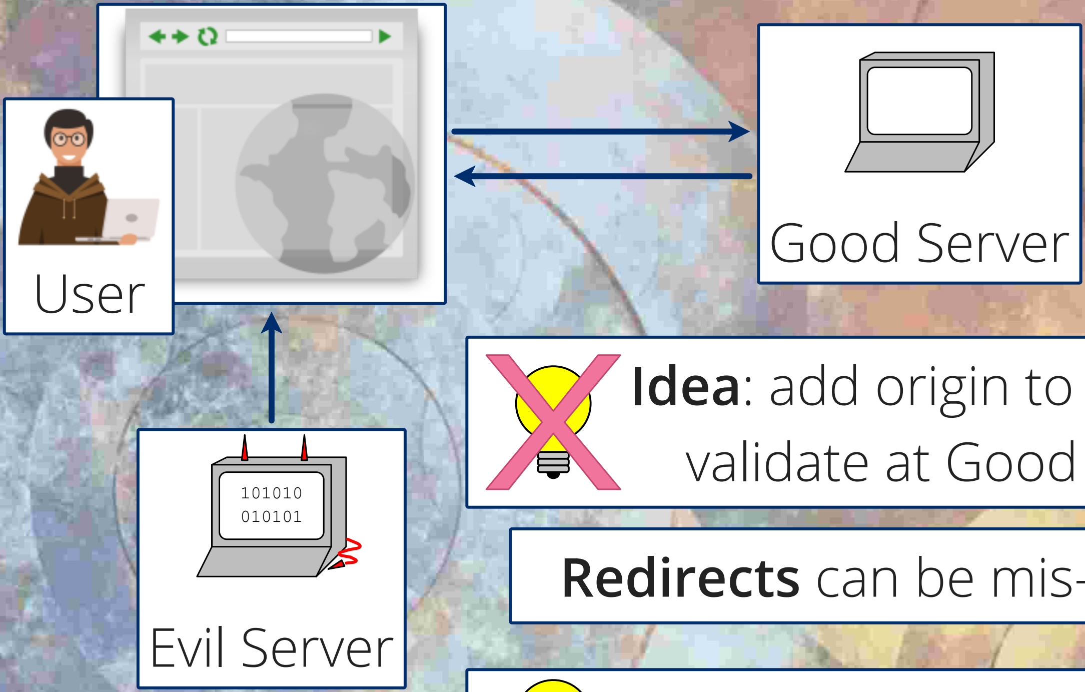
No instances?

Checks

```
run {
  // can we find (hope not)
  some good, bad: Server {
    EnforceOrigins[good]
    // ...
  }
} for exactly 2 Server,
      exactly 1 Client,
      5 HTTPEvent
```

Uh Oh!





~~**Idea**: add origin to requests, validate at Good Server~~

Redirects can be mis-labeled

How about a set of origins??

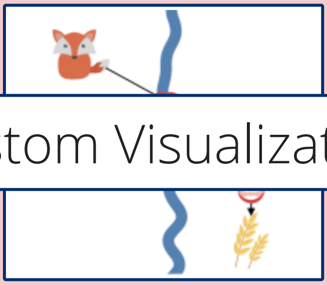
FORGE

Quickly found a bug!

🐛

What sets Forge apart?

What sets Forge apart?

