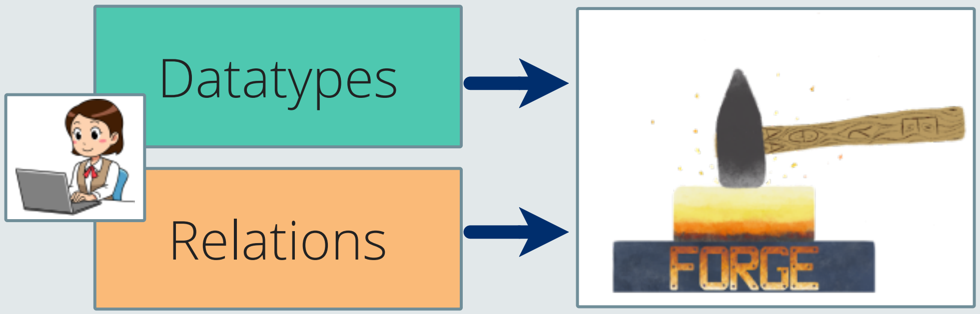
Forge:

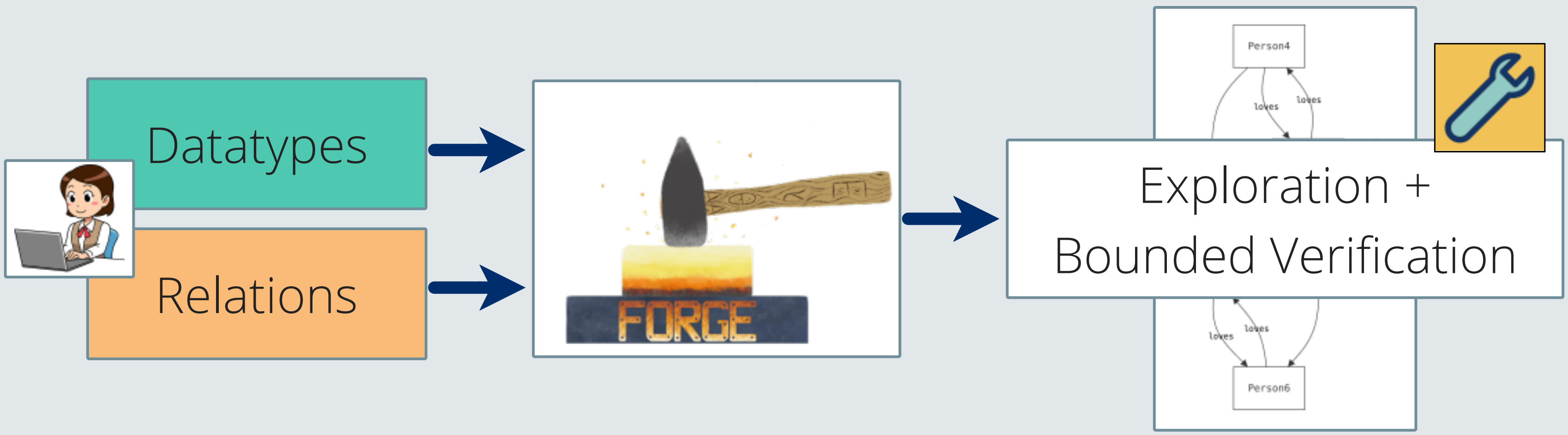
Usable Model-Finding

IETF 120

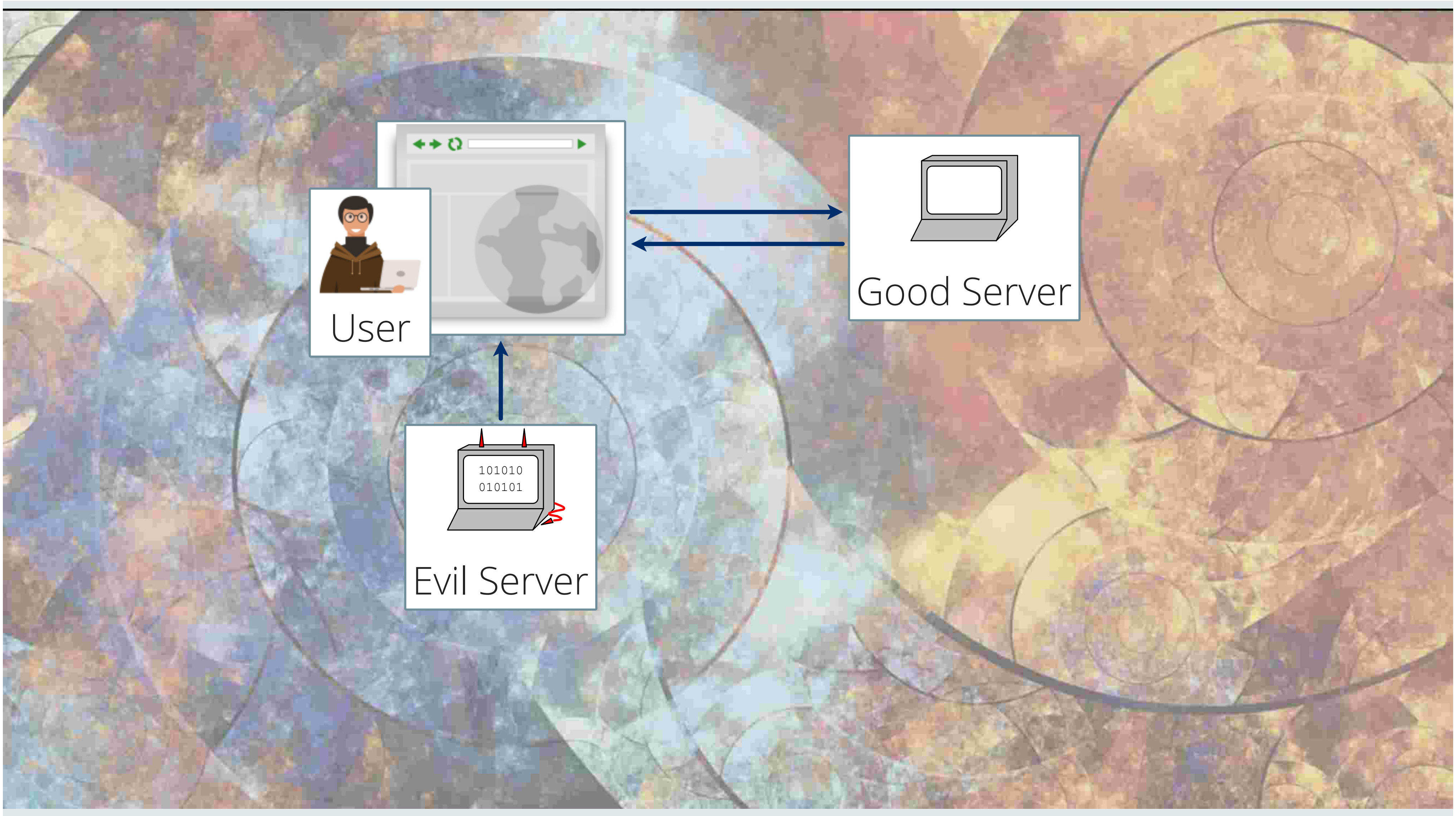
Ben Greenman
Siddhartha Prasad
Tim Nelson
Shriram Krishnamurthi

