

Teaching Formal Methods with Forge

Ben Greenman

Northwestern

2024-09-09



Forge = a solver-aided
modeling language





Forge = a solver-aided
modeling language



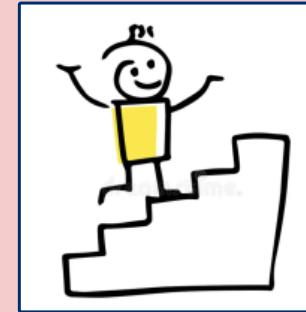
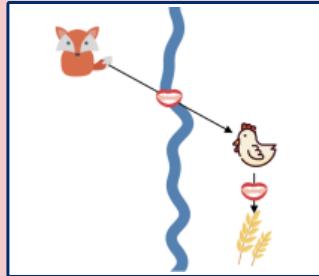
inspired by Alloy



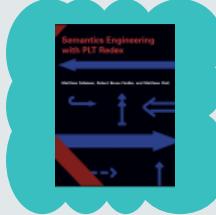
Forge = a solver-aided
modeling language



inspired by Alloy



The Problem



Idea

too far!



Code

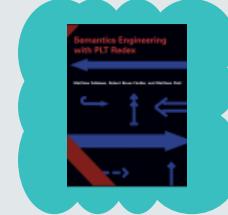
The Problem



Idea



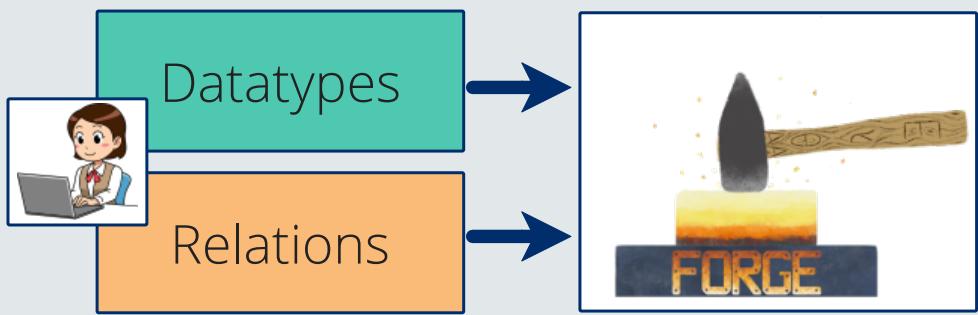
Debug your designs

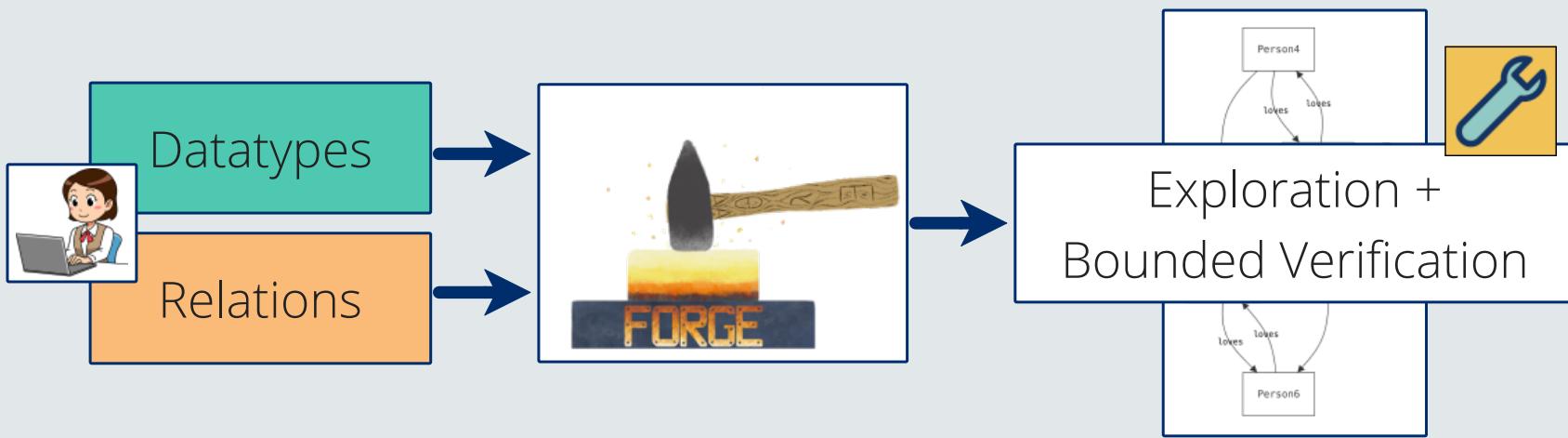


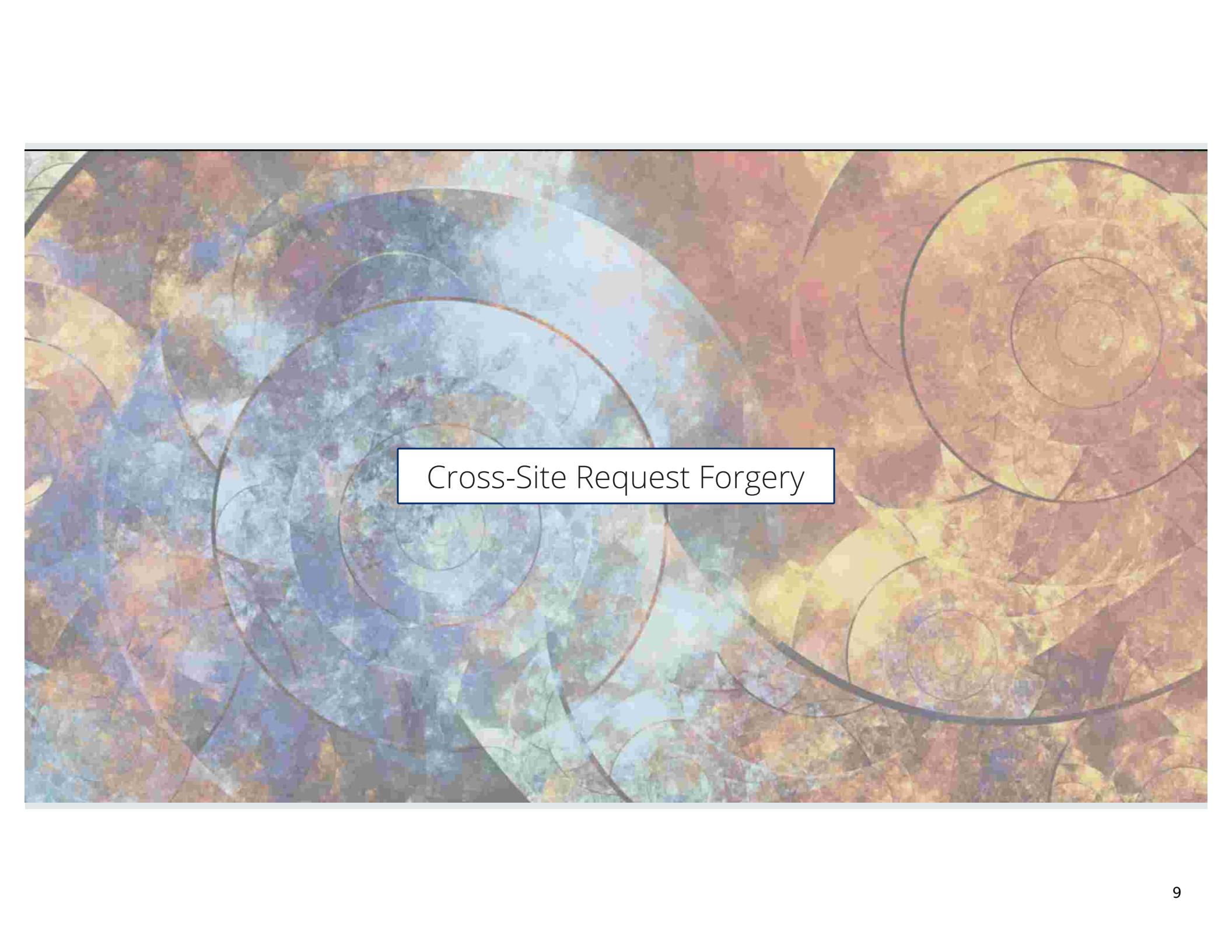
too far!