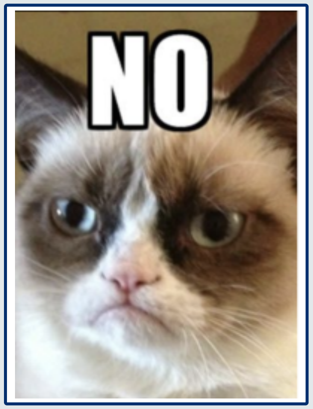


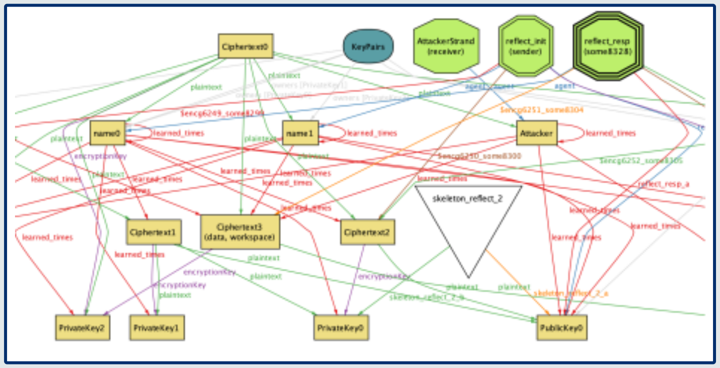
Custom Visualization

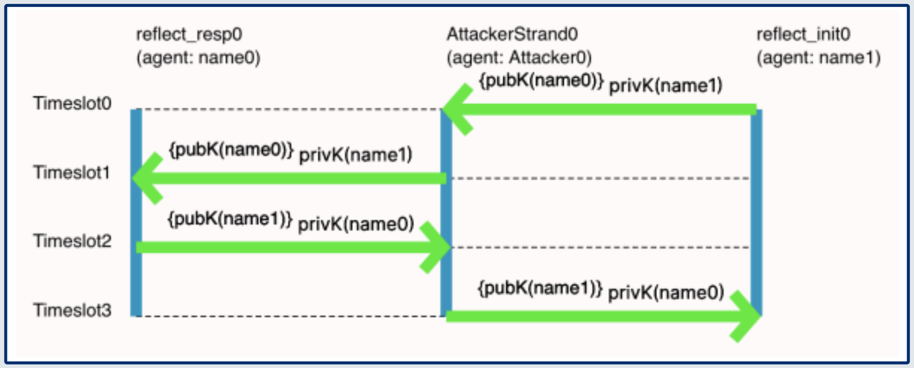
Unit Testing

Language Levels

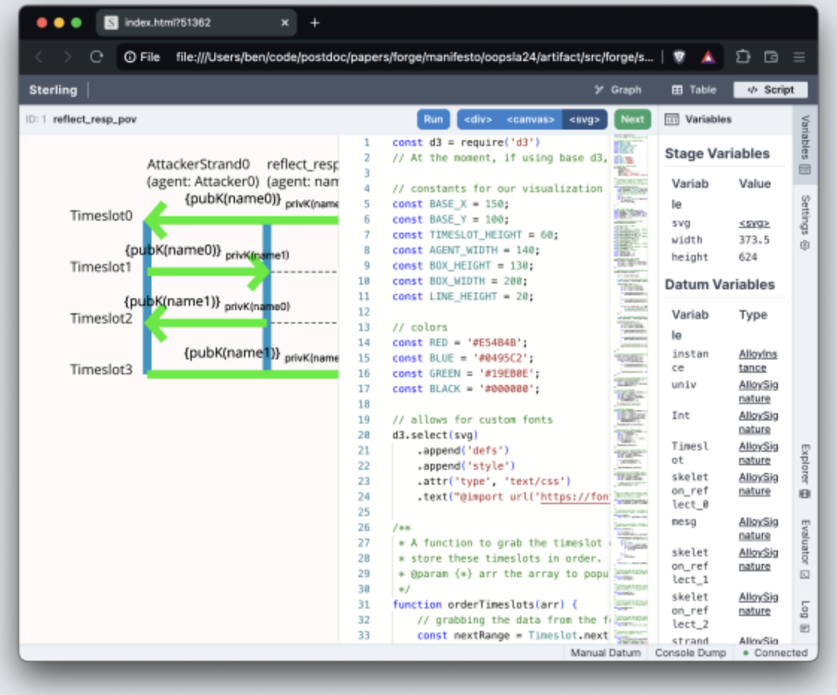
Custom Visualization







Custom Visualization



```
const d3 = require('d3')
// At the moment, if using base d3,

// constants for our visualization
const BASE_X = 150;
const BASE_Y = 100;
const TIMESLOT_HEIGHT = 60;
const AGENT_WIDTH = 140;
const BOX_HEIGHT = 130;
const BOX_WIDTH = 200;
const LINE_HEIGHT = 20;

// colors
const RED = '#E54B4B';
const BLUE = '#0495C2';
const GREEN = '#19EB0E';
const BLACK = '#000000';

// allows for custom fonts
d3.select(svg)
    .append('defs')
    .append('style')
    .attr('type', 'text/css')
    .text("@import url('https://fon

/**
 * A function to grab the timeslot
 * store these timeslots in order.
 * @param {*} arr the array to popu
 */
function orderTimeslots(arr) {
    // grabbing the data from the f
    const nextRange = Timeslot.next
```