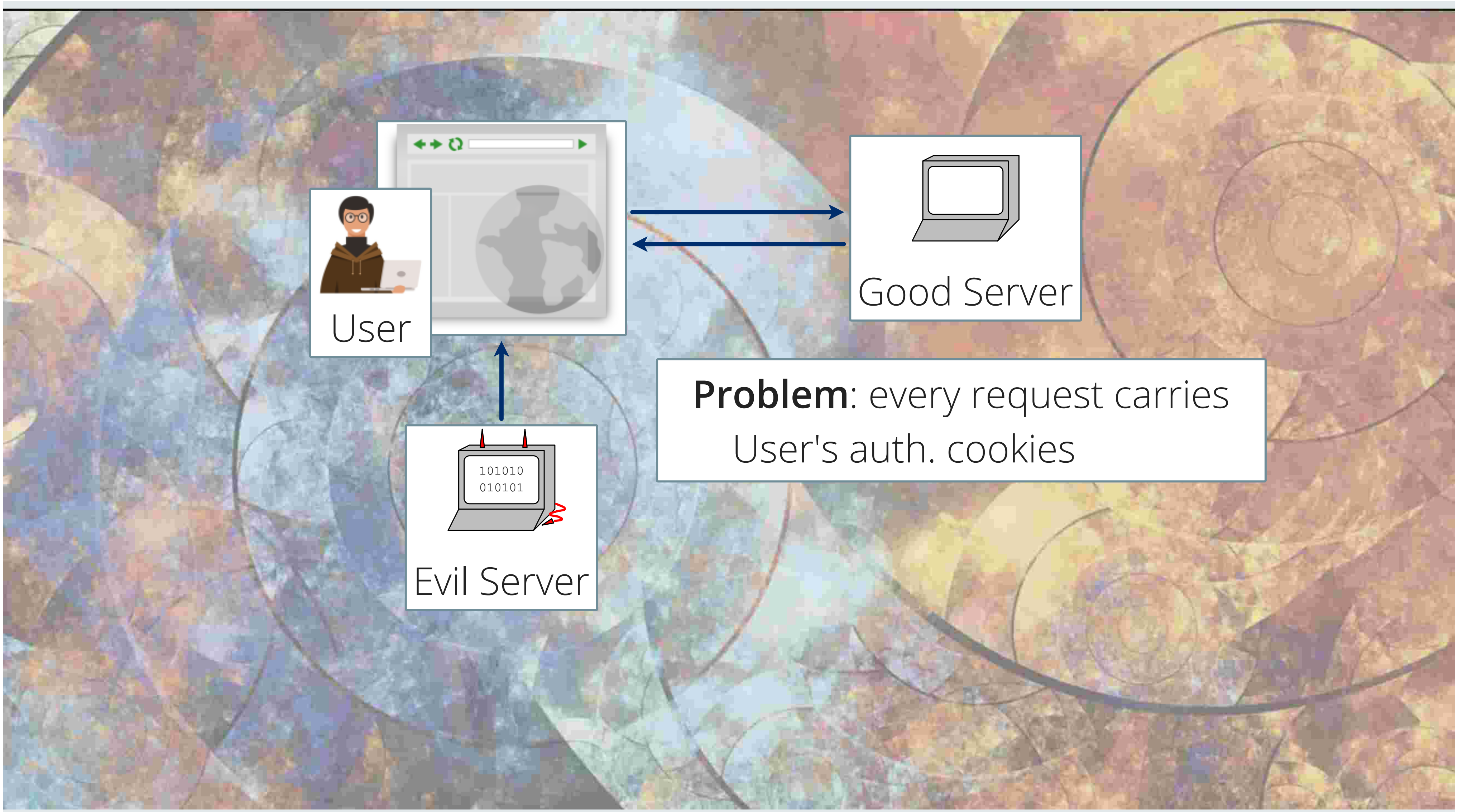
BROWN
THE UNIVERSITY OF UTAH


Datatypes
Relations
FORGE

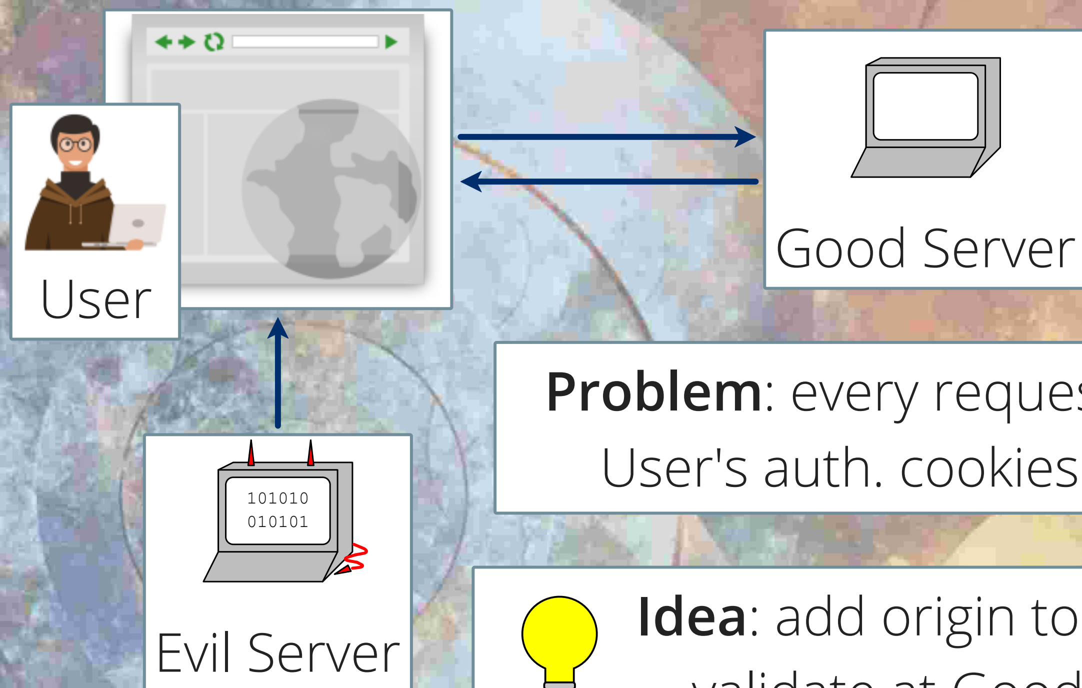

Datatypes
Relations
FORGE
Person4
loves
loves
Exploration +
Bounded Verification
loves
loves
Person6

Example:
Cross-Site Request Forgery


User
Good Server
101010
010101
Evil Server



User
Good Server
Evil Server
101010
010101
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

