

Forge:

Usable Model-Finding

IETF 120



Ben Greenman

Siddhartha Prasad

Tim Nelson

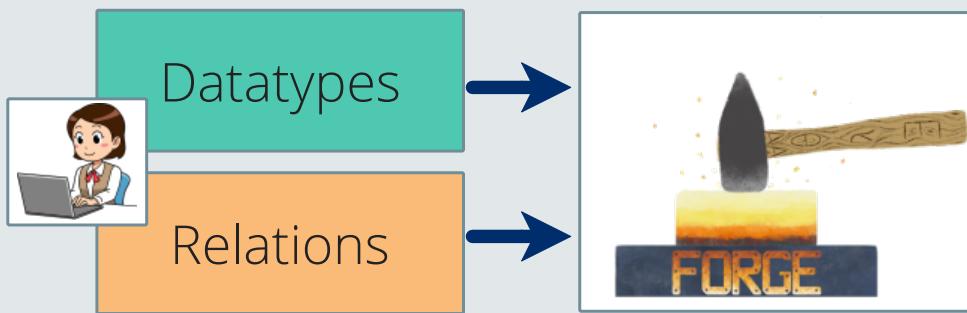
Shriram Krishnamurthi

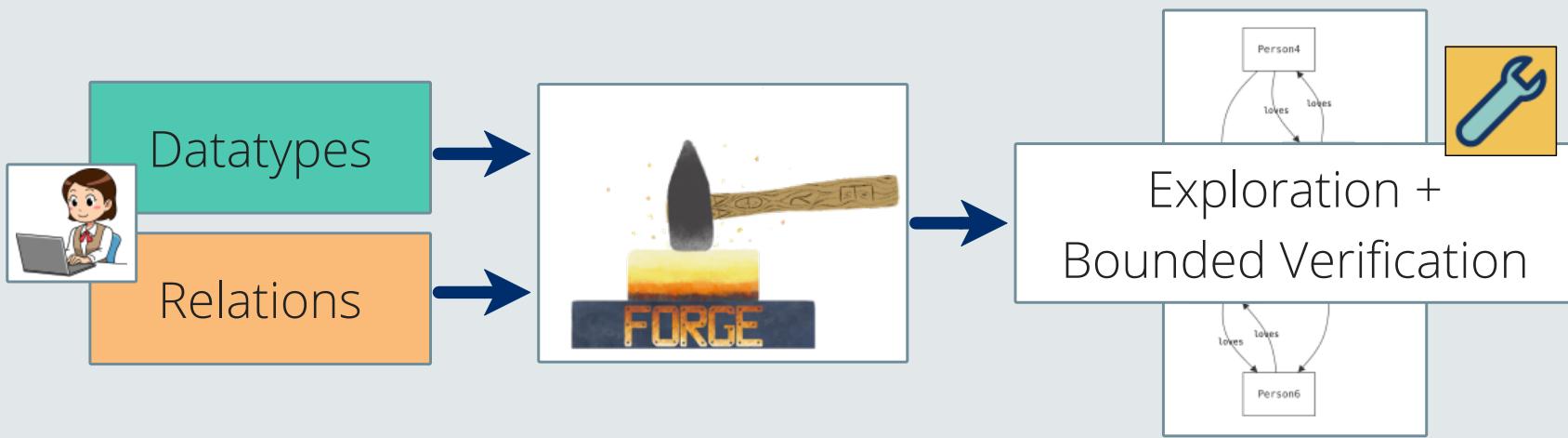