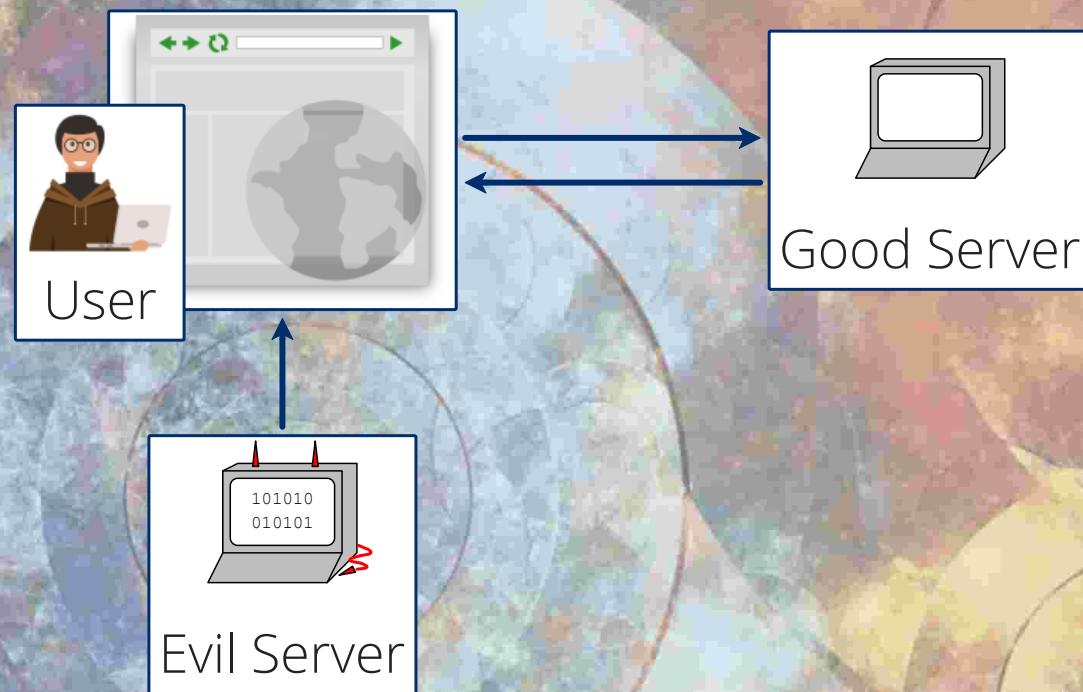
Code

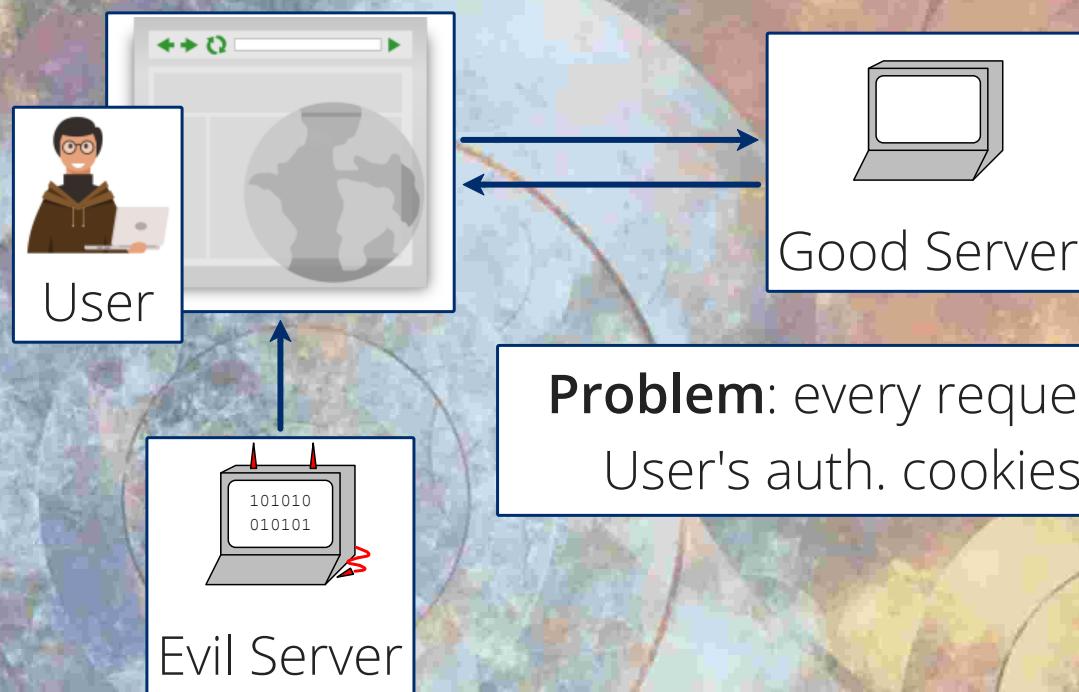




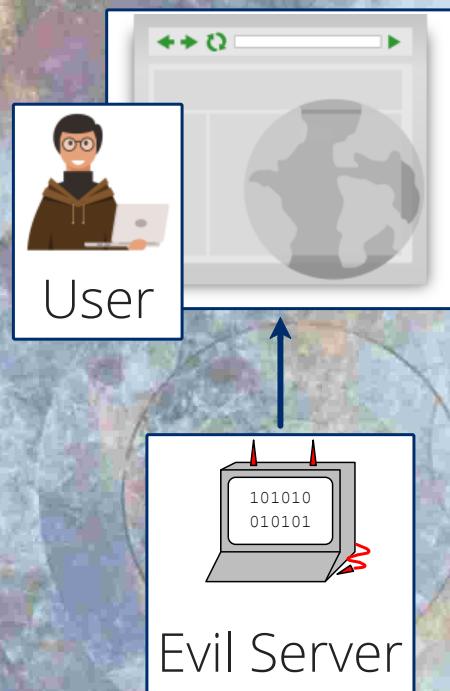


Cross-Site Request Forgery





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

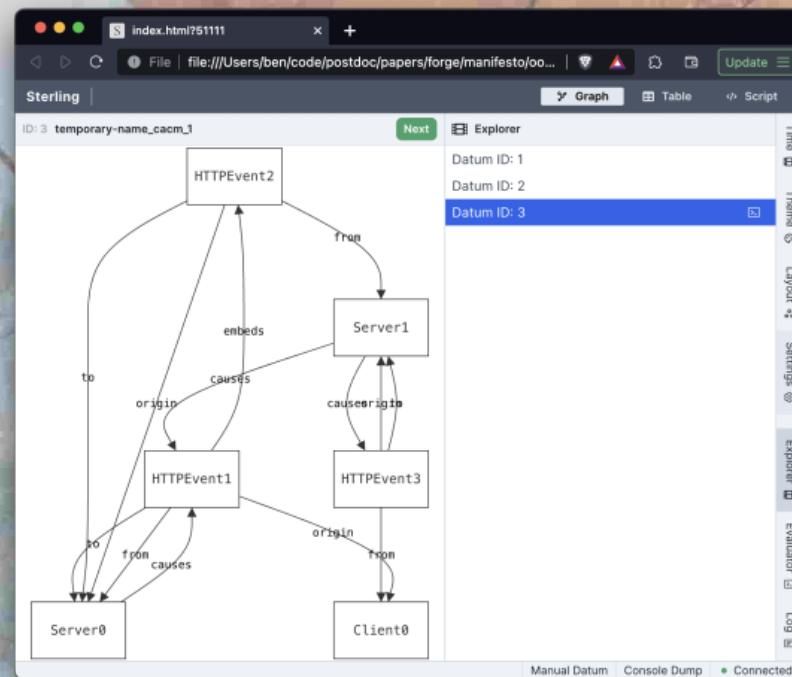
```
abstract sig EndPoint {}  
  
sig Client  
  extends EndPoint {}  
  
sig Server  
  extends EndPoint {  
    causes: set HTTPEvent  
  }
```

```
abstract sig HTTPEvent {  
  from : one EndPoint,  
  to : one EndPoint,  
  origin : one EndPoint  
}  
  
// Request, Response, Redirect  
// extends HTTPEvent
```

