

Forge:

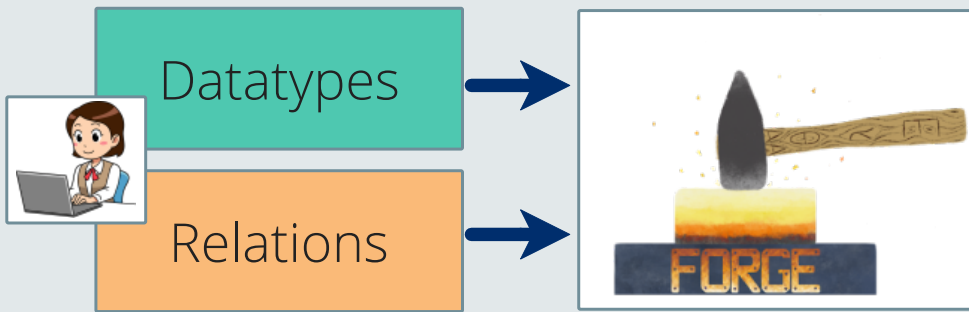
Usable Model-Finding

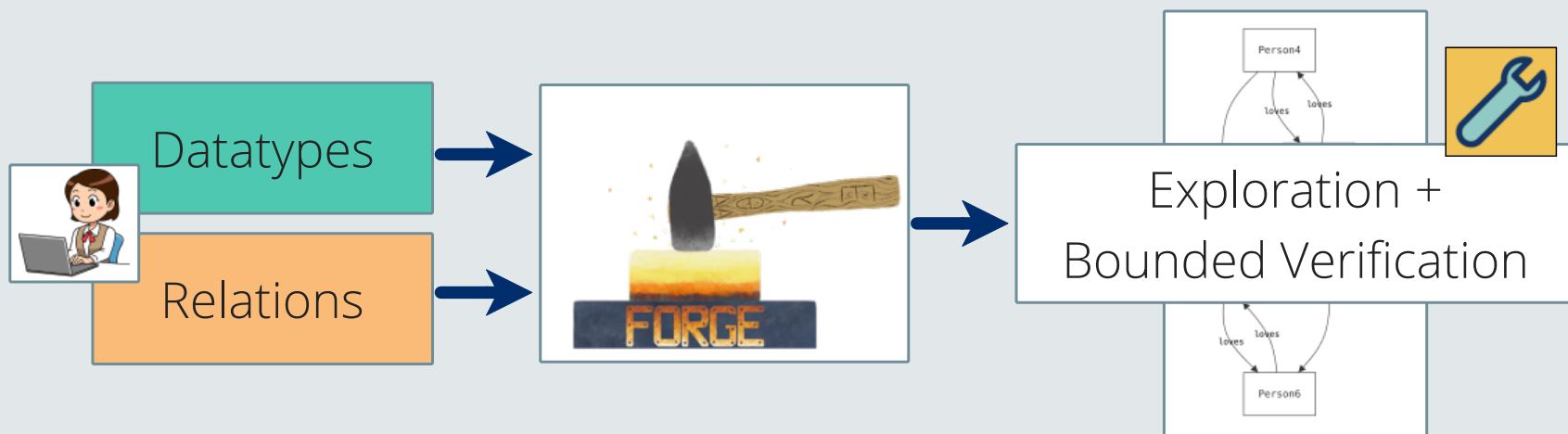
IETF 120



Ben Greenman
Siddhartha Prasad
Tim Nelson
Shriram Krishnamurthi

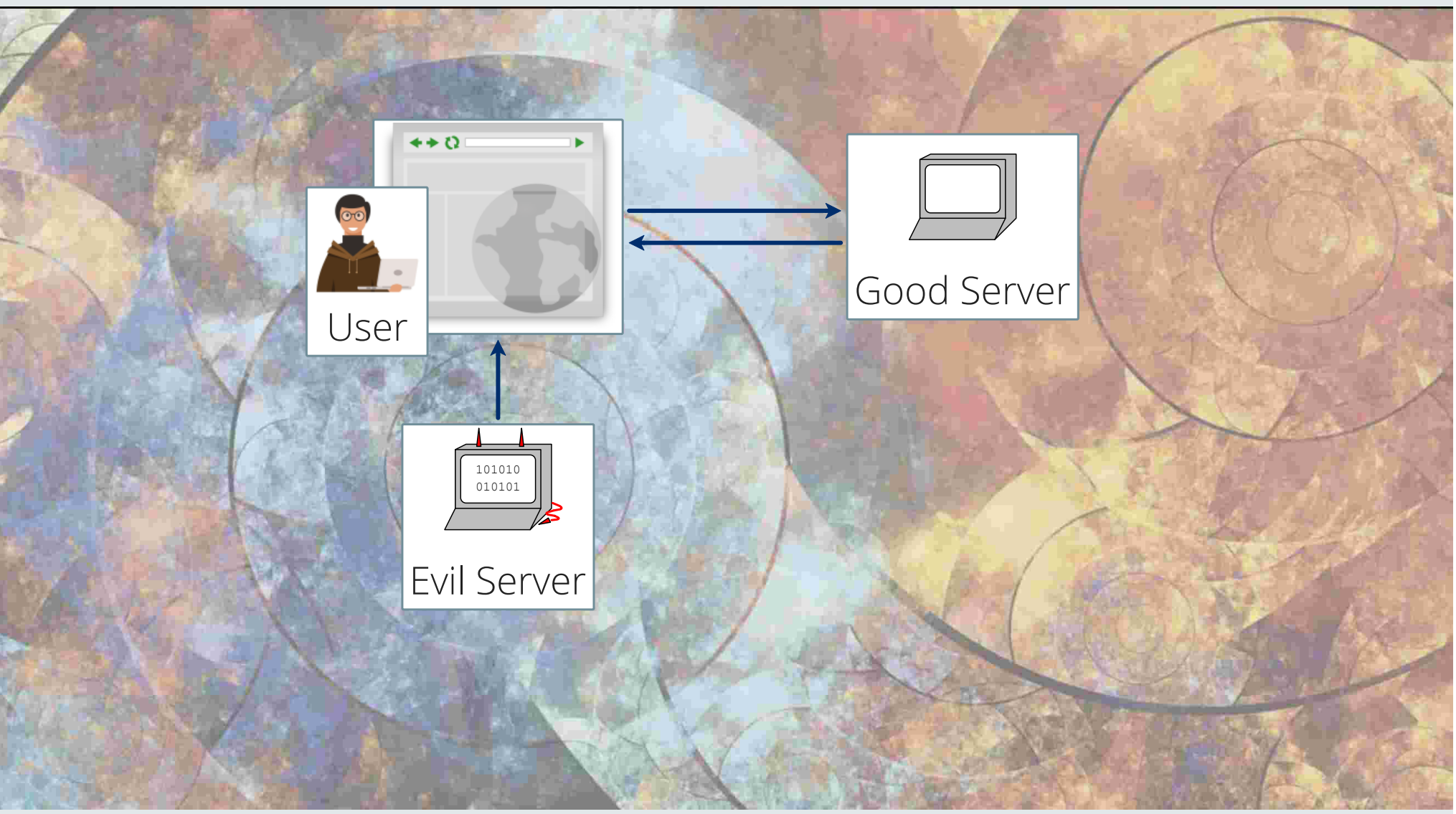


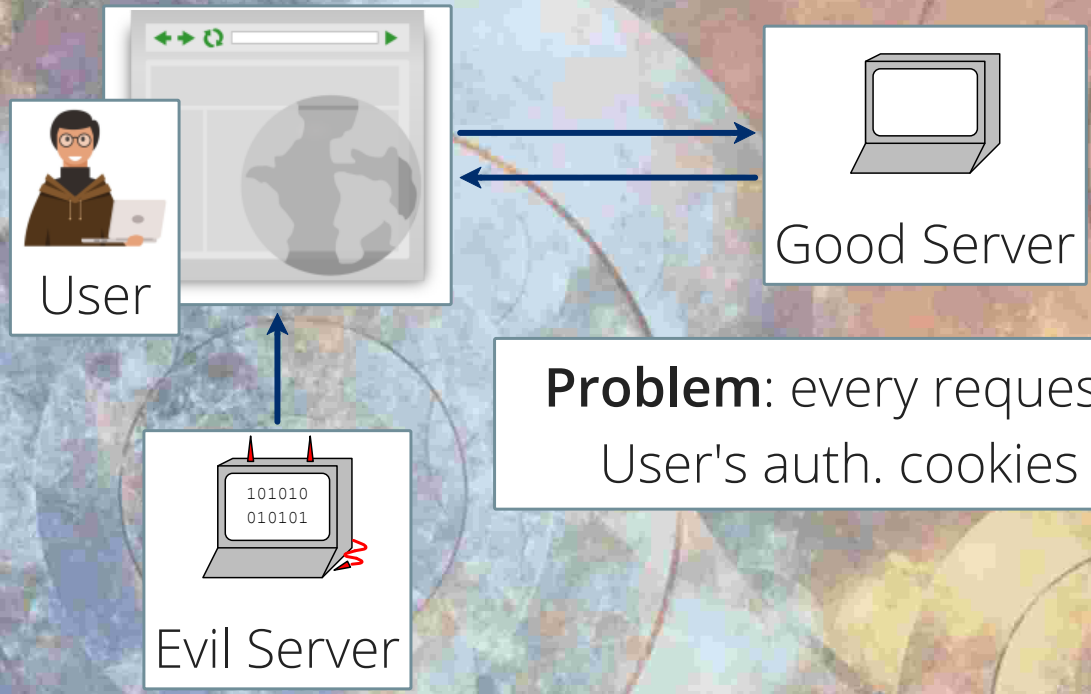