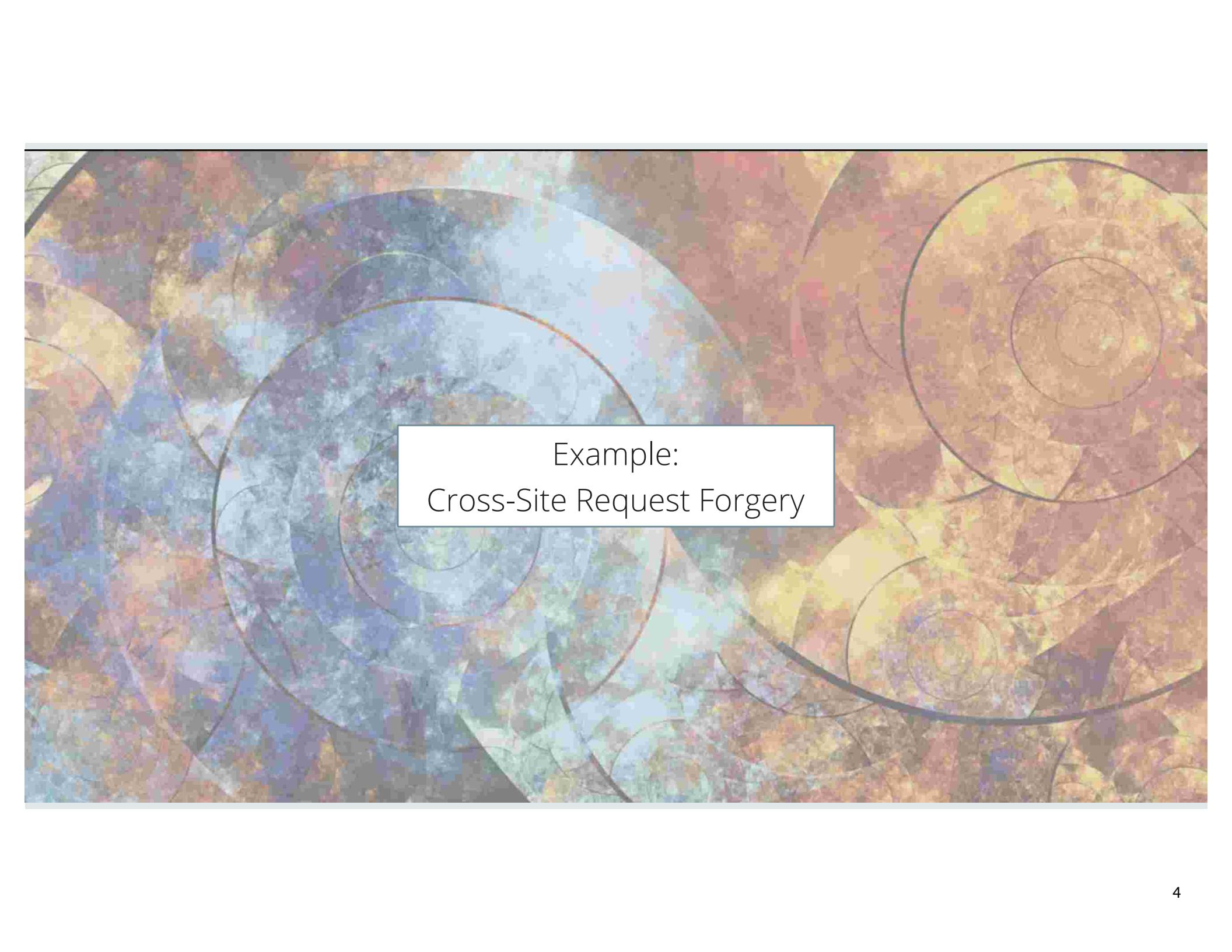
BROWN



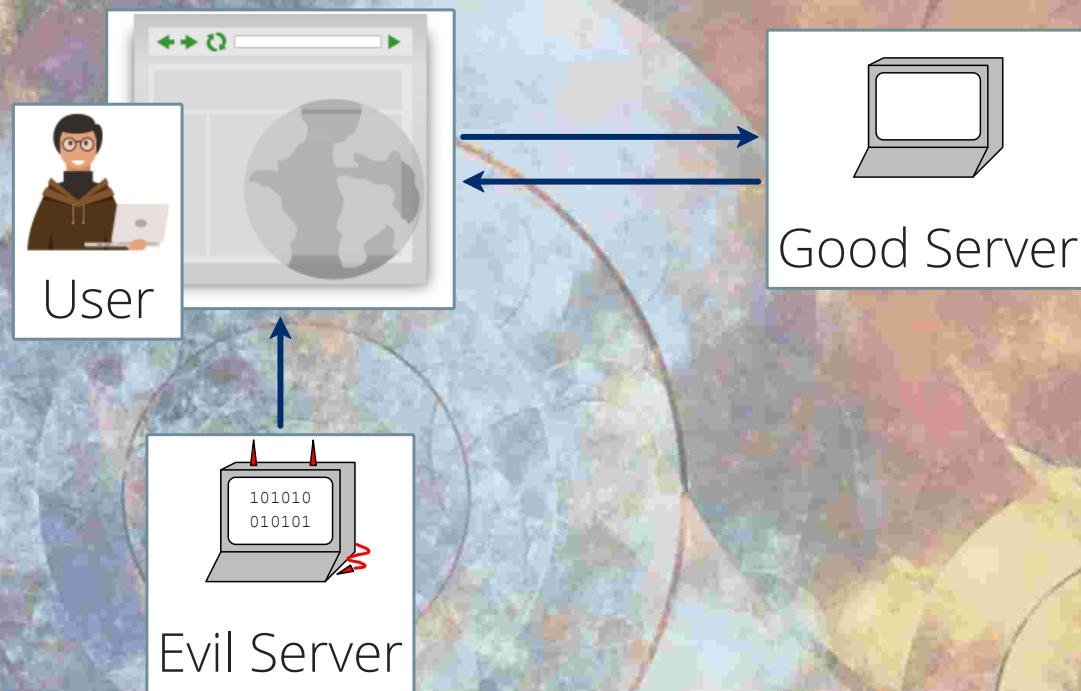
THE
UNIVERSITY
OF UTAH

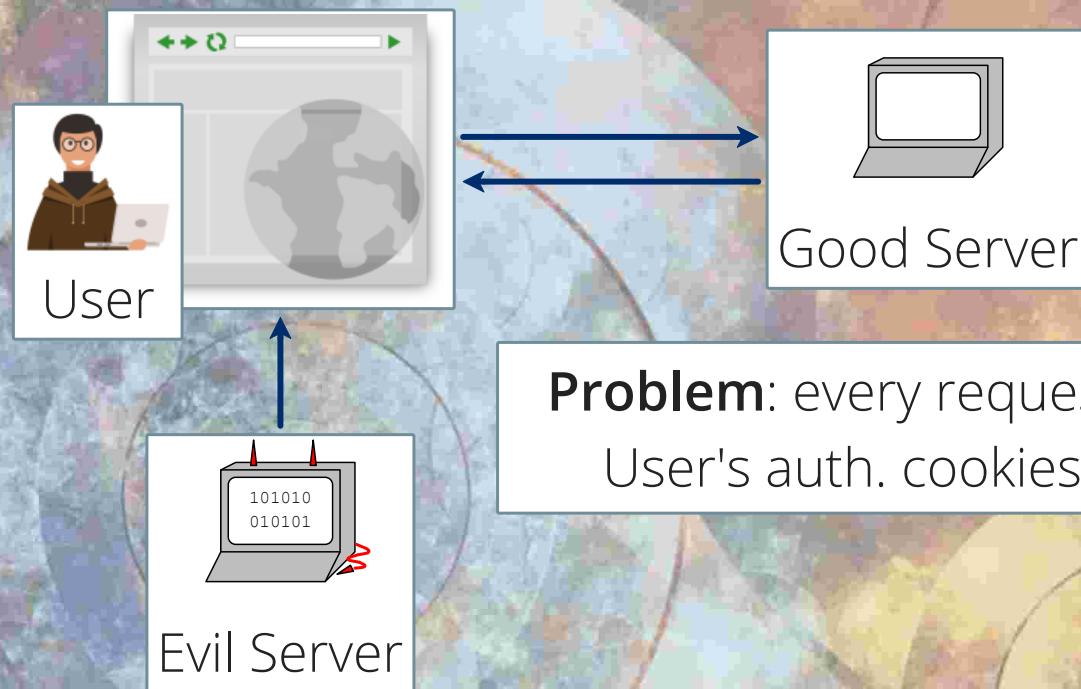




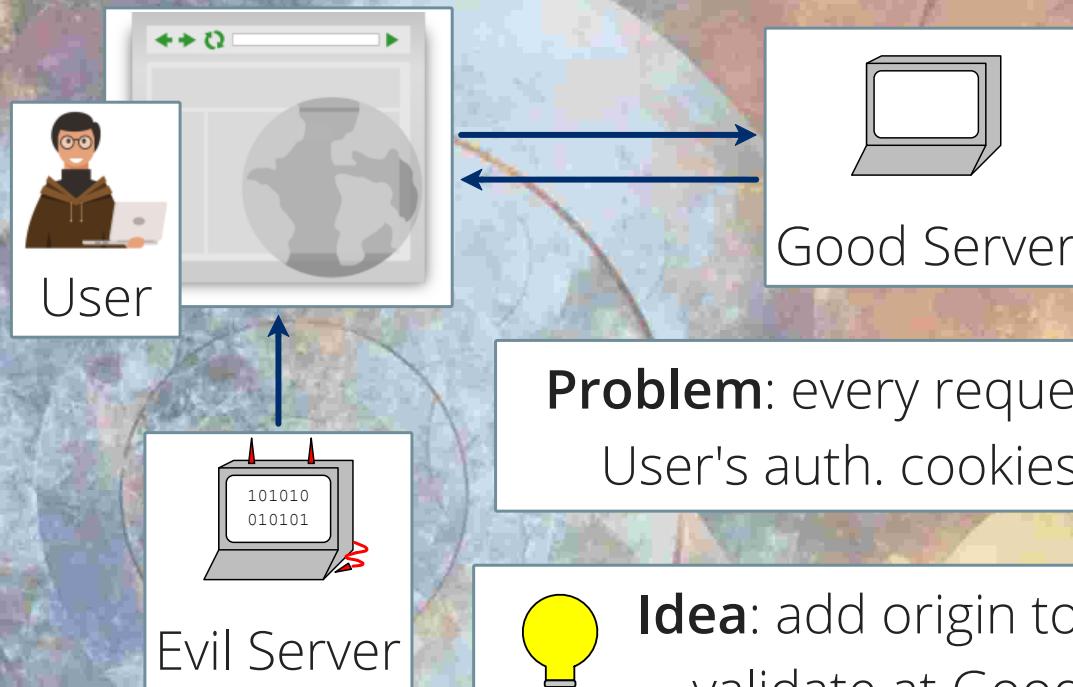