Redirect ==> auto-retry

Bounded Exploration

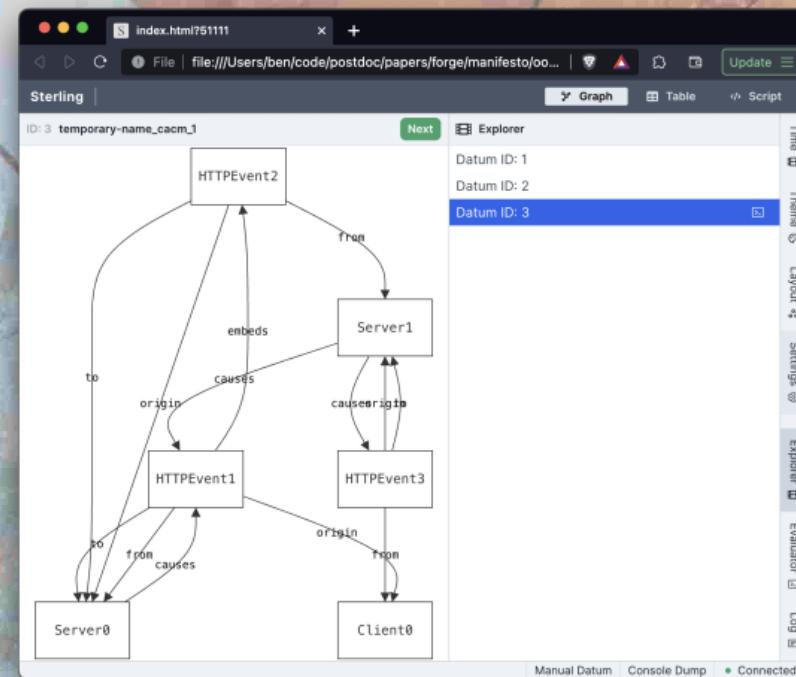
```
cacm.frg - DrRacket  
Run ▶ Stop □  
1 #lang forge  
2  
3 abstract sig EndPoint {}  
4  
5 sig Server extends EndPoint {  
6   causes: set HTTPEvent  
7 }  
8  
9 sig Client extends EndPoint {}  
10  
11 abstract sig HTTPEvent {  
12   from : one EndPoint,  
13   to : one EndPoint,  
14   origin : one EndPoint  
15 }  
16  
17 sig Request extends HTTPEvent {  
18   response: lone Response  
19 }  
20  
21 sig Response extends HTTPEvent {  
22   embeds: set Request  
23 }  
24  
25 sig Redirect extends Response {}  
26  
27 run {} for exactly 2 Server, exactly 1 Client  
28
```



Bounded Exploration

rex - check ?

```
cacm.frg - DrRacket  
Run Stop  
1 #lang forge  
2  
3 abstract sig EndPoint {}  
4  
5 sig Server extends EndPoint {  
    causes: set HTTPEvent  
}  
6  
7 sig Client extends EndPoint {}  
8  
9 abstract sig HTTPEvent {  
    from : one EndPoint,  
    to : one EndPoint,  
    origin : one EndPoint  
10}  
11  
12 sig Request extends HTTPEvent {  
    response: lone Response  
}  
13  
14 sig Response extends HTTPEvent {  
    embeds: set Request  
}  
15  
16 sig Redirect extends Response {}  
17  
18 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
```

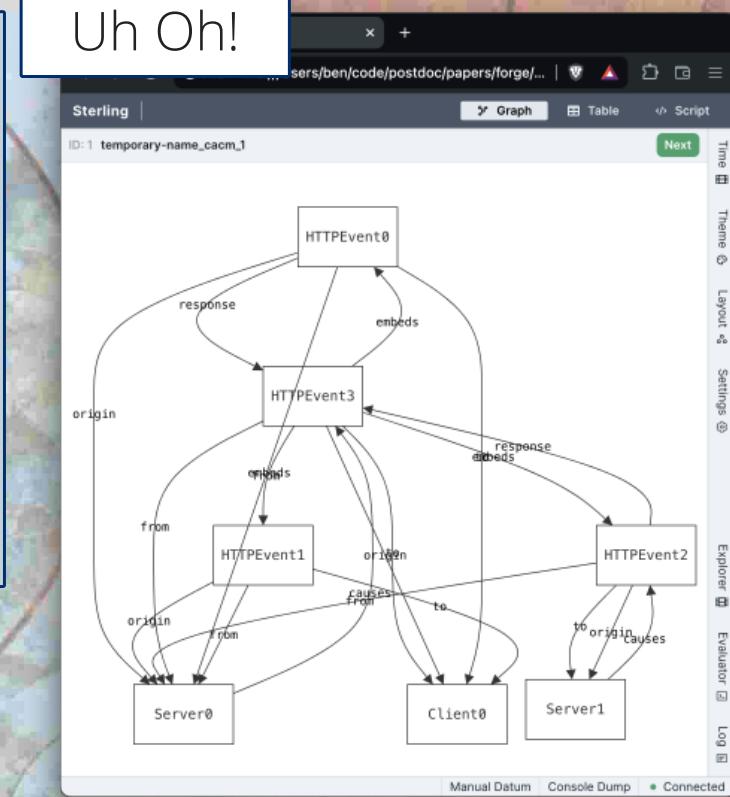
bounds

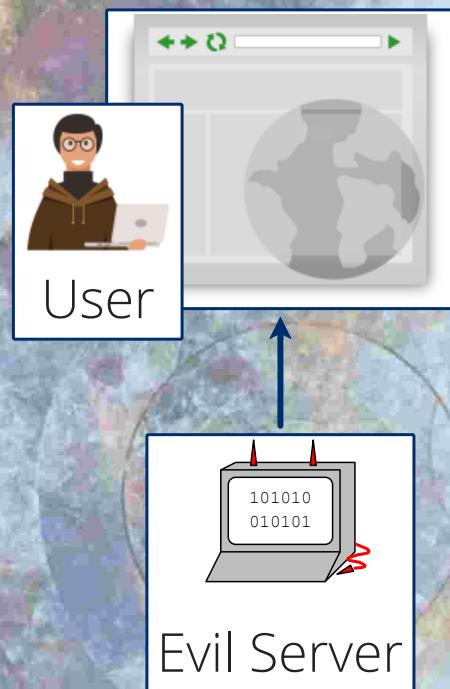
No instances?