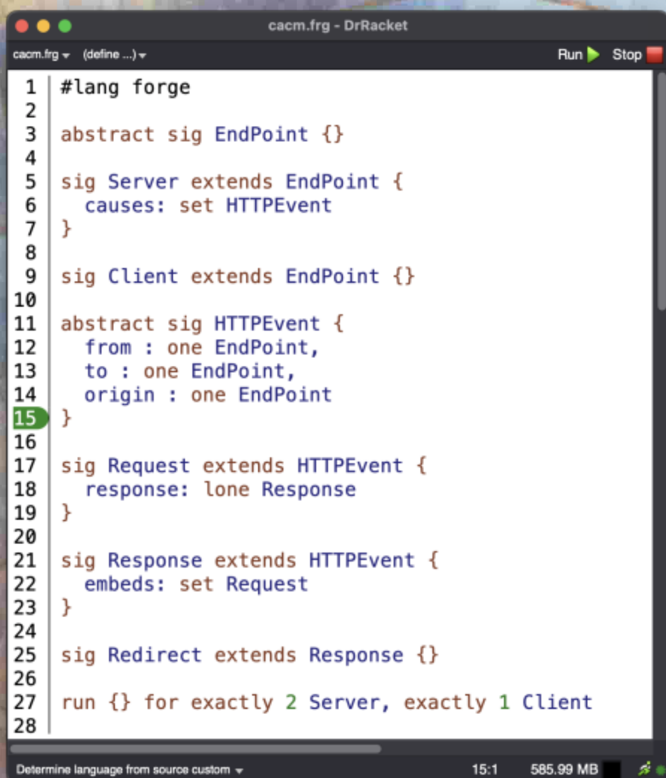
Datatypes

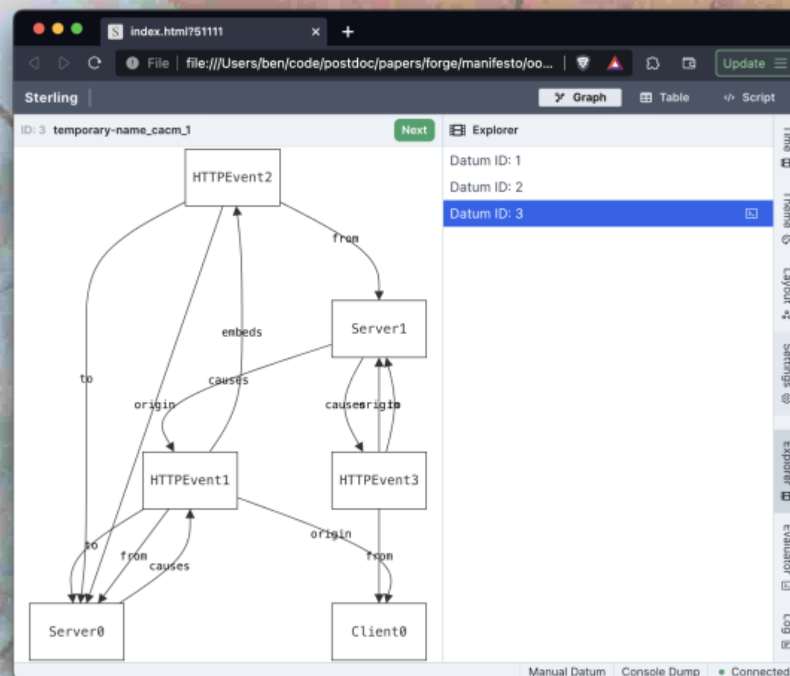
```
abstract sig EndPoint {}

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





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

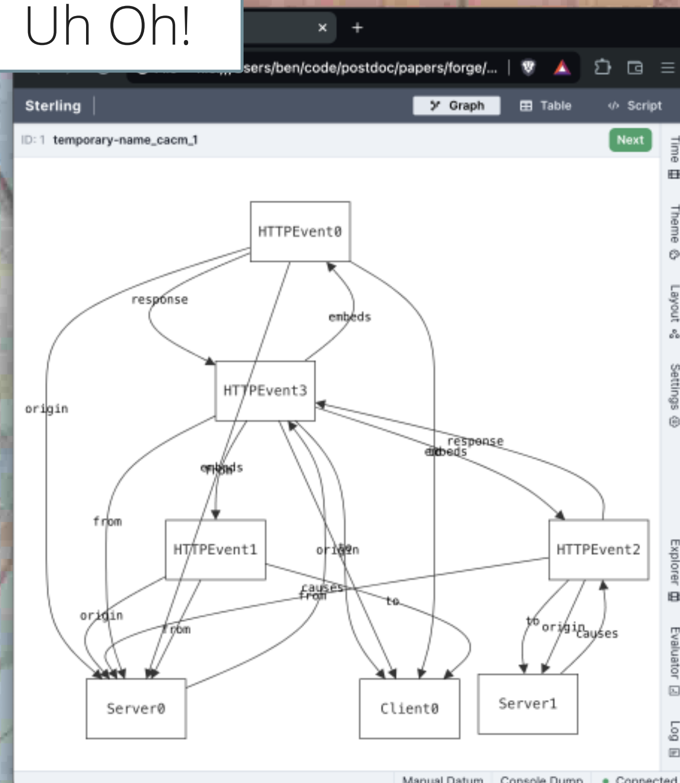
```
run {
  // can we find (hope not)
  some good, bad: Server {
    EnforceOrigins[good]
    // ...
  }
} for exactly 2 Server,
      exactly 1 Client,
      5 HTTPEvent
```

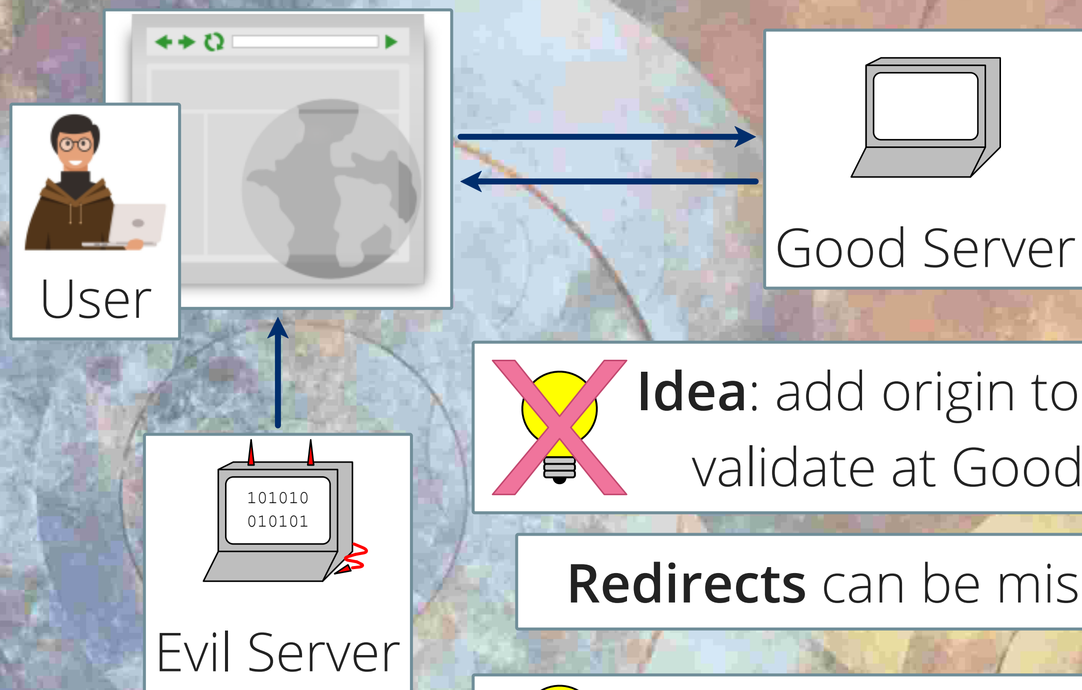
bounds

Checks

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