Example:
Cross-Site Request Forgery





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
```

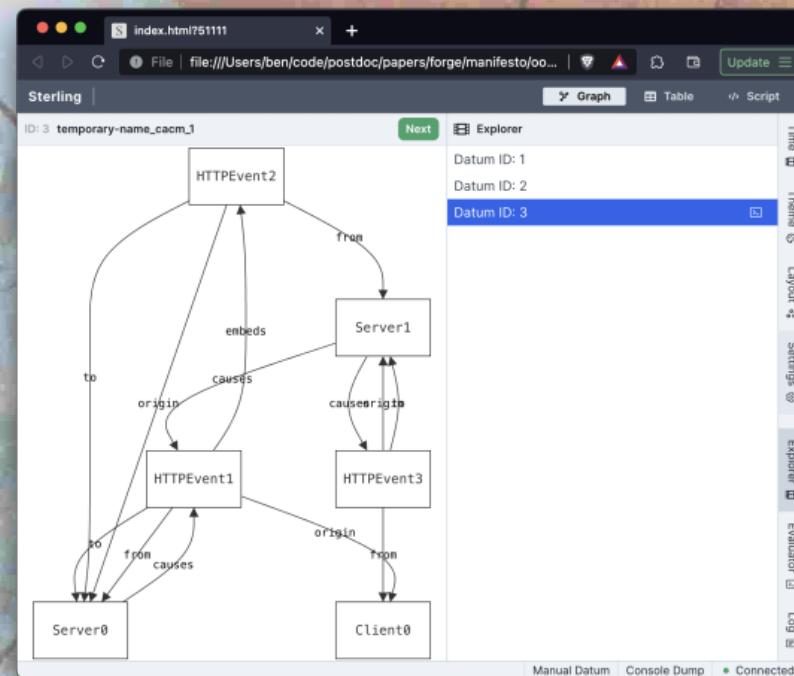
Datatypes

==> Exploration

cacm.frg - DrRacket