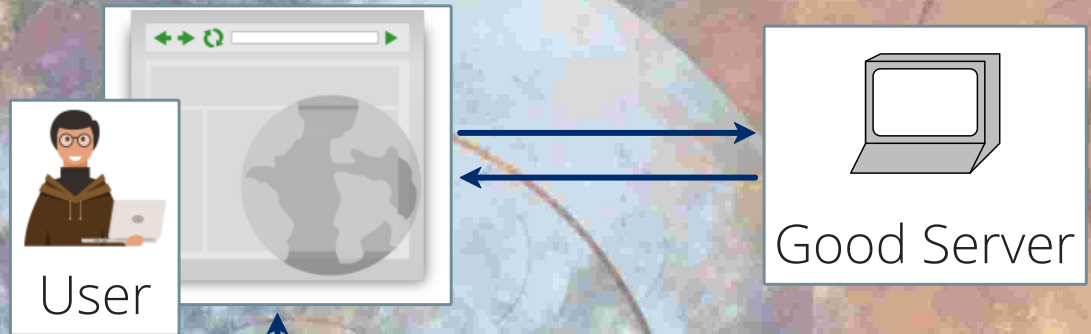




Example:
Cross-Site Request Forgery







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

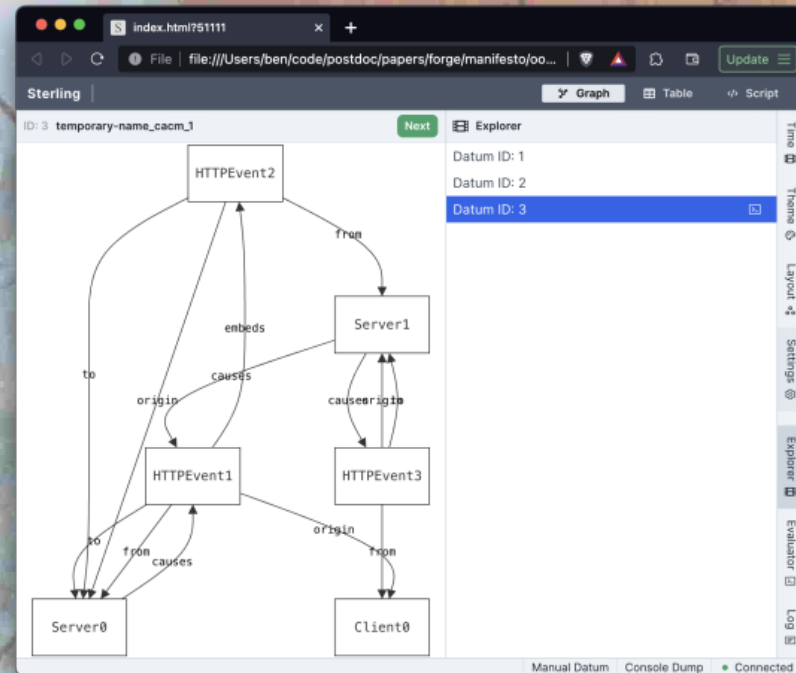
```
abstract sig EndPoint {}  
  
sig Client  
  extends EndPoint {}  
  
sig Server  
  extends EndPoint {  
    causes: set HTTPEvent  
  }
```

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

Datatypes

==> Exploration

```
cacm.frg - DrRacket
cacm.frg (define ...) 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
```