Checks

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

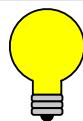
Uh Oh!





Idea: add origin to requests,
validate at Good Server

Redirects can be mis-labeled



How about a set of origins??



Quickly found a bug!



What sets Forge apart?



What sets Forge apart?



Custom Visualization



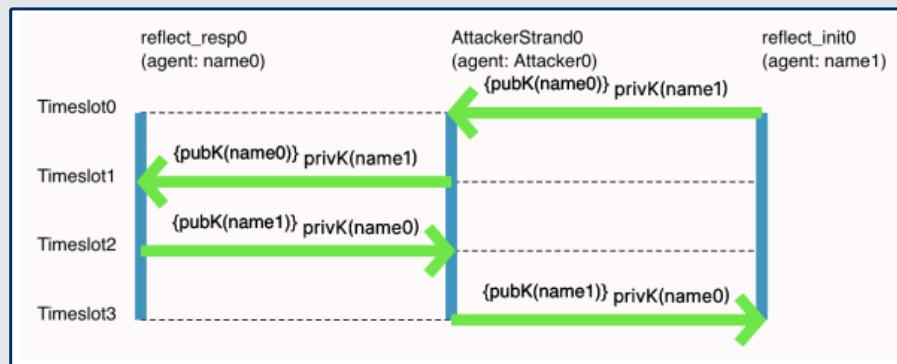
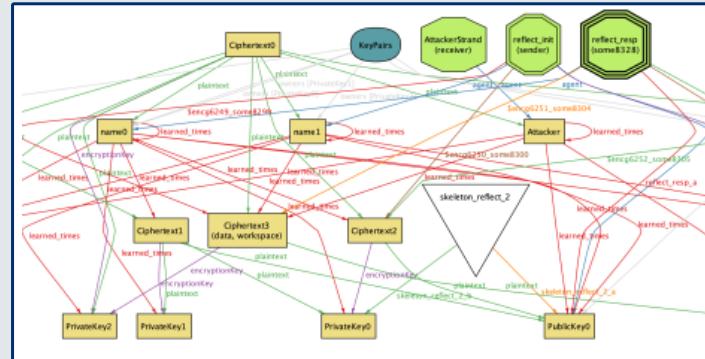
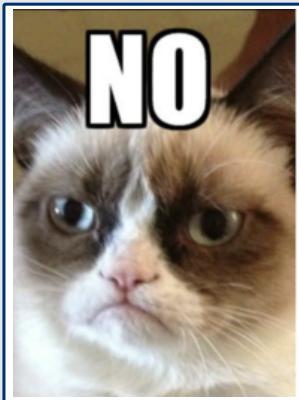
Unit Testing



Language Levels



Custom Visualization



Custom Visualization

The screenshot shows the Sterling IDE interface with a code editor and a visualization pane.

Code Editor:

```
1 const d3 = require('d3')
2 // At the moment, if using base d3,
3 // constants for our visualization
4 const BASE_X = 150;
5 const BASE_Y = 100;
6 const TIMESLOT_HEIGHT = 60;
7 const AGENT_WIDTH = 140;
8 const BOX_HEIGHT = 130;
9 const BOX_WIDTH = 200;
10 const LINE_HEIGHT = 20;
11
12 // colors
13 const RED = '#E54B4B';
14 const BLUE = '#0495C2';
15 const GREEN = '#19E0EE';
16 const BLACK = '#000000';
17
18 // allows for custom fonts
19 d3.select(svg)
20   .append('defs')
21   .append('style')
22   .attr('type', 'text/css')
23   .text(`@import url('${fontPath}')`)
24
25 /**
26 * A function to grab the timeslot
27 * store these timeslots in order.
28 * @param {[]} arr the array to popu
29 */
30
31 function orderTimeslots(arr) {
32   // grabbing the data from the f
33   const nextRange = Timeslot.next
```

Visualization:

The visualization shows four horizontal bars representing timeslots. Each bar has two segments: a blue segment at the top and a green segment at the bottom. The bars are labeled from top to bottom: Timeslot0, Timeslot1, Timeslot2, and Timeslot3. Each bar also contains text indicating a public key (`{pubK(name0)}`) and a private key (`privK(name)`). Arrows point from the text in the visualization to the corresponding code in the editor.

Variables:

Stage Variables	Variable	Type
le	instanc	AlloySig
svg	instanc	AlloySig
width	Int	AlloySig
height	Int	AlloySig

Datum Variables	Variable	Type
le	instanc	AlloySig
ot	instanc	AlloySig
skelet	instanc	AlloySig
on_reflect_0	instanc	AlloySig
msg	instanc	AlloySig
skelot	instanc	AlloySig
on_reflect_1	instanc	AlloySig
skelet	instanc	AlloySig
on_reflect_2	instanc	AlloySig
strand	instanc	AlloySig

Custom Visualization

```

ID:1 reflect_resp.pov
AttackerStrand0 reflect_resp
(agent: Attacker0) (agent: nan
{pubK(name0)} privK(name
Timeslot0 {pubK(name0)} privK(name1)
Timeslot1 {pubK(name1)} privK(name0)
Timeslot2 {pubK(name1)} privK(name0)
Timeslot3 {pubK(name1)} privK(name