```
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
```

Determine language from source custom ▾ 15:1 585.99 MB



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

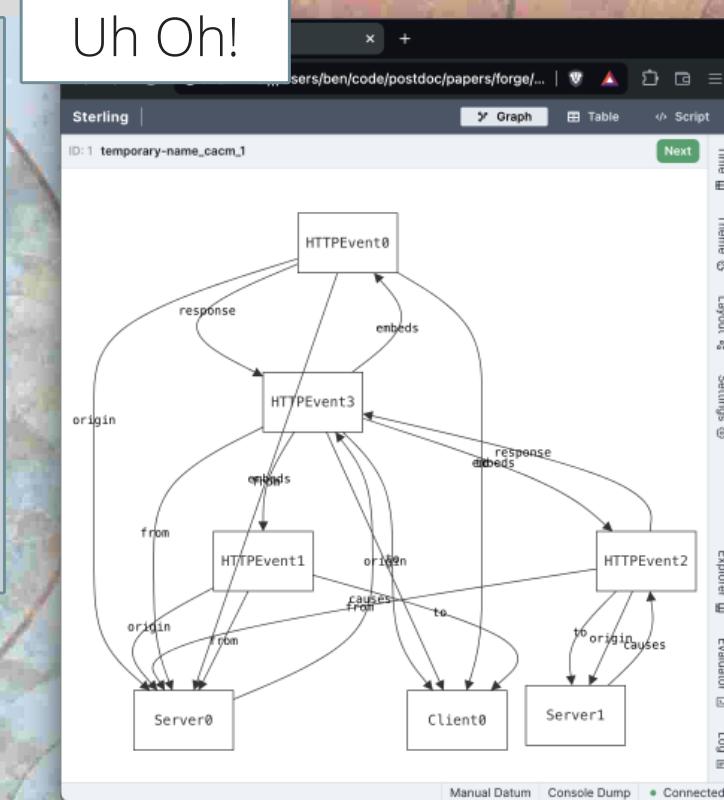
```
run {
    // can we find (hope not)
    some good, bad: Server {
        EnforceOrigins[good]
        // ...
    }
} for exactly 2 Server,
    exactly 1 Client,
    5 HTTPEvent
```

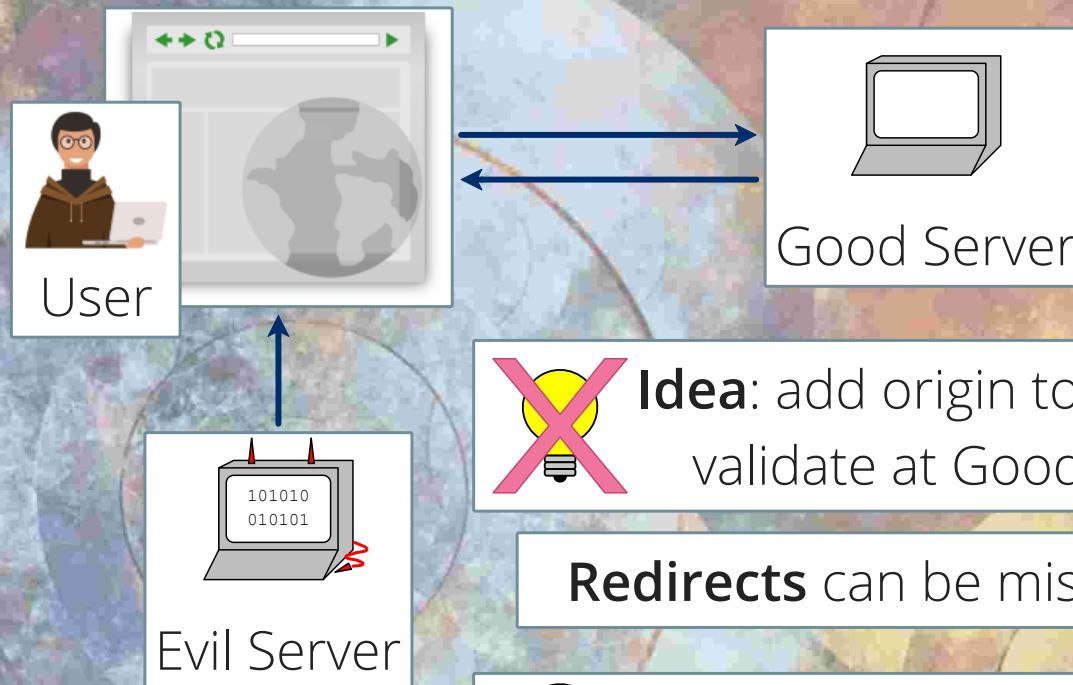
bounds

Checks

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

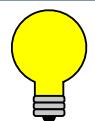
Uh Oh!

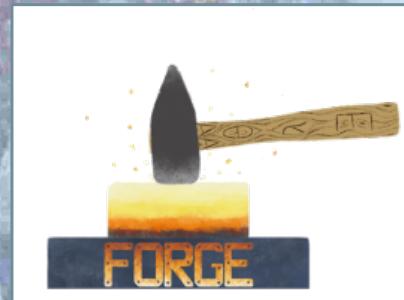