Uh Oh!



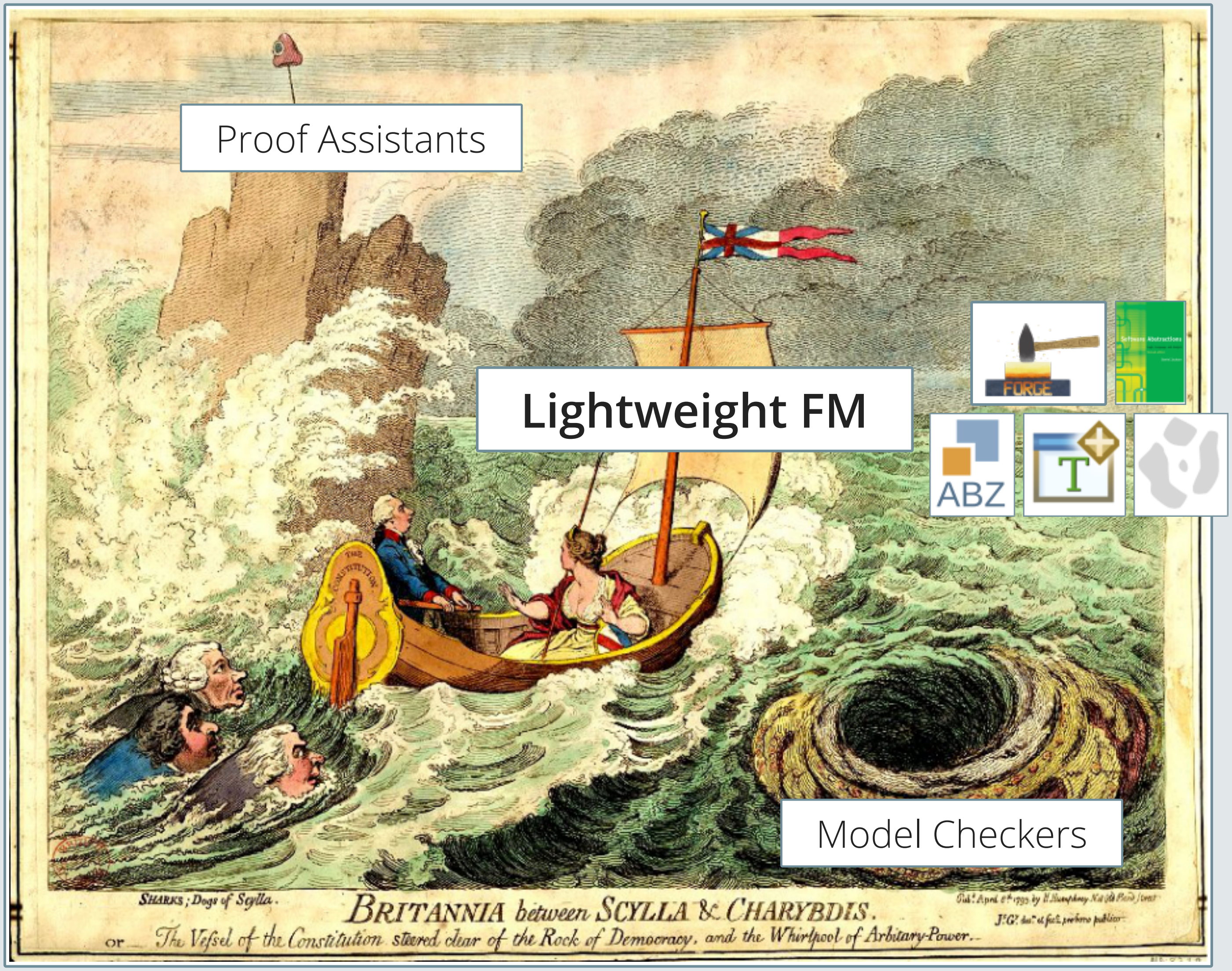


Idea: add origin to requests, validate at Good Server

Redirects can be mis-labeled

How about a set of origins??

Quickly found a bug!


Proof Assistants
Lightweight FM
Model Checkers
FORGE
ABZ
T
SHARKS; Dogs of Scylla.
BRITANNIA between SCYLLA & CHARYBDIS.
or — The Vessel of the Constitution steered clear of the Rock of Democracy, and the Whirlpool of Arbitary-Power.

Lightweight FM

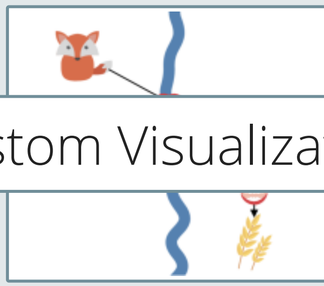
Usability >> Completeness

Insight: Most bugs have **small** instances