```

```

1 const d3 = require('d3')
2 // At the moment, if using base d3,
3
4 // constants for our visualization
5 const BASE_X = 150;
6 const BASE_Y = 100;
7 const TIMESLOT_HEIGHT = 60;
8 const AGENT_WIDTH = 140;
9 const BOX_HEIGHT = 130;
10 const BOX_WIDTH = 200;
11 const LINE_HEIGHT = 20;
12
13 // colors
14 const RED = '#E5484B';
15 const BLUE = '#0495C2';
16 const GREEN = '#19E0BD';
17 const BLACK = '#000000';
18
19 // allows for custom fonts
20 d3.select(svg)
21   .append('defs')
22   .append('style')
23   .attr('type', 'text/css')
24   .text(`@import url('${fontPath}')`)
25
26 /**
27 * A function to grab the timeslot
28 * store these timeslots in order.
29 * @param {*} arr the array to popu
30 */
31 function orderTimeslots(arr) {
32   // grabbing the data from the f
33   const nextRange = Timeslot.next

```

Run <div> <canvas> <svg> Next

Variables

Stage Variables	Varib	Value
le	svg	SVGS
	width	373.5
	height	624

Datum Variables

Datum Variables	Varib	Type
instan	le	AlloyInst
univ	ce	AlloySig
	univ	nature
Int	Int	AlloySig
	Timesl	AlloySig
ot	ot	nature
skelet	skelet	AlloySig
on_ref	on_ref	nature
lect_0	lect_0	AlloySig
msg	msg	nature
	skelet	AlloySig
on_ref	on_ref	nature
lect_1	lect_1	AlloySig
	skelet	AlloySig
on_ref	on_ref	nature
lect_2	lect_2	AlloySig
strand	strand	AlloySig

Manual Datum Console Dump Connected

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

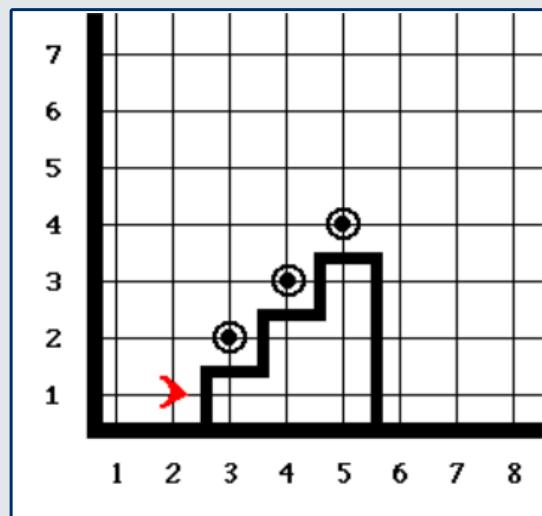
example

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 { partial instance
    Board = `Board0      X = `X      0 = `0      Player = `X + `0
    `Board0.board = (1, 1) -> `X + (1, 2) -> `0 }
```

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      0 = `0      Player = `X + `0
    `Board0.board = (1, 1) -> `X + (1, 2) -> `0 }
```

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      0 = `0      Player = `X + `0
    `Board0.board = (1, 1) -> `X + (1, 2) -> `0 }
```

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

```
pred vs inst
```

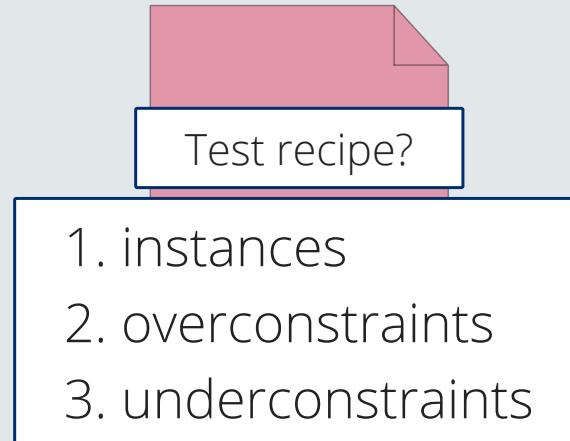
Unit Testing

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

inst good_ttt {
    Board = `Board0      X = `X      0 = `0      Player = `X + `0
    `Board0.board = (1, 1) -> `X + (1, 2) -> `0 }
```

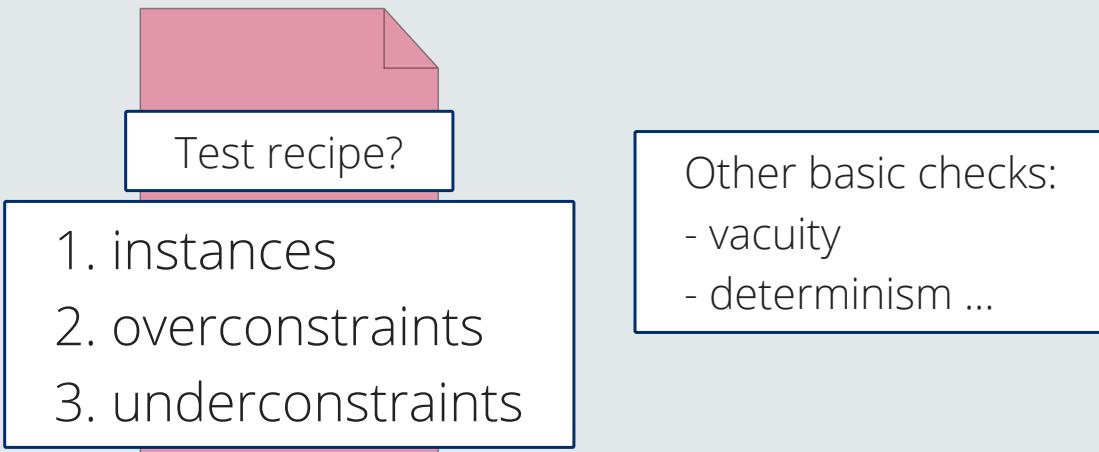
```
example moveMiddleFirst is {wellformed} for good_ttt
test suite for winning {
    assert row1_Xfull is sufficient for winning for 1 Board
    assert some_moved is necessary for winning for 1 Board }
```