 **Idea:** add origin to requests,
validate at Good Server

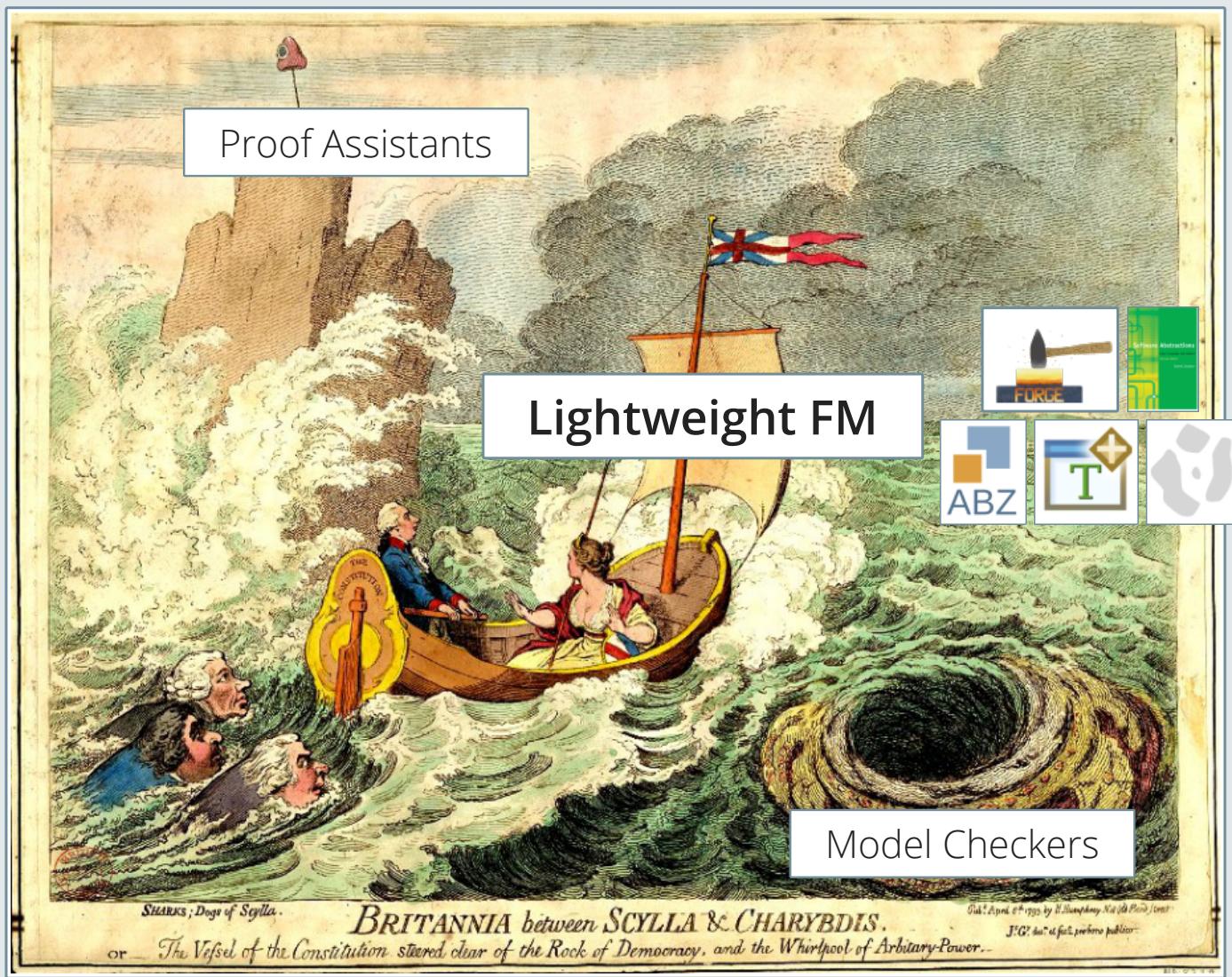
Redirects can be mis-labeled

 How about a set of origins??



Quickly found a bug!





Lightweight FM



Usability >> Completeness

Insight: Most bugs have **small** instances
small scope hypothesis - D. Jackson

What sets Forge apart?



What sets Forge apart?



Custom Visualization



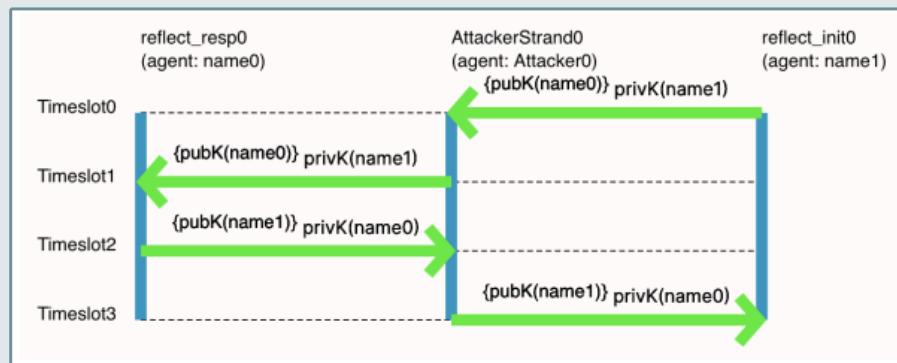
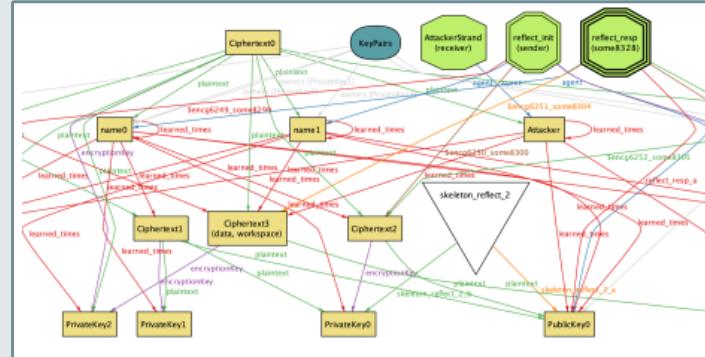
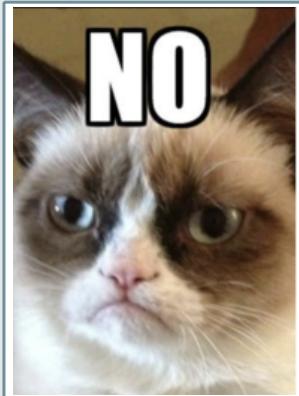
Unit Testing



Language Levels



Custom Visualization



Custom Visualization