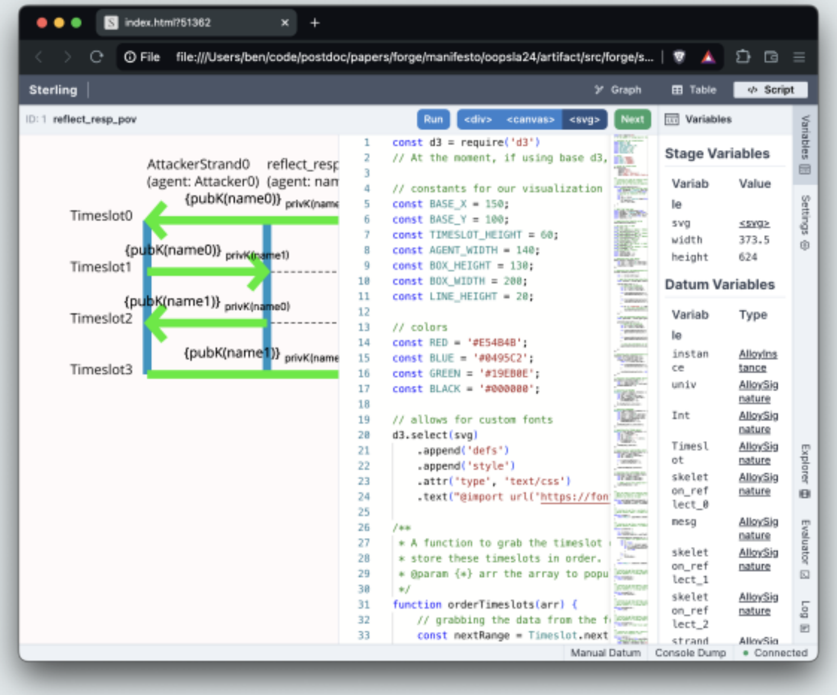
Stage Variables

Variable	Value
svg	<svg>
width	373.5
height	624

Datum Variables

Variable	Type
instance	AlloyInstance
univ	AlloySignature
Int	AlloySignature
Timeslot	AlloySignature
skeleton_reflect_0	AlloySignature
mesg	AlloySignature
skeleton_reflect_1	AlloySignature
skeleton_reflect_2	AlloySignature
strand	AlloySig

Custom Visualization



```
const d3 = require('d3')
// At the moment, if using base d3,

// constants for our visualization
const BASE_X = 150;
const BASE_Y = 100;
const TIMESLOT_HEIGHT = 60;
const AGENT_WIDTH = 140;
const BOX_HEIGHT = 130;
const BOX_WIDTH = 200;
const LINE_HEIGHT = 20;

// colors
const RED = '#E54B4B';
const BLUE = '#0495C2';
const GREEN = '#19EB0E';
const BLACK = '#000000';

// allows for custom fonts
d3.select(svg)
    .append('defs')
    .append('style')
    .attr('type', 'text/css')
    .text("@import url('https://fon

/**
 * A function to grab the timeslot
 * store these timeslots in order.
 * @param {*} arr the array to popu
 */
function orderTimeslots(arr) {
    // grabbing the data from the f
    const nextRange = Timeslot.next
```

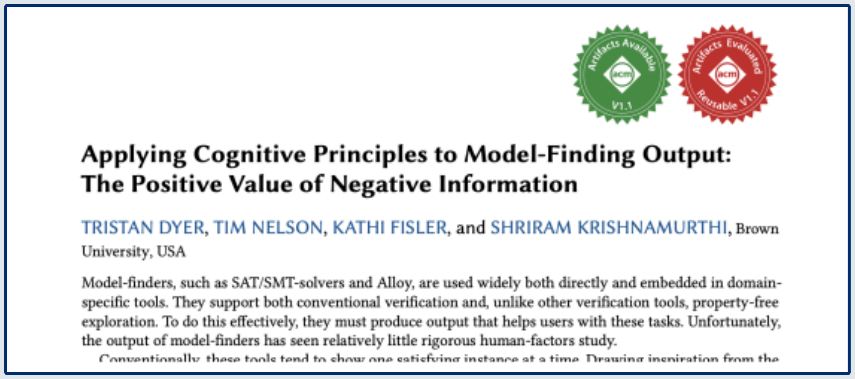
Stage Variables

Variable	Value
svg	<svg>
width	373.5
height	624

Datum Variables

Variable	Type
instance	AlloyInstance
univ	AlloySignature
Int	AlloySignature
Timeslot	AlloySignature
skeleton_reflect_0	AlloySignature
mesg	AlloySignature
skeleton_reflect_1	AlloySignature
skeleton_reflect_2	AlloySignature
strand	AlloySig

Much more than pretty pictures!

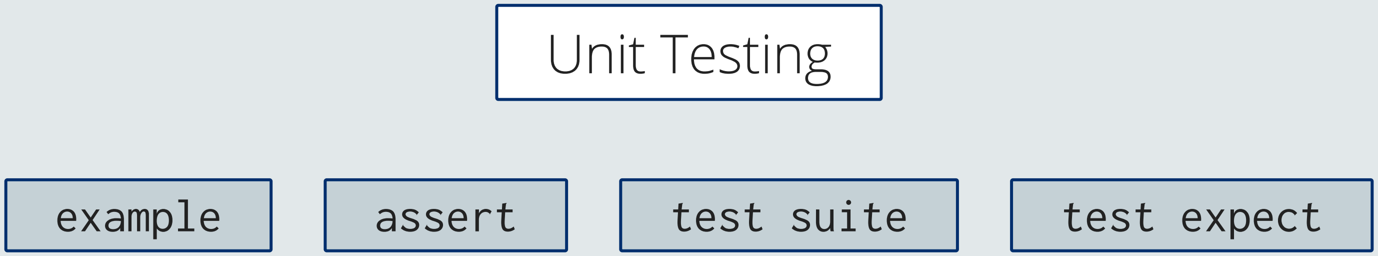


Applying Cognitive Principles to Model-Finding Output: The Positive Value of Negative Information

TRISTAN DYER, TIM NELSON, KATHI FISLER, and SHRIRAM KRISHNAMURTHI, Brown University, USA

Model-finders, such as SAT/SMT-solvers and Alloy, are used widely both directly and embedded in domain-specific tools. They support both conventional verification and, unlike other verification tools, property-free exploration. To do this effectively, they must produce output that helps users with these tasks. Unfortunately, the output of model-finders has seen relatively little rigorous human-factors study.