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, 0 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 }
```

set

\wedge = transitive closure

```
pred connected {
```

```
all disj n1, n2: Node | n2 in n1. $\wedge$ (edges.Int) }
```

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
```

```
#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)
```

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

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	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
UNSAT	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Error, because the field c.prereqs does not exist	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Error, because a Course cannot be its own prereq	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

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	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	
UNSAT	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	
Error, because the field c.prereqs does not exist	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	
Error, because a Course cannot be its own prereq	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	

Please feel free to explain your reasoning:

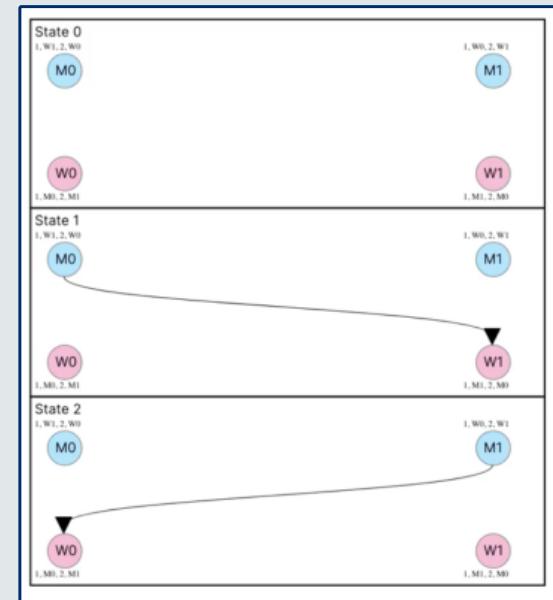
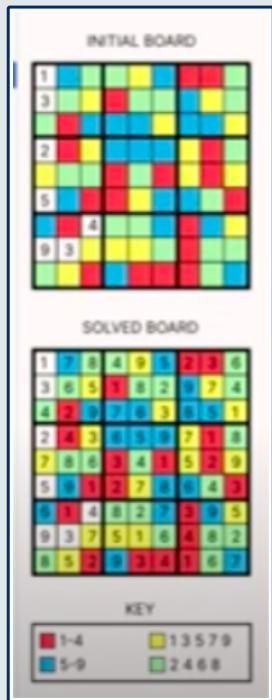
Your answer