small scope hypothesis - D. Jackson

What sets Forge apart?

What sets Forge apart?

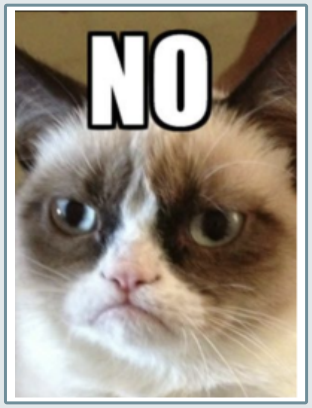


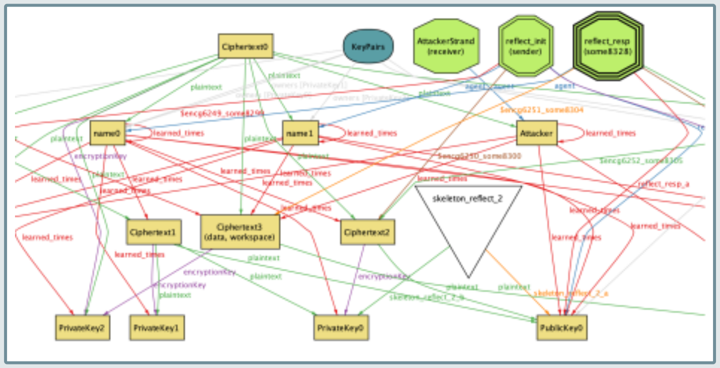
Custom Visualization

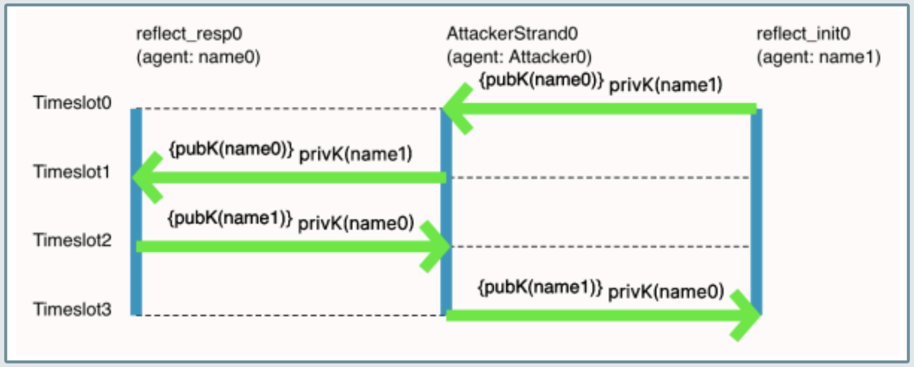
Unit Testing

Language Levels

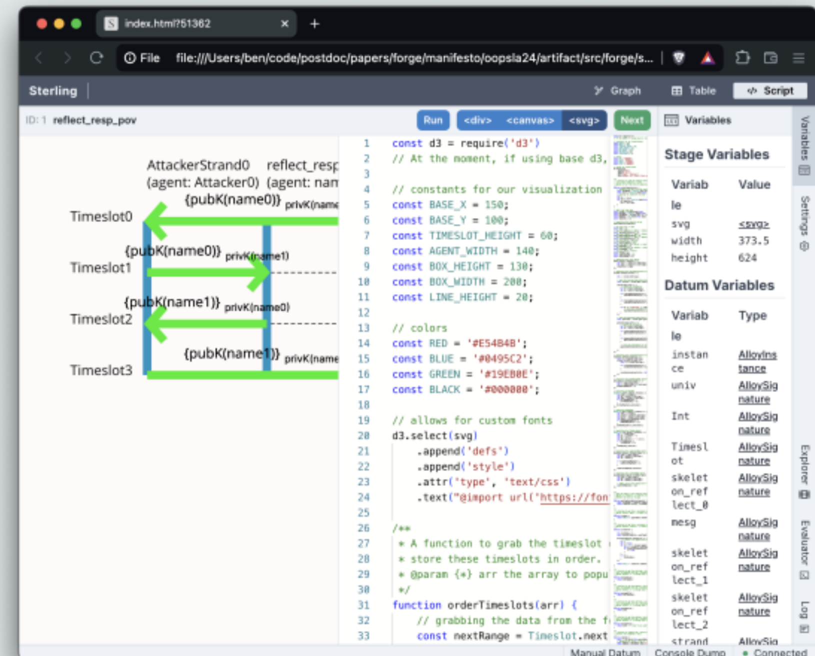
Custom Visualization



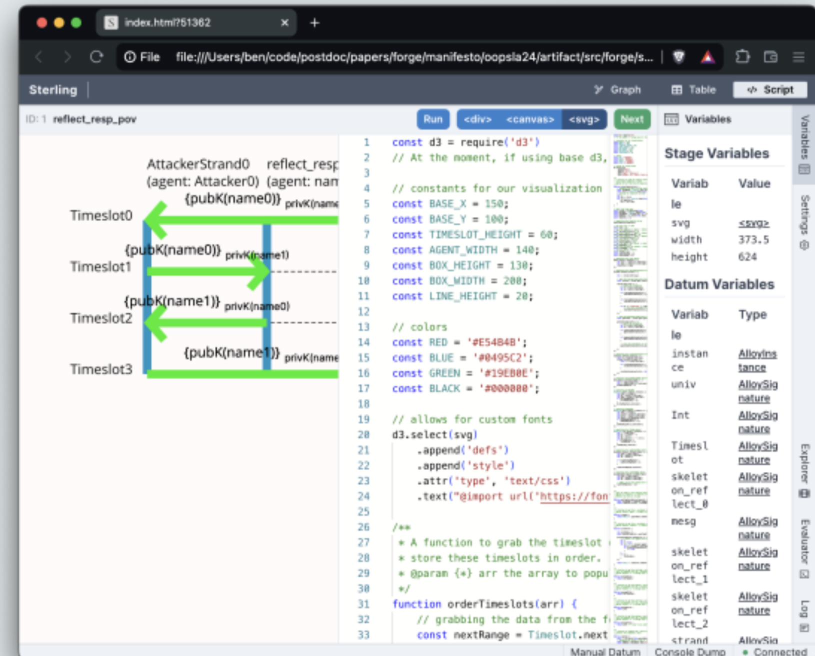




Custom Visualization

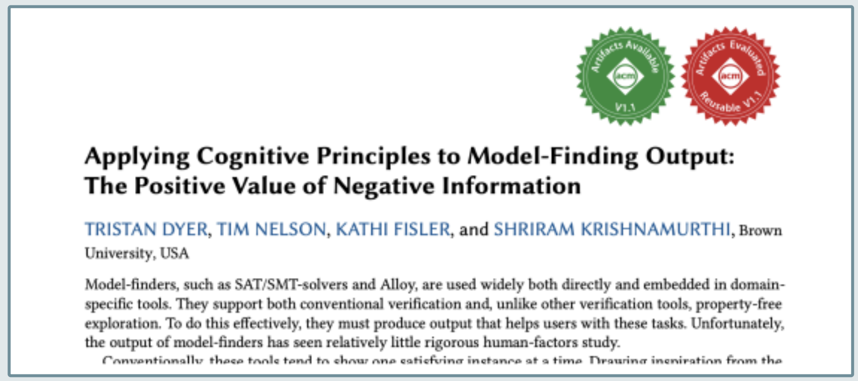


Custom Visualization



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

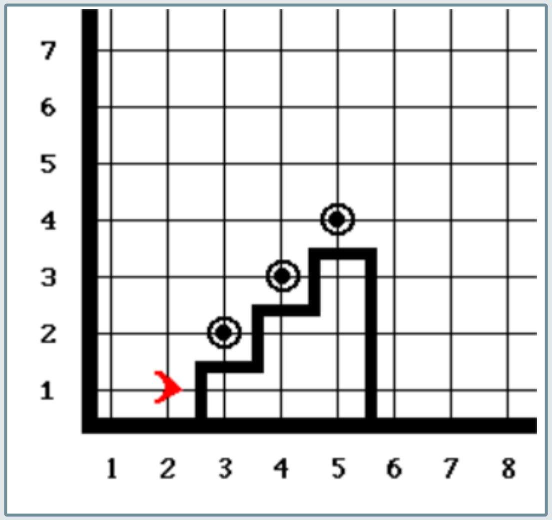
Model-finders, such as SAT/SMT-solvers and Alloy, are used widely both directly and embedded in domain-specific tools. They support both conventional verification and, unlike other verification tools, property-free exploration. To do this effectively, they must produce output that helps users with these 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

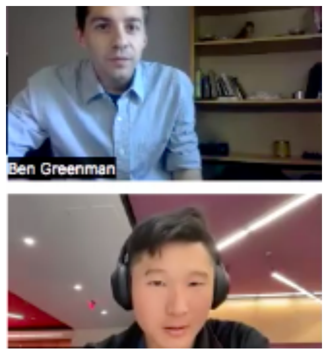
```
#lang forge/temporal
```
++ Linear Temporal Logic

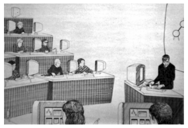
```
#lang forge/relational
```
++ N-ary Relations