Conventionally, these tools tend to show one satisfying instance at a time. Drawing inspiration from the



Unit Testing

`example`

`assert`

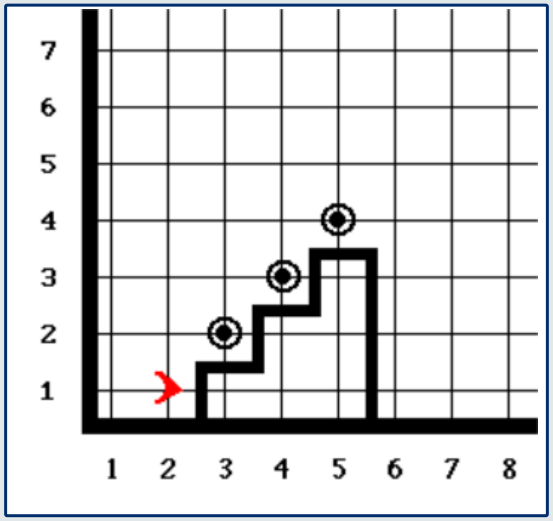
`test suite`

`test expect`

Unit Testing

`example` `assert` `test suite` `test expect`

But: Programming != Modeling



Unit Testing

```
pred row1_Xfull {(Board.board[0]).X = (0+1+2)}
pred some_moved {some Board.board}

inst good_ttt {
    Board = `Board0    X = `X    O = `O    Player = `X + `O
    `Board0.board = (1, 1) -> `X + (1, 2) -> `O }
```

partial instance

Unit Testing

```
pred row1_Xfull {(Board.board[0]).X = (0+1+2)}
pred some_moved {some Board.board}

inst good_ttt {
    Board = `Board0    X = `X    O = `O    Player = `X + `O
    `Board0.board = (1, 1) -> `X + (1, 2) -> `O }
```

Unit Testing

```
pred row1_Xfull {(Board.board[0]).X = (0+1+2)}
pred some_moved {some Board.board}

inst good_ttt {
    Board = `Board0    X = `X    O = `O    Player = `X + `O
    `Board0.board = (1, 1) -> `X + (1, 2) -> `O }
```

```
example moveMiddleFirst is {wellformed} for good_ttt
```

pred vs inst

Unit Testing

```
pred row1_Xfull {(Board.board[0]).X = (0+1+2)}
pred some_moved {some Board.board}

inst good_ttt {
    Board = `Board0    X = `X    O = `O    Player = `X + `O
    `Board0.board = (1, 1) -> `X + (1, 2) -> `O }
```

```
example moveMiddleFirst is {wellformed} for good_ttt

test suite for winning {
  assert row1_Xfull is sufficient for winning for 1 Board
  assert some_moved is necessary for winning for 1 Board }
```

pred vs pred: over- / under-constraint

Test recipe?

1. instances
2. overconstraints
3. underconstraints

Test recipe?

1. instances
2. overconstraints
3. underconstraints

Other basic checks:
- vacuity
- determinism ...

Language Levels

Language Levels

```
r not in r.^(response.embeds)
```

Language Levels

```
r not in r.^(response.embeds)
```

```
CS1 in prereqs.CS2
```

"What a travesty that would be!"

Language Levels

```
#lang forge/temporal
```

++ Linear Temporal Logic

```
#lang forge/relational
```

++ N-ary Relations

```
#lang forge/bsl
```

Functional Relations

Language Levels

```
#lang forge/froglet

abstract sig Player {}
one sig X, O extends Player {}
sig Board { board : pfunc ( Int -> Int) -> Player }

pred wellformed {
  all b: Board | all row, col : Int | {
    -- no out-of-bounds marks
    (row < 0 or row > 2 or col < 0 or col > 2) ==>
      no b.board[row][col] } }
```

simple functions

no relational operators

Language Levels

```
#lang forge/relational

sig Node { edges : set Node -> Int }

pred connected {
  all disj n1, n2: Node | n2 in n1.^(edges.Int) }
```

set

^ = transitive closure

Language Levels

```
#lang forge/temporal

option max_tracelength 12

sig Counter { var value : one Int }

pred incrs {
  Counter.value = 0
  always {
    Counter.value' = add[Counter.value, 1] }}
```

var

'