Sterling | Run <div> <canvas> <svg> Next Variables Stage Variables

ID: 1 reflect_resp.pov

AttackerStrand0 reflect_resp
(agent: Attacker0) (agent: nam
{pubK(name0)} privK(name

Timeslot0 {pubK(name0)} privK(name1)

Timeslot1 {pubK(name1)} privK(name0)

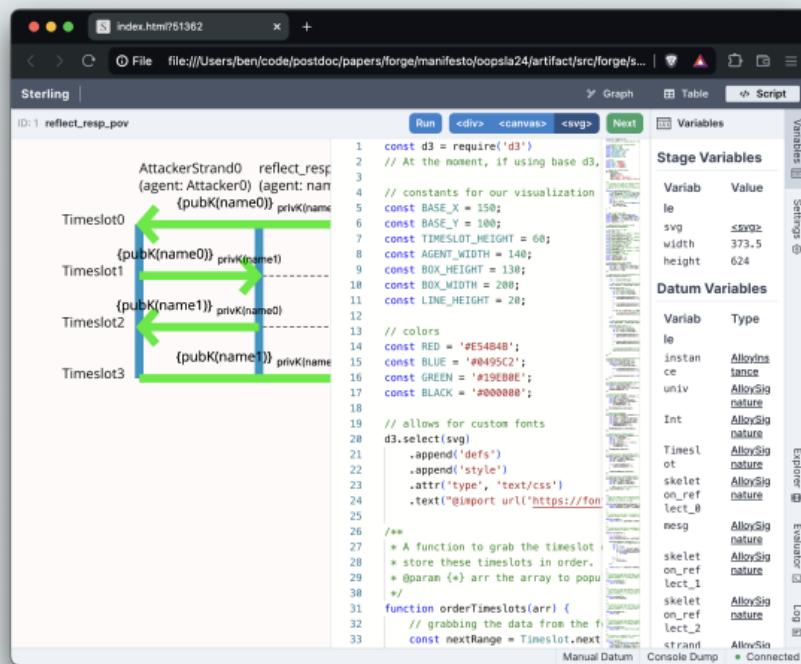
Timeslot2 {pubK(name1)} privK(name0)

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 = '#E54B4B';
15 const BLUE = '#0495C2';
16 const GREEN = '#19EB0E';
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("${https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;family=Open+Sans}");`)
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
```