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

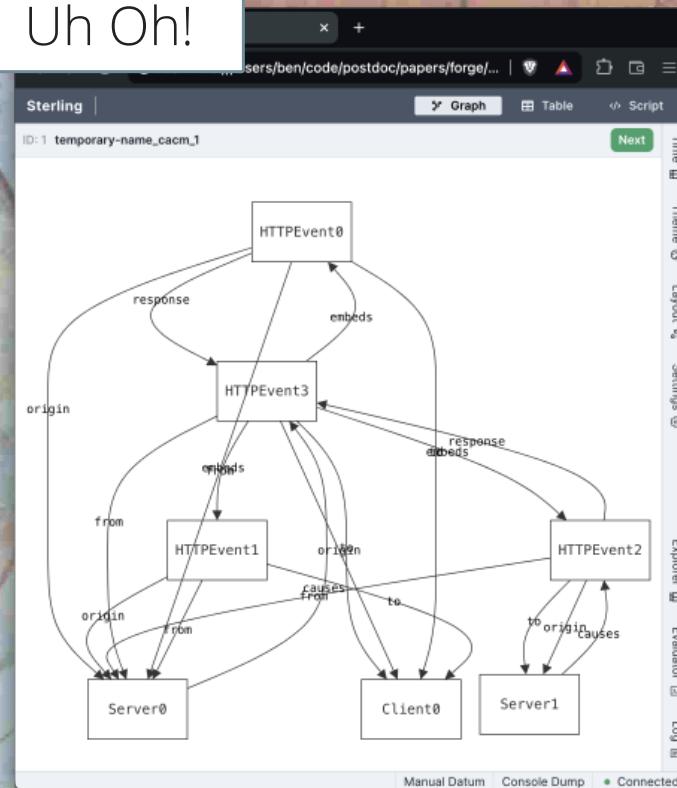
```
run {  
  // can we find (hope not)  
  some good, bad: Server {  
    EnforceOrigins[good]  
    // ...  
  }  
} for exactly 2 Server,  
  exactly 1 Client,  
  5 HTTPEvent
```

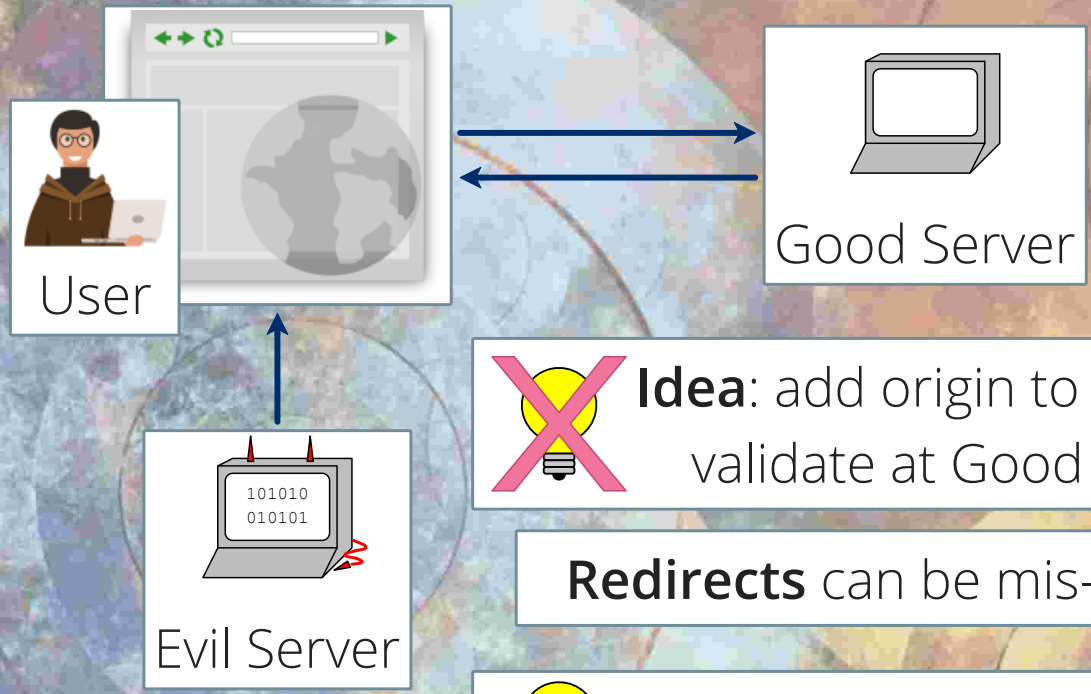
bounds

Checks

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

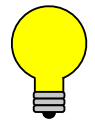
Uh Oh!