Core Language

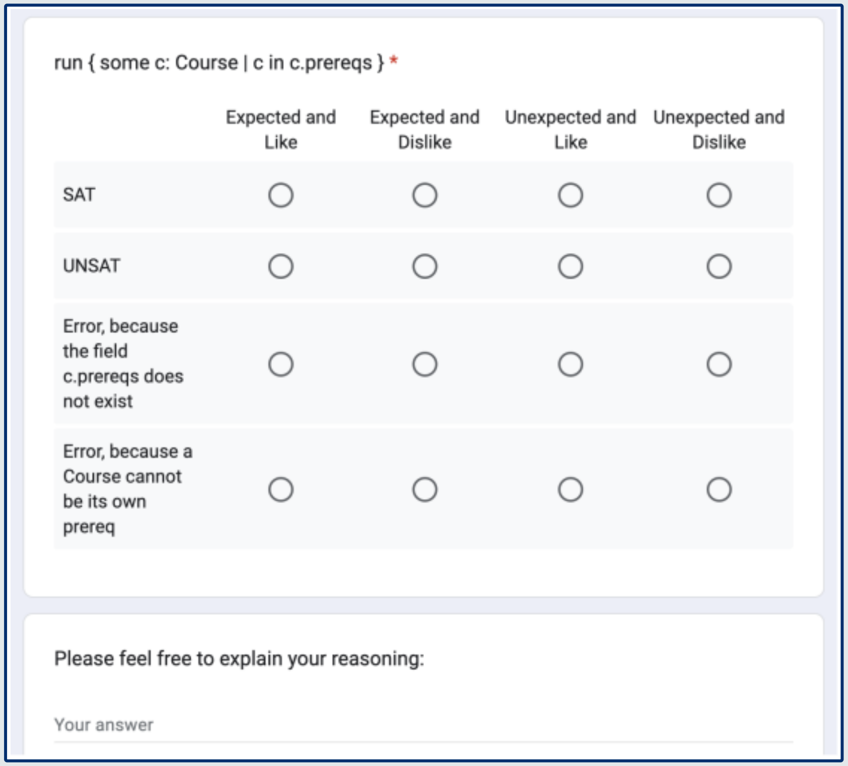
```
#lang forge/temporal
```

```
e, 1] }}
```

```
#lang forge/core

(set-option! 'problem_type 'temporal)
(set-option! 'max_tracelength 12)

(sig Counter)
(relation value (Counter Int) #:is-var "var")
(pred incrs
  (and (= (join Counter value) (int 0))
       (always (= (join Counter (prime value))
                  (add (join Counter value) (int 1))))))

(run incrs_run #:preds [incrs])
(display incrs_run)
```

Evaluation

Pre-switch Surveys



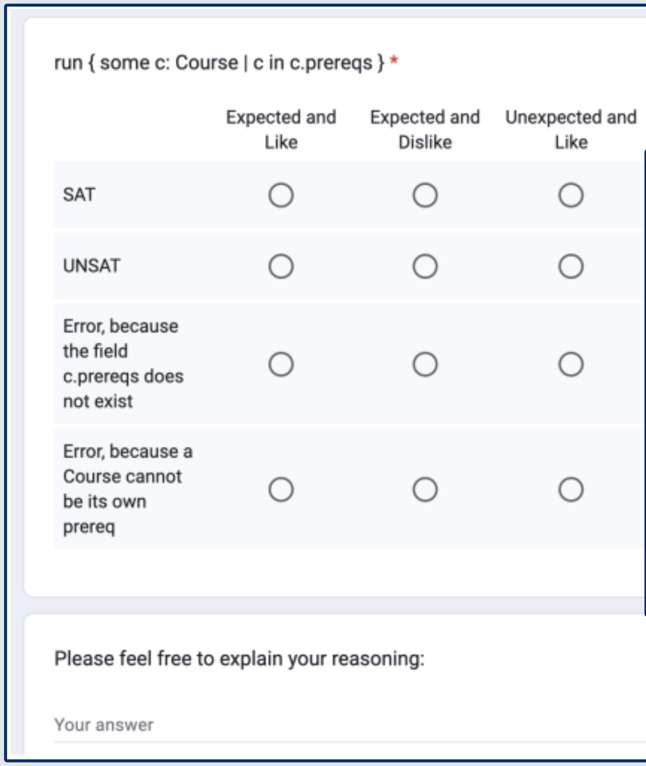
run { some c: Course | c in c.prereqs } *

	Expected and Like	Expected and Dislike	Unexpected and Like	Unexpected and Dislike
SAT	○	○	○	○
UNSAT	○	○	○	○
Error, because the field c.prereqs does not exist	○	○	○	○
Error, because a Course cannot be its own prereq	○	○	○	○

Please feel free to explain your reasoning:

Your answer

Pre-switch Surveys

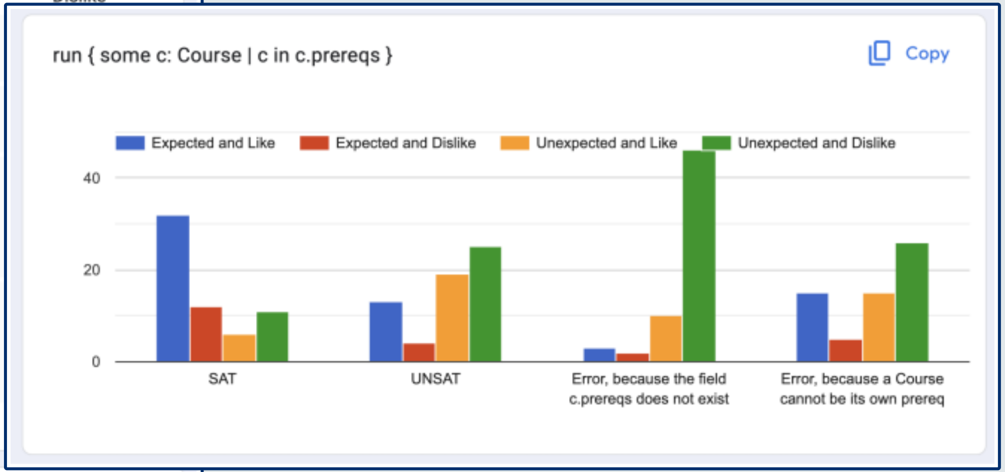


run { some c: Course | c in c.prereqs } *

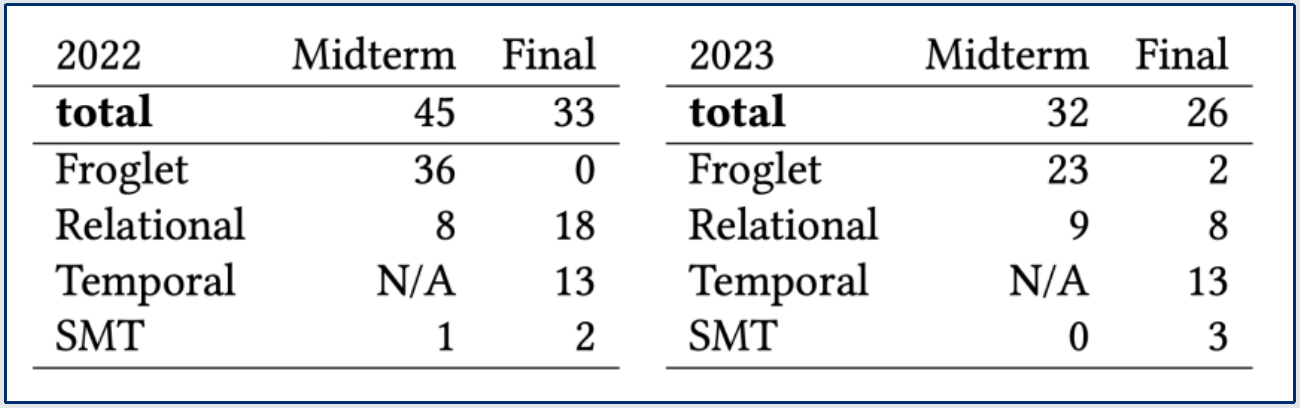
	Expected and Like	Expected and Dislike	Unexpected and Like	Unexpected and Dislike
SAT	○	○	○	
UNSAT	○	○	○	
Error, because the field c.prereqs does not exist	○	○	○	
Error, because a Course cannot be its own prereq	○	○	○	

Please feel free to explain your reasoning:

Your answer

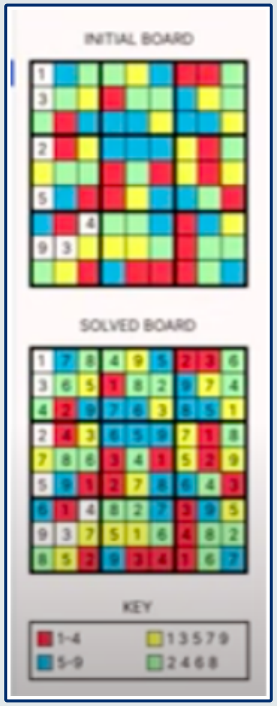


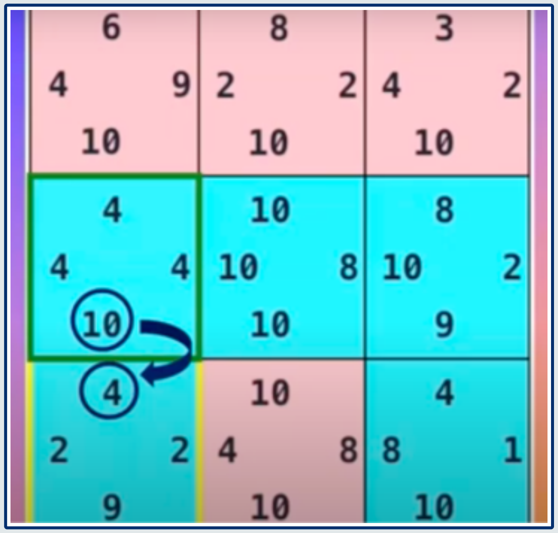
run { some c: Course | c in c.prereqs }

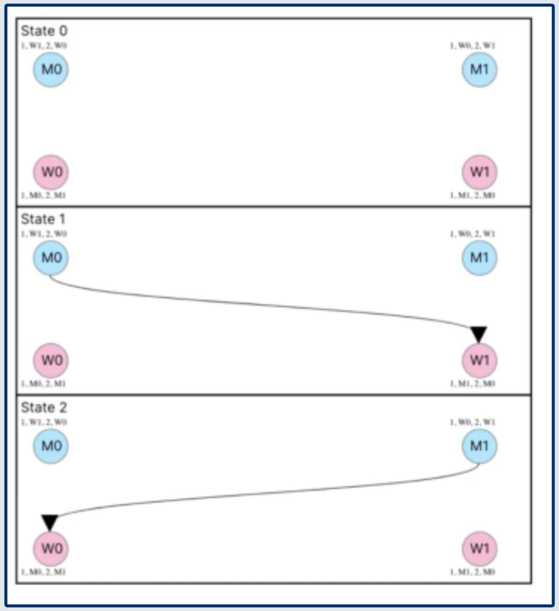


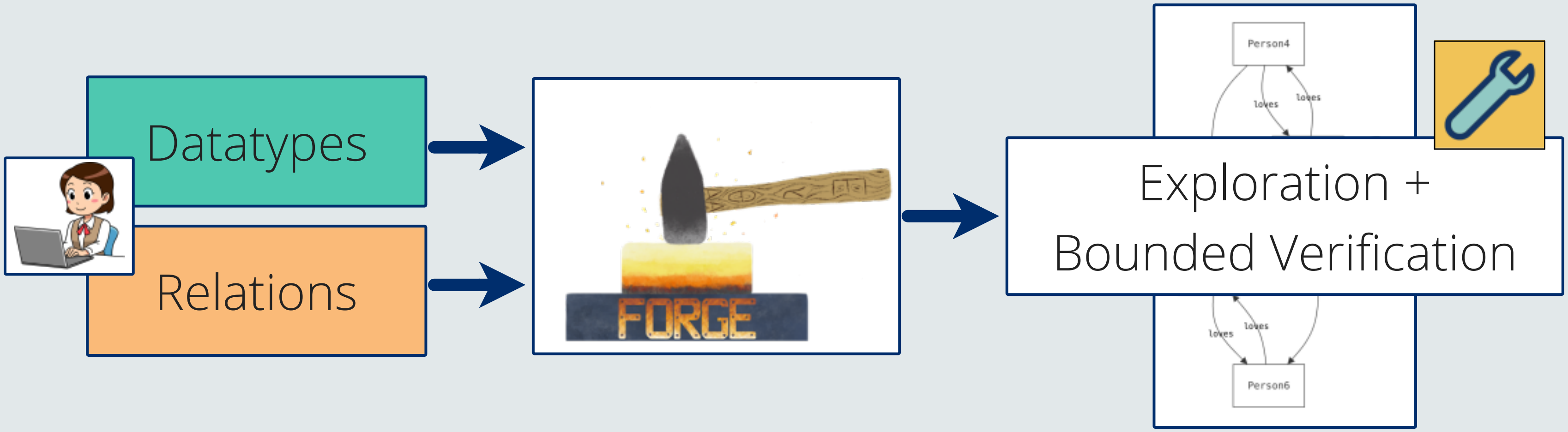
2022	Midterm	Final
total	45	33
Froglet	36	0
Relational	8	18
Temporal	N/A	13
SMT	1	2

2023	Midterm	Final
total	32	26
Froglet	23	2
Relational	9	8
Temporal	N/A	13
SMT	0	3


INITIAL BOARD
SOLVED BOARD
KEY
1-4
5-9
13579
2468


6
4 9
10
8
2 2
10
3
4 2
10
4
4 4
10
10
10 8
10
8
10 2
9
4
2 2
9
10
4 8
10
4
8 1
10