```
#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

✗ **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))`

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

Q. Translate to LTL:

The green light turns on exactly once

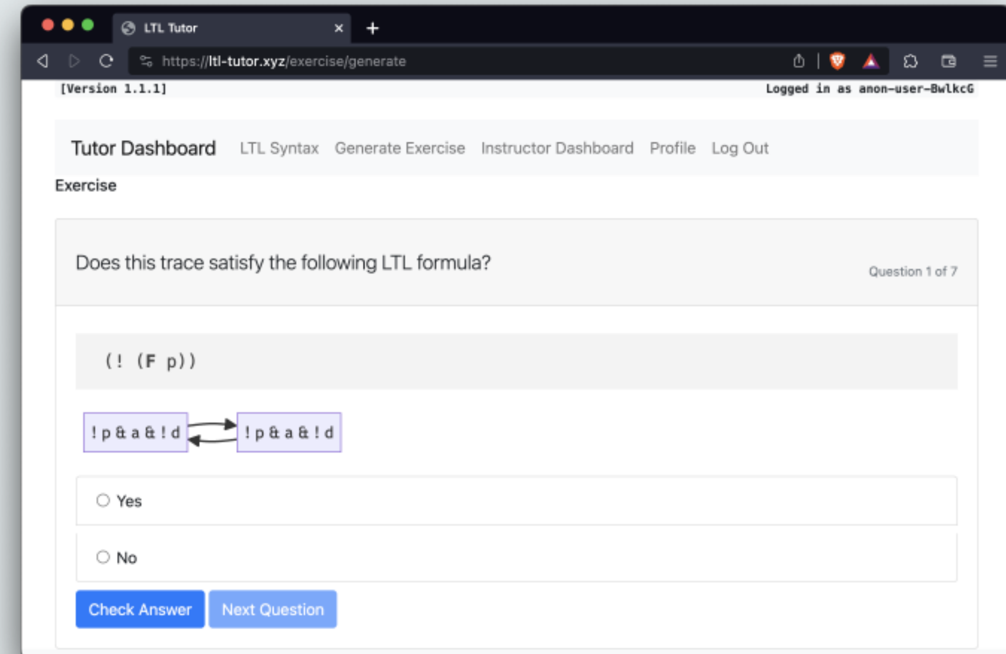
F = eventually **G** = always **X** = next state