Manual Datum Console Dump Connected

Custom Visualization



The screenshot shows the Sterling IDE interface. On the left, there is a visualization window titled "reflect_resp_pov" showing a sequence of events across four time slots (Timeslot0, Timeslot1, Timeslot2, Timeslot3). Each time slot contains a green horizontal bar with two boxes: one labeled "pubK(name0)" and another labeled "privK(name1)". Arrows point from the first box in one slot to the second box in the next slot. On the right side of the interface, there is a code editor with the following JavaScript code:

```
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 = '#E54B4B';
15 const BLUE = '#0495C2';
16 const GREEN = '#19E8D0';
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'https://fonts.googleapis.com/css?family=Open+Sans:400,700&display=block';")
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 first
33   const nextRange = Timeslot.next
```

Much more than pretty pictures!
Building on decades of CogSci research



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

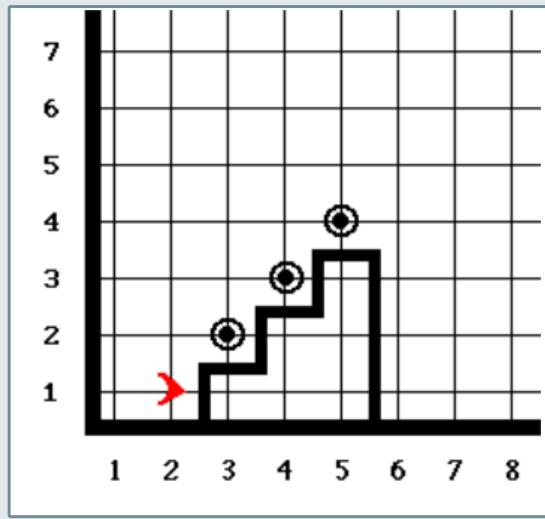
assert

test suite

test expect

How do we know the **model**
is correct?

Unit Testing



Challenge: Programming != Modeling

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/temporal
++ Linear Temporal Logic

#lang forge/relational
++ N-ary Relations

#lang forge/bsl
Functional Relations

In what ways is LTL difficult to use?

+3 years of studies with researchers and students



Categories of LTL Errors

Bad Prop

Implicit G

Bad State Index

Implicit Prefix

Bad State Quantification

Other Implicit

Cycle G

Trace Split U

Exclusive U

Spreading X

Implicit F

Weak U

Q. Translate to LTL:

The green light turns on exactly once

F = eventually **G** = always **X** = next state

Q. Translate to LTL:

The green light turns on exactly once

F = eventually **G** = always **X** = next state

F(green) & G(green => X(!green))

Q. Translate to LTL:

The green light turns on exactly once

F = eventually **G** = always **X** = next state

X

F(green) & G(green => X(!green))

F(green) & G(green => X(G(!green)))

Q. Translate to LTL:

The green light turns on exactly once

F = eventually **G** = always **X** = next state

X

F(green) & G(green => X(!green))

✓

F(green) & G(green => X(G(!green)))

Q. Translate to LTL:

The green light turns on exactly once

F = eventually **G** = always **X** = next state



F(green) & G(green => X(!green))

Implicit G



F(green) & G(green => X(G(!green)))

<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 [Version 1.1.1]".

The main content area is titled "Exercise". A question is displayed: "Does this trace satisfy the following LTL formula?". The formula is $(\neg (\text{F} p))$. To the right, it says "Question 1 of 7".

Below the formula, there is a sequence of states represented by boxes: $\neg p \wedge a \wedge \neg d$, followed by a double-headed arrow, and then $\neg p \wedge a \wedge \neg d$. This indicates that the trace is periodic or has a loop.

Two radio buttons are available for the answer: Yes and No.

At the bottom are two buttons: "Check Answer" and "Next Question".

<https://ltl-tutor.xyz>

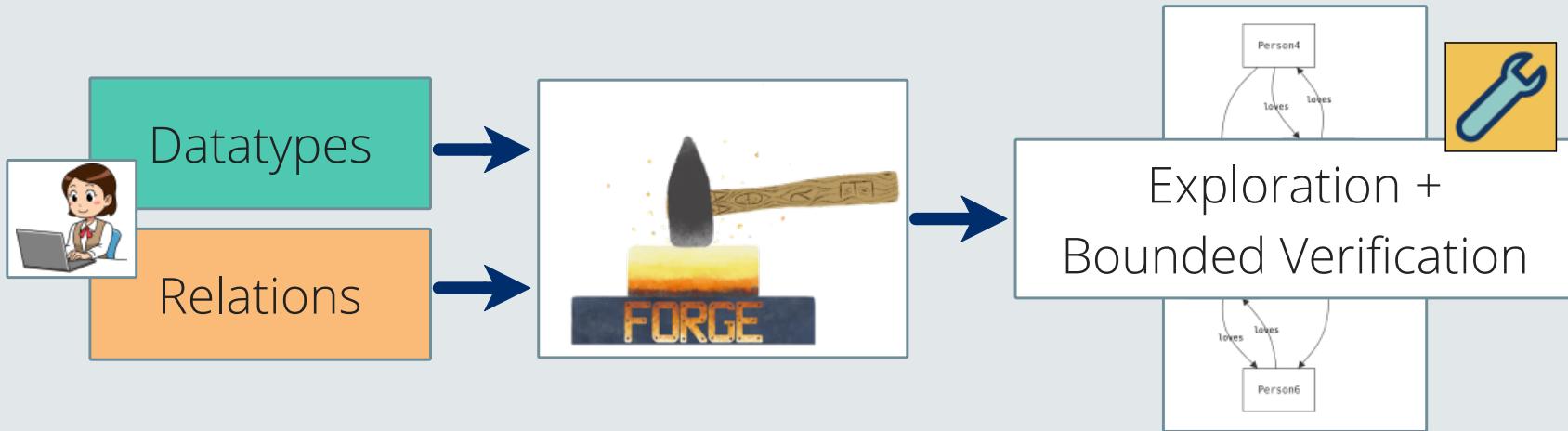
The image displays two side-by-side screenshots of the LTL Tutor web application, version 1.1.1, running in a browser window.