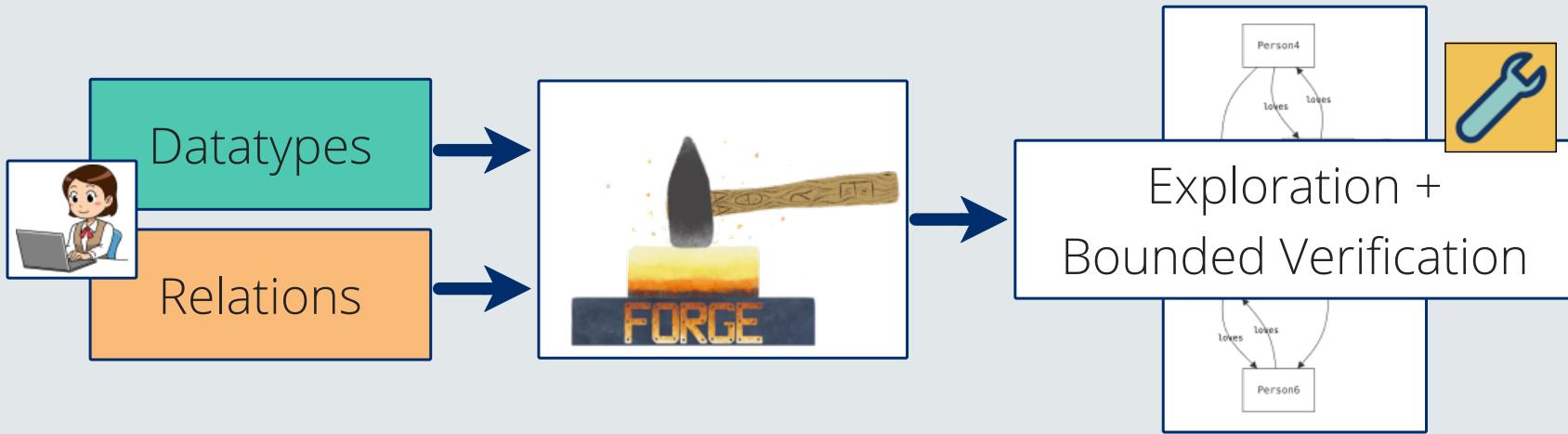
run { some c: Course | c in c.prereqs }

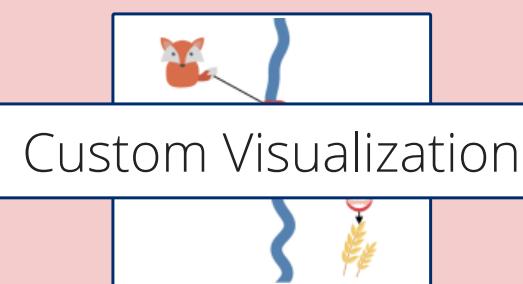
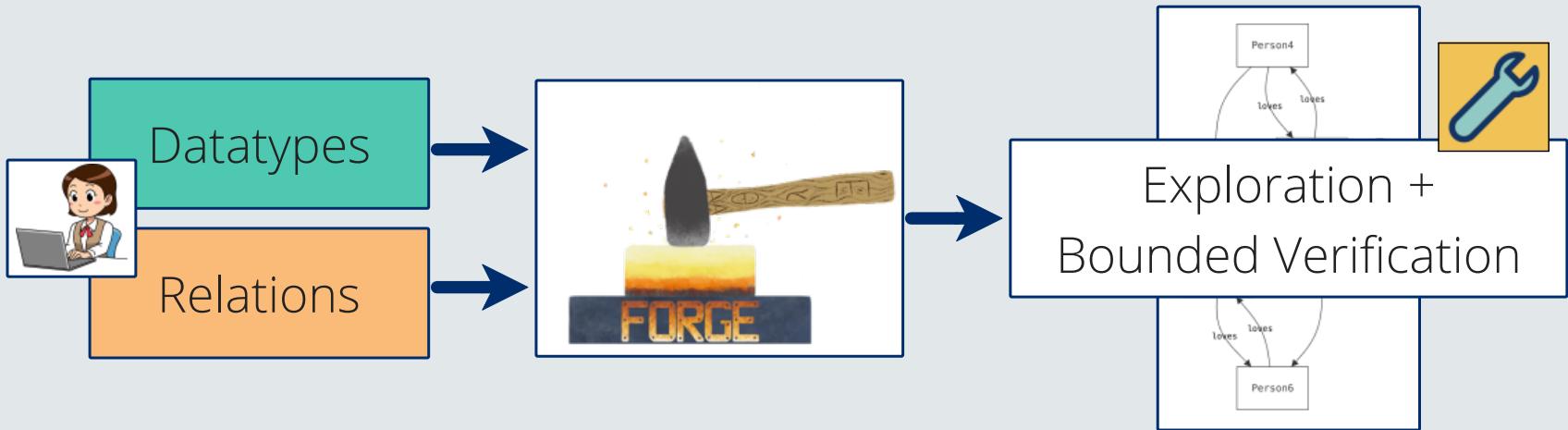
[Copy](#)

Category	Expected and Like	Expected and Dislike	Unexpected and Like	Unexpected and Dislike
SAT	32	14	8	14
UNSAT	15	4	20	24
Error, because the field c.prereqs does not exist	3	2	12	48
Error, because a Course cannot be its own prereq	16	6	15	25

2022	Midterm	Final	2023	Midterm	Final
total	45	33	total	32	26
Froglet	36	0	Froglet	23	2
Relational	8	18	Relational	9	8
Temporal	N/A	13	Temporal	N/A	13
SMT	1	2	SMT	0	3





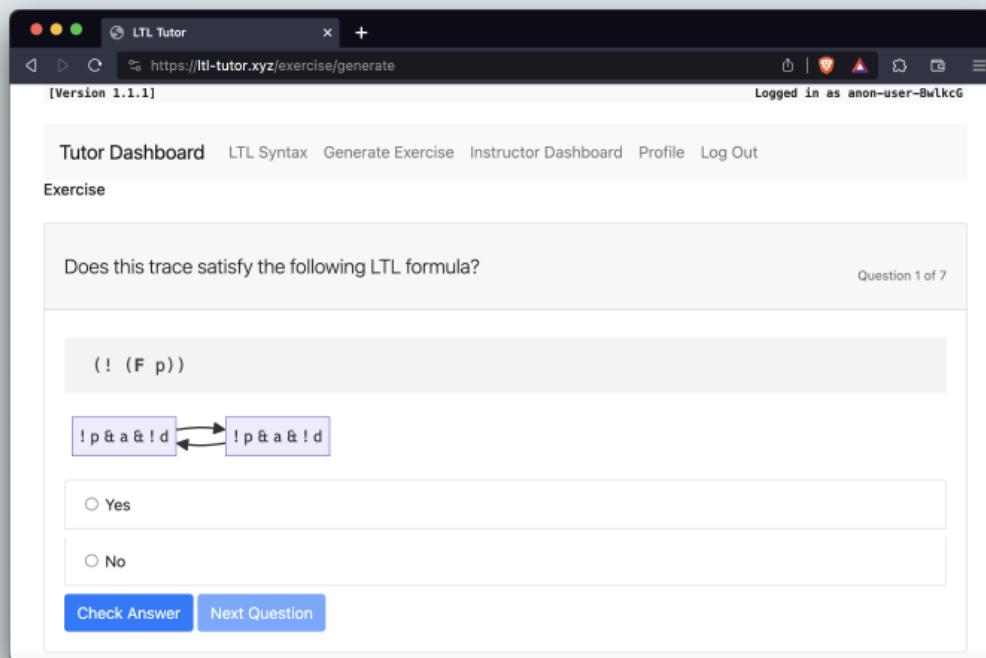




<https://forge-fm.org>

blg@cs.utah.edu

<https://ltl-tutor.xyz>



The screenshot shows a web browser window for the LTL Tutor application. The URL in the address bar is `https://ltl-tutor.xyz/exercise/generate`. The page title is "LTL Tutor". The top navigation bar includes links for "Tutor Dashboard", "LTL Syntax", "Generate Exercise", "Instructor Dashboard", "Profile", and "Log Out". A message at the top right says "Logged in as anon-user-BwlkcG". Below the navigation, a section titled "Exercise" asks: "Does this trace satisfy the following LTL formula?". The formula is $(\neg (\text{F} p))$. Below the formula, two boxes are shown: $\neg p \wedge a \wedge \neg d$ and $\neg p \wedge a \wedge \neg d$, with a double-headed arrow between them. At the bottom, there are two radio buttons: Yes and No. Two blue buttons at the bottom are labeled "Check Answer" and "Next Question".



Siddhartha