✗ `F(green) & G(green => X(!green))`

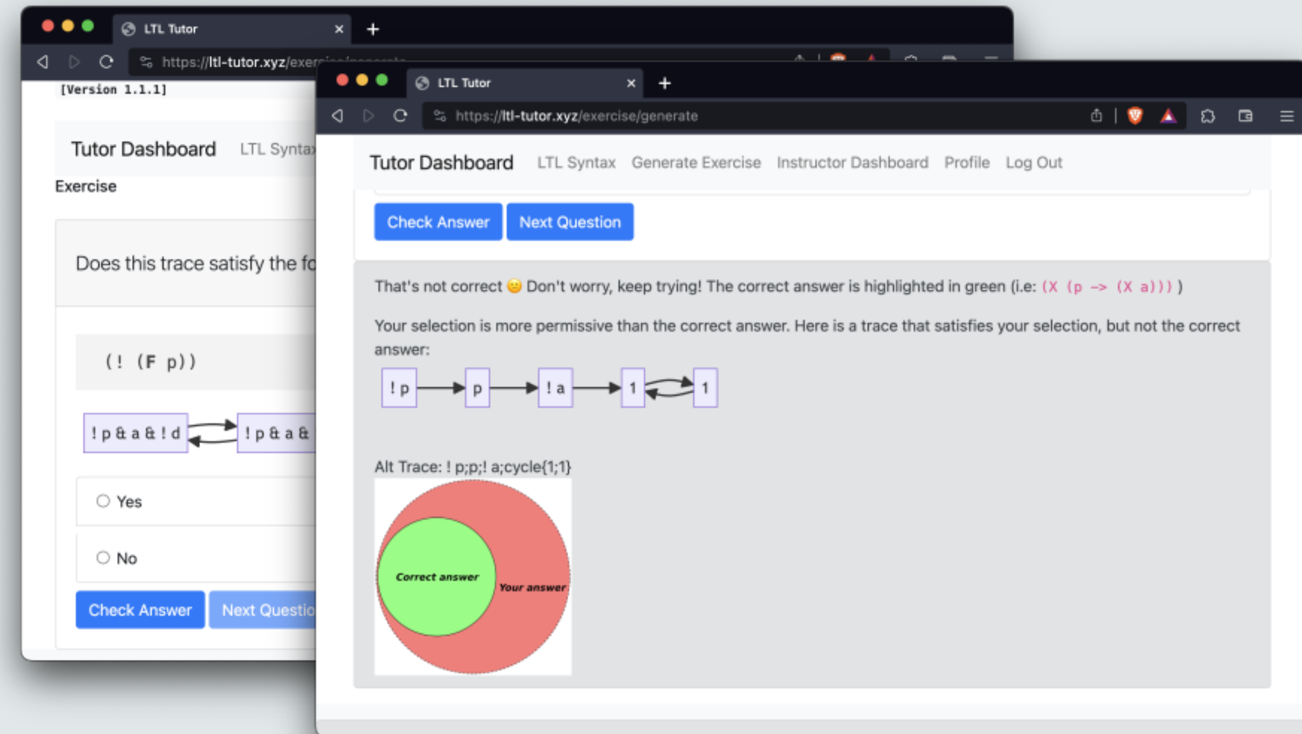
Implicit G

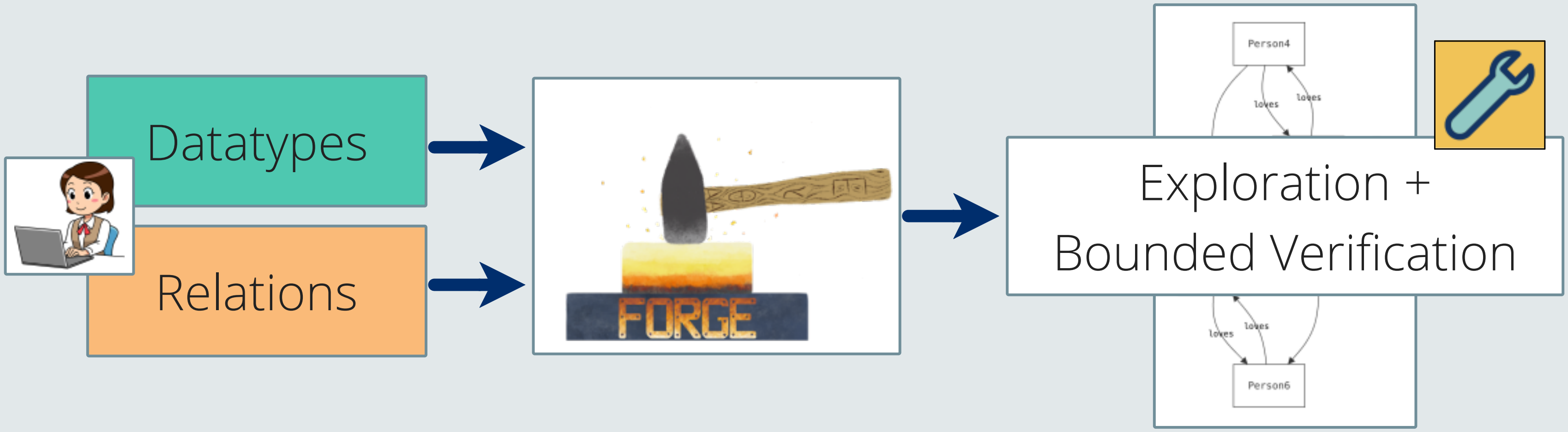
✓ `F(green) & G(green => X(G(!green)))`

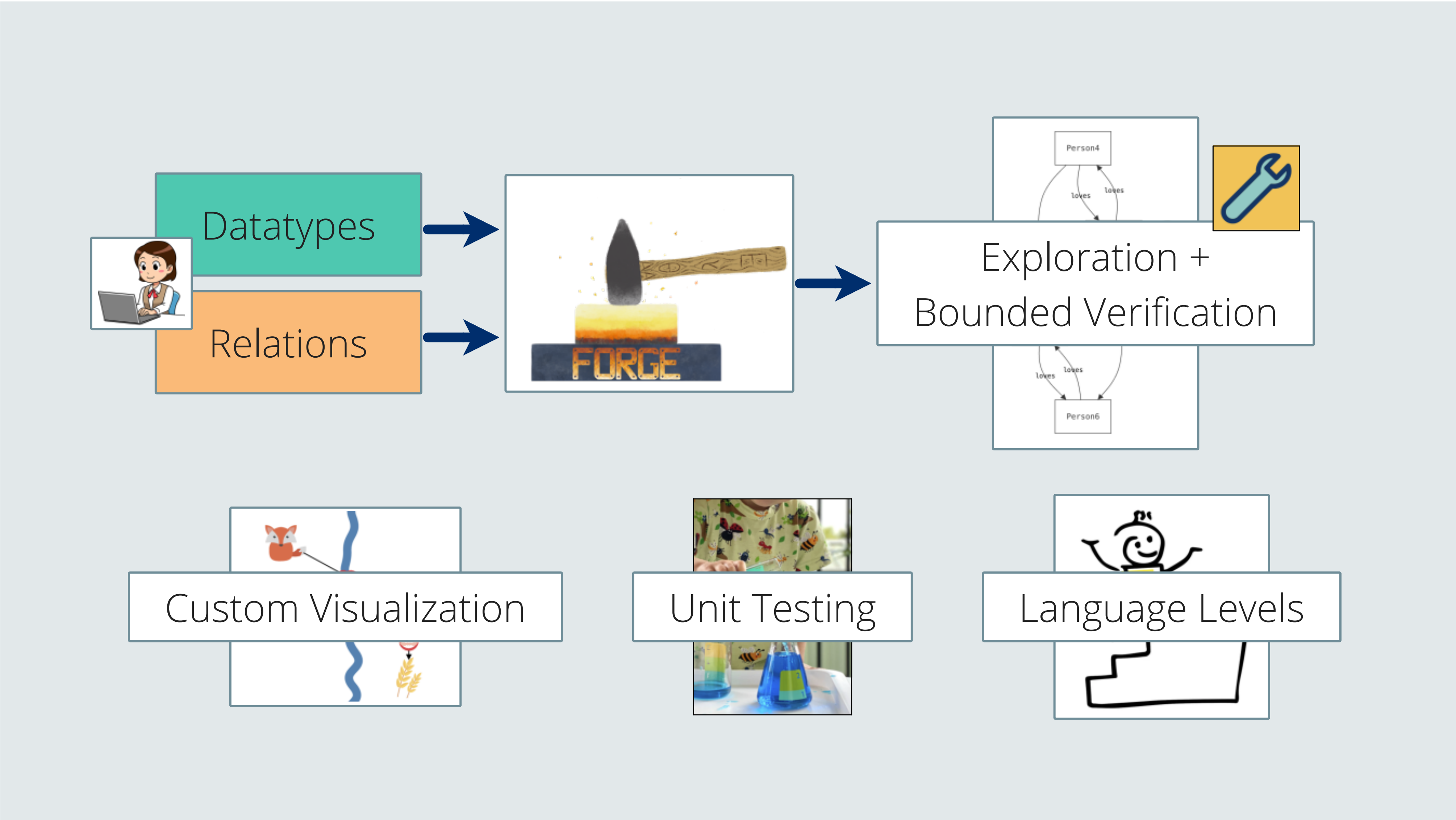
https://ltl-tutor.xyz



https://ltl-tutor.xyz




Datatypes
Relations
FORGE
Person4
loves
loves
Exploration +
Bounded Verification
loves
loves
Person6


Datatypes
Relations
FORGE
Person4
loves
loves
Exploration +
Bounded Verification
loves
loves
Person6
Custom Visualization
Unit Testing
Language Levels

https://forge-fm.org

blg@cs.utah.edu