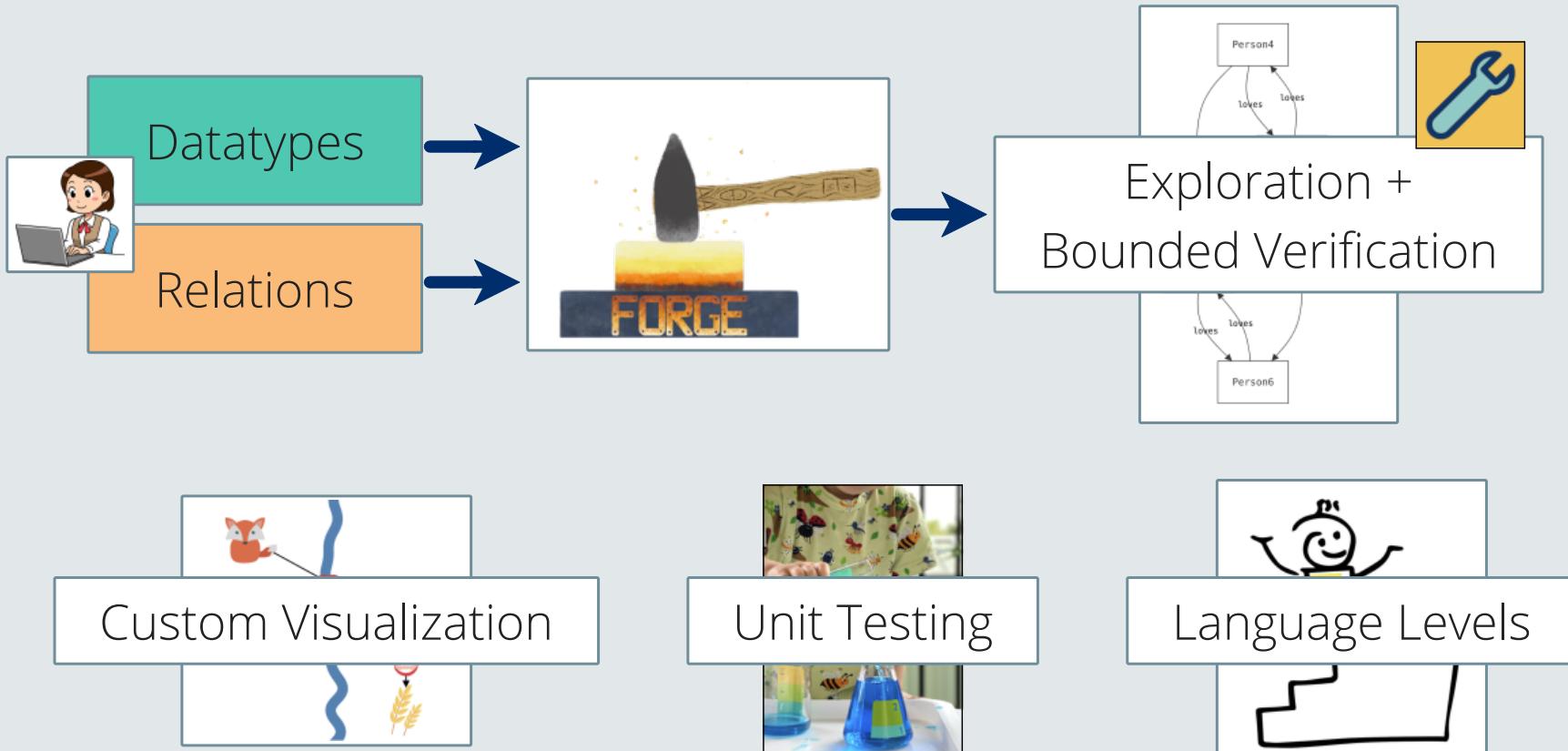
Left Screenshot (Exercise View):

- The title bar says "LTL Tutor [Version 1.1.1]".
- The main content area asks, "Does this trace satisfy the formula $(\neg (\mathbf{F} p))$?".
- A state transition diagram shows a sequence of states: !p → p → !a → 1. A double-headed arrow connects the !p and p states, and another between !a and 1.
- Below the diagram are two radio buttons: Yes and No.
- At the bottom are "Check Answer" and "Next Question" buttons.

Right Screenshot (Feedback View):

- The title bar says "LTL Tutor [Version 1.1.1]".
- The main content area starts with "Tutor Dashboard" and other navigation links.
- A message states: "That's not correct 😞 Don't worry, keep trying! The correct answer is highlighted in green (i.e: $(\mathbf{X} (p \rightarrow (\mathbf{X} a)))$)".
- Text below explains: "Your selection is more permissive than the correct answer. Here is a trace that satisfies your selection, but not the correct answer:" followed by the same state transition diagram as the left screenshot.
- An "Alt Trace" is shown: $!p;p;!a;\text{cycle}\{1;1\}$.
- A Venn diagram compares "Correct answer" (green circle) and "Your answer" (red circle overlapping it).
- At the bottom are "Check Answer" and "Next Question" buttons.







<https://forge-fm.org>

blg@cs.utah.edu