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

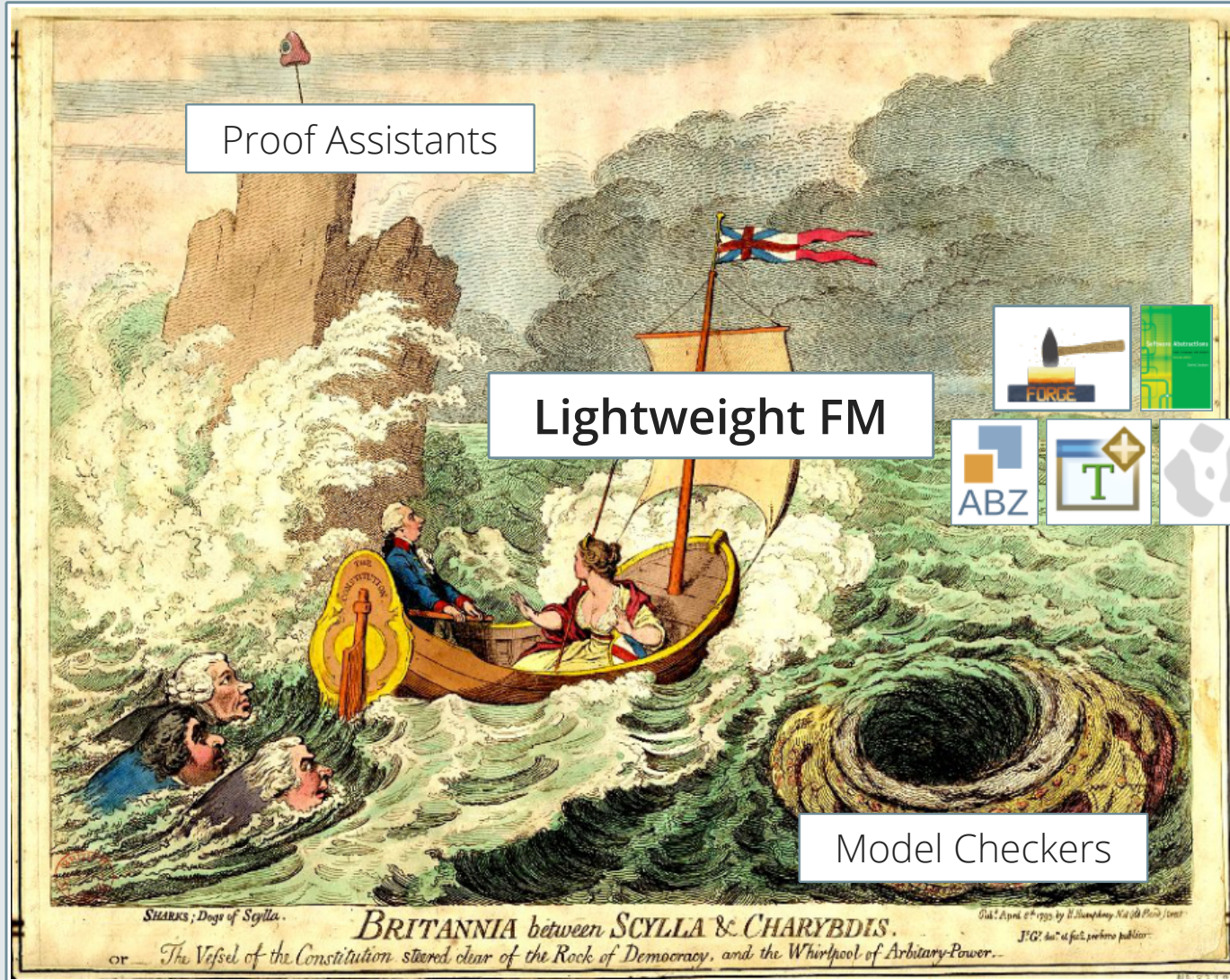
Redirects can be mis-labeled

 How about a set of origins??



Quickly found a bug!





Proof Assistants

Lightweight FM

Model Checkers



SHARKS; Dogs of Scylla.

BRITANNIA between SCYLLA & CHARYBDIS.

Printed & sold by H. Bunbury, No. 10, Pall Mall, 1795.

or The Vessel of the Constitution steered clear of the Rock of Democracy, and the Whirlpool of Arbitrary Power.

J.G. del. et fecit. per bene publicum.

Lightweight FM



Usability >> Completeness

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

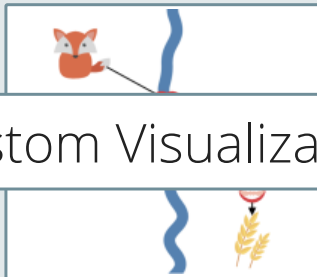
What sets Forge apart?