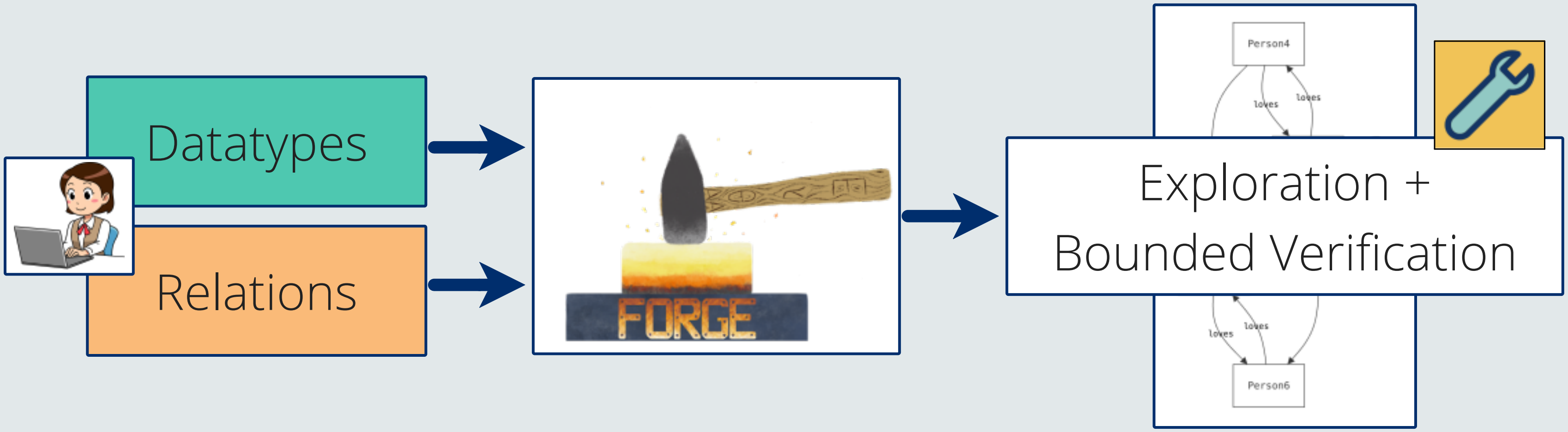

State 0
1, W1, 2, W0
M0
1, W0, 2, W1
M1
W0
1, M0, 2, M1
W1
1, M1, 2, M0
State 1
1, W1, 2, W0
M0
1, W0, 2, W1
M1
W0
1, M0, 2, M1
W1
1, M1, 2, M0
State 2
1, W1, 2, W0
M0
1, W0, 2, W1
M1
W0
1, M0, 2, M1
W1
1, M1, 2, M0



Datatypes

Relations

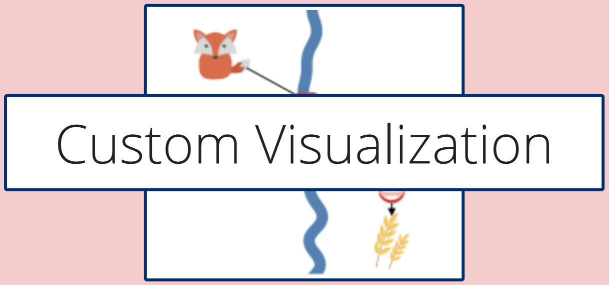
Exploration +
Bounded Verification



Datatypes

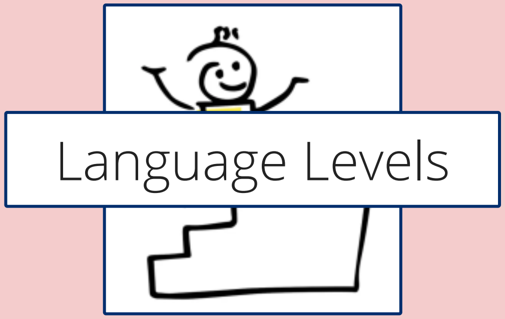
Relations

Exploration +
Bounded Verification



Custom Visualization

Unit Testing

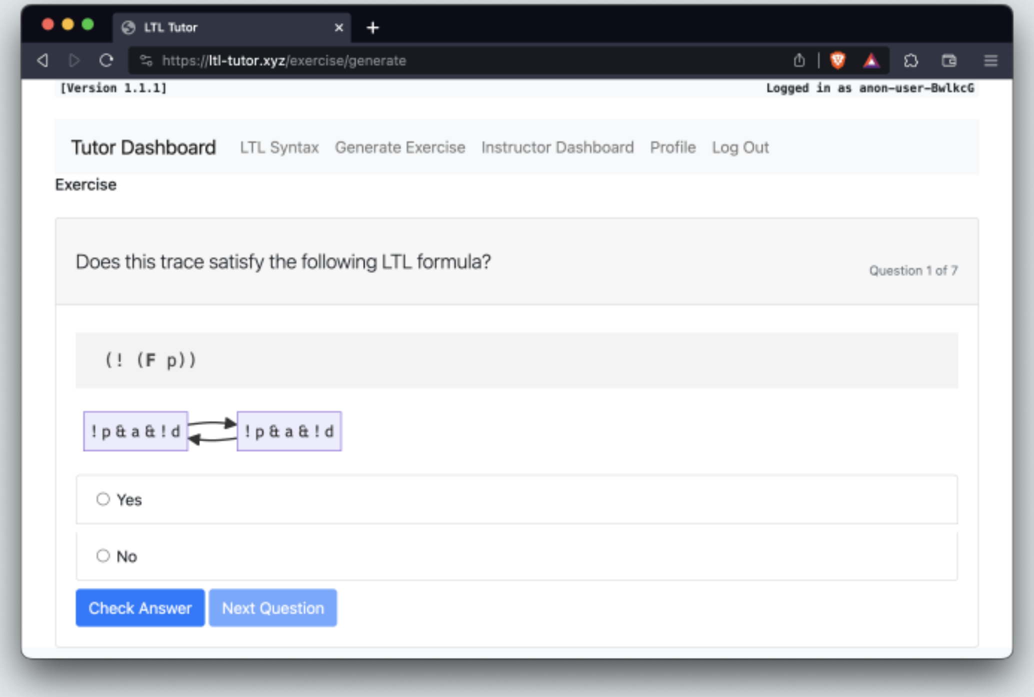


Language Levels

https://forge-fm.org

blg@cs.utah.edu

https://ltl-tutor.xyz



Exercise

Does this trace satisfy the following LTL formula?

Question 1 of 7

```
(! (F p))
```

!p & a & !d ⇄ !p & a & !d

○ Yes

○ No

Siddhartha