What sets Forge apart?



Custom Visualization



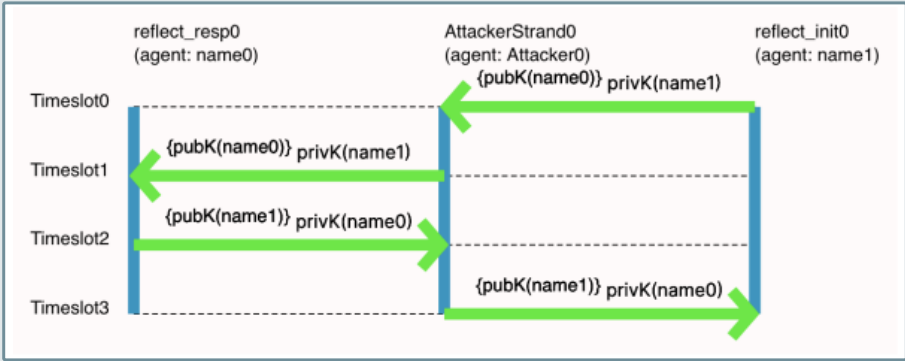
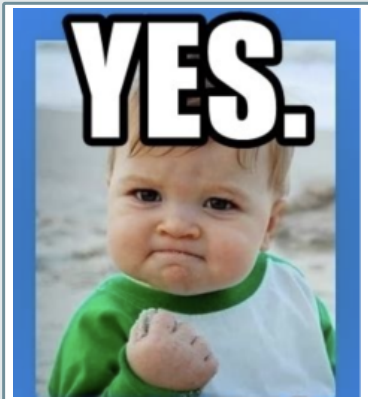
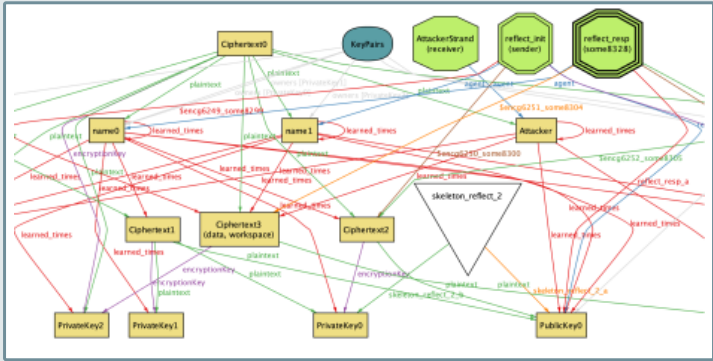
Unit Testing



Language Levels



Custom Visualization



Custom Visualization

The screenshot shows a web browser window with a custom visualization. The visualization is a diagram with four horizontal timeslots labeled Timeslot0, Timeslot1, Timeslot2, and Timeslot3. Green arrows indicate data flow: Timeslot0 to Timeslot1, Timeslot1 to Timeslot2, and Timeslot2 to Timeslot3. A vertical blue line is positioned between Timeslot1 and Timeslot2. The code editor on the right shows 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 = '#4495C2';
16 const GREEN = '#19E88E';
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://font');`);
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
```

The sidebar on the right shows the following variables:

Stage Variables	Value
le	svg
width	373.5
height	624

Datum Variables	Type
le	instan
ce	AllowSig
univ	AllowSig
Int	AllowSig
Timeslot	AllowSig
skelet	AllowSig
on_ref	AllowSig
lect_0	AllowSig
mesq	AllowSig
skelet	AllowSig
on_ref	AllowSig
lect_1	AllowSig
skelet	AllowSig
on_ref	AllowSig
lect_2	AllowSig
strand	AllowSig

Custom Visualization

```
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 = '#4495C2';
16 const GREEN = '#39E88E';
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://font');
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
```

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

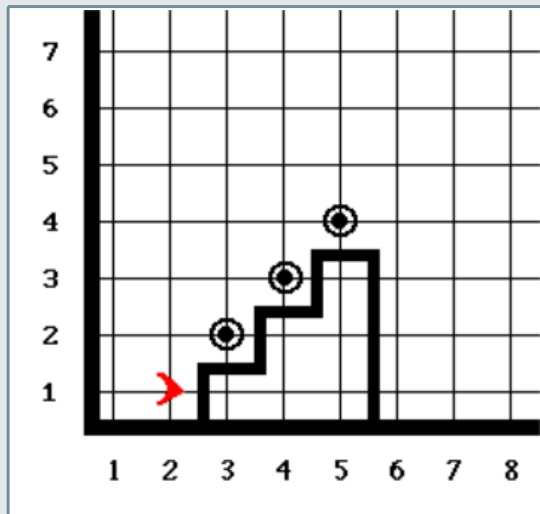
assert

test suite

test expect

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

Unit Testing



Challenge: Programming \neq 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



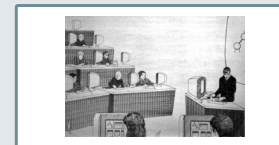
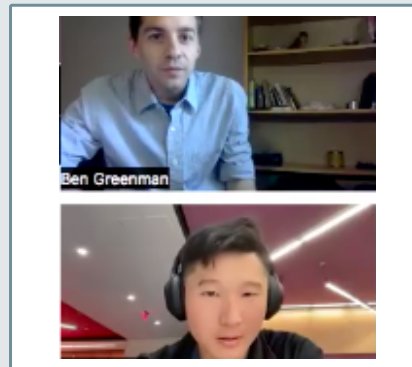
#lang forge/temporal
++ Linear Temporal Logic

#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

$F(\text{green}) \ \& \ G(\text{green} \Rightarrow X(\! \text{green}))$

Q. Translate to LTL:

The green light turns on exactly once

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

X

$F(\text{green}) \ \& \ G(\text{green} \Rightarrow X(\! \text{green}))$

$F(\text{green}) \ \& \ G(\text{green} \Rightarrow X(G(\! \text{green})))$

Q. Translate to LTL:

The green light turns on exactly once

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

X

$F(\text{green}) \ \& \ G(\text{green} \Rightarrow X(\! \text{green}))$

✓

$F(\text{green}) \ \& \ G(\text{green} \Rightarrow X(G(\! \text{green})))$

Q. Translate to LTL:

The green light turns on exactly once

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

✗

$F(\text{green}) \ \& \ G(\text{green} \Rightarrow X(\neg \text{green}))$

Implicit G

✓

$F(\text{green}) \ \& \ G(\text{green} \Rightarrow X(G(\neg \text{green})))$

<https://ltl-tutor.xyz>

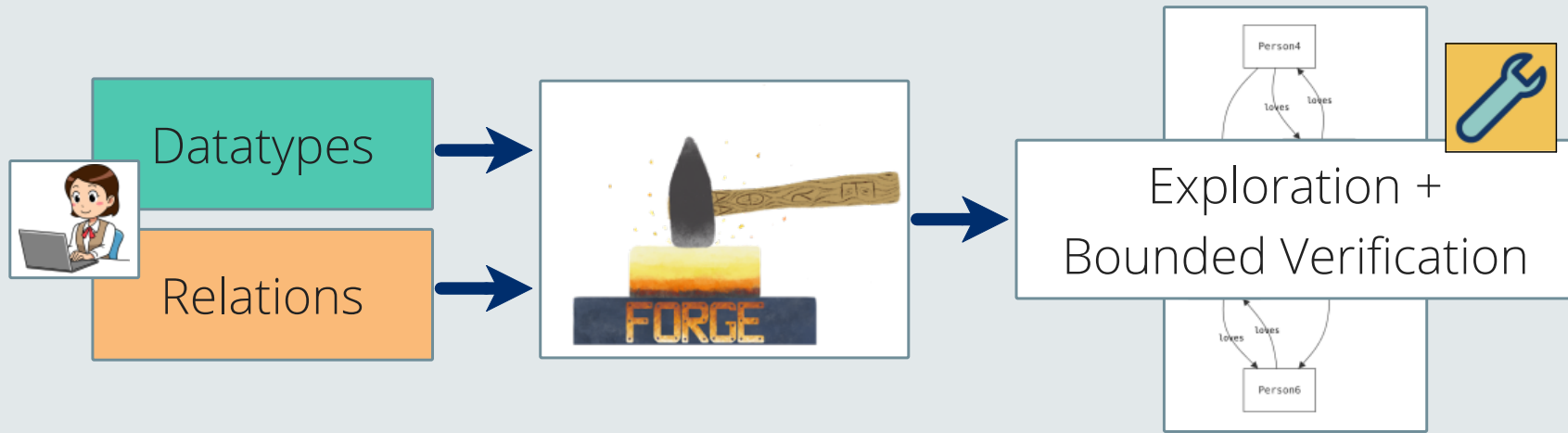
The screenshot shows a web browser window with the URL `https://ltl-tutor.xyz/exercise/generate`. The page title is "LTL Tutor" and the version is "1.1.1". The user is logged in as "anon-user-BwLkG". The navigation menu includes "Tutor Dashboard", "LTL Syntax", "Generate Exercise", "Instructor Dashboard", "Profile", and "Log Out".

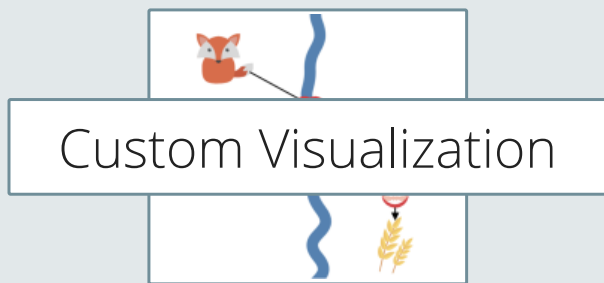
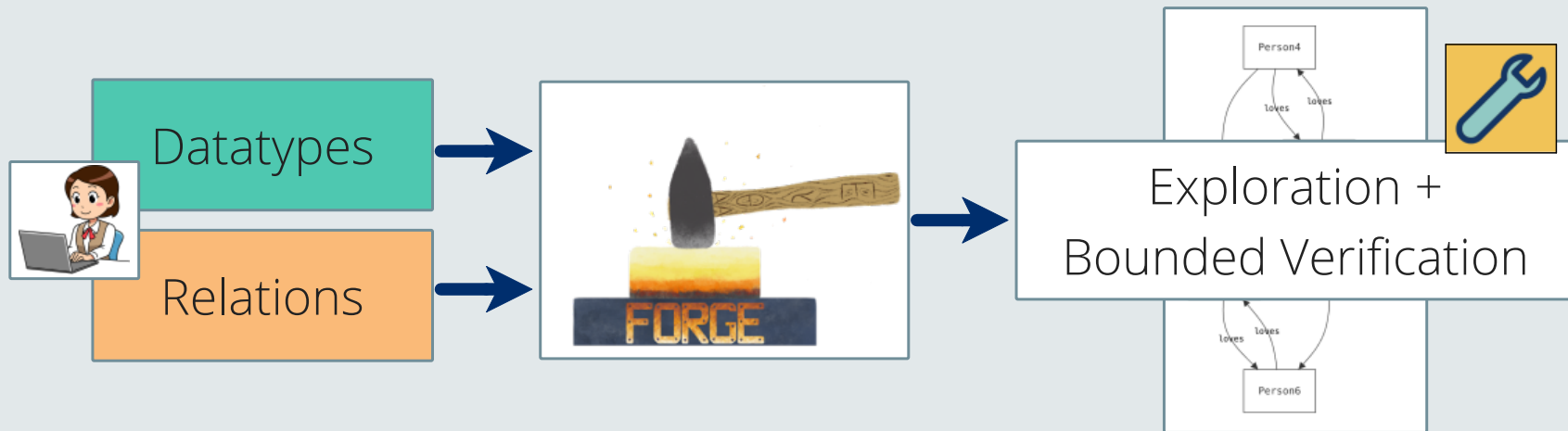
The main content area is titled "Exercise" and contains the question: "Does this trace satisfy the following LTL formula?". The formula is $(! (F p))$. The trace is a sequence of two states, both labeled `!p & a & !d`, connected by a double-headed arrow, indicating a loop.

Below the trace, there are two radio button options: "Yes" and "No". At the bottom, there are two buttons: "Check Answer" and "Next Question".

<https://ltl-tutor.xyz>

The image shows two overlapping browser windows of the LTL Tutor application. The foreground window displays a feedback message for an incorrect answer. The message reads: "That's not correct 😞 Don't worry, keep trying! The correct answer is highlighted in green (i.e. $(X (p \rightarrow (X a)))$)". Below this, it states: "Your selection is more permissive than the correct answer. Here is a trace that satisfies your selection, but not the correct answer:". A state transition diagram shows a sequence of states: $!p \rightarrow p \rightarrow !a \rightarrow 1 \leftrightarrow 1$. Below the diagram, the text "Alt Trace: $!p;p;!a;cycle\{1;1\}$ " is shown. At the bottom, a Venn diagram consists of two overlapping circles: a green circle labeled "Correct answer" and a red circle labeled "Your answer". The intersection of the two circles is shaded, indicating that the user's answer is more permissive than the correct one.







<https://forge-fm.org>

blg@cs.utah.edu

