Automating Formal Proofs for Reactive Systems

Daniel Ricketts, Valentin Robert, Dongseok Jang, Sorin Lerner

University of California, San Diego

Zachary “Danger” Tatlock

University of Washington
Proof Assistant Based Verification
Proof Assistant Based Verification

Testing tool for C compilers  [Yang et al. PLDI 11]
Proof Assistant Based Verification

Testing tool for C compilers  [Yang et al. PLDI 11]

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Bugs Found</th>
</tr>
</thead>
</table>


Proof Assistant Based Verification

Testing tool for C compilers [Yang et al. PLDI 11]

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Bugs Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>122</td>
</tr>
</tbody>
</table>
Proof Assistant Based Verification

Testing tool for C compilers

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Bugs Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>122</td>
</tr>
<tr>
<td>Clang/LLVM</td>
<td>181</td>
</tr>
</tbody>
</table>
Proof Assistant Based Verification

Testing tool for C compilers [Yang et al. PLDI 11]

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Bugs Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>122</td>
</tr>
<tr>
<td>Clang/LLVM</td>
<td>181</td>
</tr>
<tr>
<td>CompCert</td>
<td>?</td>
</tr>
</tbody>
</table>
Proof Assistant Based Verification

Testing tool for C compilers [Yang et al. PLDI 11]

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Bugs Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>122</td>
</tr>
<tr>
<td>Clang/LLVM</td>
<td>181</td>
</tr>
<tr>
<td>CompCert</td>
<td>0</td>
</tr>
</tbody>
</table>
Proof Assistant Based Verification

Testing tool for C compilers

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Bugs Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>122</td>
</tr>
<tr>
<td>Clang/LLVM</td>
<td>181</td>
</tr>
<tr>
<td>CompCert</td>
<td>0</td>
</tr>
</tbody>
</table>

[Yang et al. PLDI 11]
[Le et al. PLDI 14]
Proof Assistant Based Verification
Proof Assistant Based Verification

Proof Assistant
Proof Assistant Based Verification

Coq Theorem Prover

Proof Assistant
Proof Assistant Based Verification

- Code
- Proof Assistant
Proof Assistant Based Verification

Code

Proof Assistant

in language supporting reasoning
Proof Assistant Based Verification

- Code
- Spec

Proof Assistant
Proof Assistant Based Verification

- Code
- Spec
- Proof Assistant

logical properties characterizing correctness
Proof Assistant Based Verification

- Code
- Spec
- Proof Assistant
- Grads
Proof Assistant Based Verification

- Code
- Spec

Proof Assistant interacts with Code and Spec interactively to show that code satisfies the specification.
Proof Assistant Based Verification

Code

Spec

Proof Assistant

ML

x86

Grads
Proof Assistant Based Verification

- Code
- Spec
- Proof Assistant
  - Grads
  - ML
- Compile down to machine code: x86
Proof Assistant Based Verification

- Code
- Spec
- Proof Assistant
- ML
- x86

Extremely strong guarantees about actual system!
Proof Assistant Based Verification

Verified Compiler: CompCert

[Leroy POPL 06]
Proof Assistant Based Verification

Verified Compiler: CompCert
[Leroy POPL 06]

Verified OS micro-kernel: seL4
[Klein et al. SOSP 09]
Proof Assistant Based Verification

Verified Compiler: **CompCert**  
[Leroy POPL 06]

Verified OS micro-kernel: **seL4**  
[Klein et al. SOSP 09]

Verified Web browser: **Quark**  
[Jang et al. Security 12]
Manual Proof Burden

Spec

Code

Proof
Verified OS micro-kernel: seL4
Verified OS micro-kernel: seL4

9,000 lines of C code
Verified OS micro-kernel: seL4
9,000 lines of C code
20 person-years for verification
Manual Proof Burden

Spec
Code
Proof
Manual Proof Burden

Requires expertise in proof assistants

Diagram:
- Spec
- Code
- Proof

Note:

- Requires expertise in proof assistants.
Manual Proof Burden

- Requires expertise in proof assistants
- Extremely brittle, maintenance burden

Diagram:
- Spec
- Code
- Proof
Manual Proof Burden

Spec
Code
Proof
Manual Proof Burden

Spec

Code

Ideal
(no manual proofs)
Single application domain
Single application domain
Single application domain

spec1

code1

proof1

spec2

code2

proof2
Single application domain
Single application domain

Similar properties
Single application domain

Similar properties
Similar architecture
Single application domain

Similar properties

Similar architecture

Similar reasoning
Single application domain

DSL for Specs

Spec_1  
Code_1  
Proof_1

Spec_2  
Code_2  
Proof_2

Spec_3  
Code_3  
Proof_3

Similar properties
Single application domain

- DSL for Specs
- Similar architecture
- Similar reasoning

- Spec₁
- Code₁
- Proof₁

- Spec₂
- Code₂
- Proof₂

- Spec₃
- Code₃
- Proof₃
Single application domain

- DSL for Specs
- DSL for Code

Spec$_1$, Code$_1$, Proof$_1$
Spec$_2$, Code$_2$, Proof$_2$
Spec$_3$, Code$_3$, Proof$_3$

Similar properties
Similar architecture
DSL for Specs
DSL for Code
Single application domain

- DSL for Specs
- DSL for Code

Similar reasoning

Spec₁, Code₁, Proof₁
Spec₂, Code₂, Proof₂
Spec₃, Code₃, Proof₃
Single application domain

- **DSL for Specs**
- **DSL for Code**
- **Proof Automation**

1. Spec$_1$, Code$_1$, Proof$_1$
2. Spec$_2$, Code$_2$, Proof$_2$

Similar properties
Similar architecture
Similar reasoning

**DSL for Specs**
**DSL for Code**
**Proof Automation**
Single application domain

- DSL for Specs
- DSL for Code

Spec\(_1\)

Code\(_1\)

Spec\(_2\)

Code\(_2\)

Spec\(_3\)

Code\(_3\)

Proof Automation
Single application domain

- DSL for Specs
- DSL for Code

Spec_1 → Code_1 → Proof Automation
Spec_2 → Code_2 → Proof Automation
Spec_3 → Code_3 → Proof Automation
Single application domain

DSL for Specs
DSL for Code

Spec_1
Code_1

Proof_1

Spec_2
Code_2

Proof_2

Spec_3
Code_3

Proof_3
Reactive systems

DSL for Specs

DSL for Code

Spec_1

Code_1

Spec_2

Code_2

Spec_3

Code_3

Proof Automation

Proof_1

Proof_2

Proof_3
Reactive systems

DSL for Specs
DSL for Code

Spec_1
Code_1

Spec_2
Code_2

Spec_3
Code_3

Proof Automation

Proof_1
Proof_2
Proof_3
Reactive systems

REFLEX
REFLEX

No manual proofs,
REFLEX

No manual proofs, yet proof assistant guarantee.
REFLEX

No manual proofs, yet proof assistant guarantee.

*Automation incomplete,*
REFLEX

No manual proofs, yet proof assistant guarantee.

Automation incomplete, but verified browser, ssh, web server.
Reactive systems
Reactive systems

Continuously read messages from components and send messages to components.
Reactive systems
Example: Web browser kernel
Example: Web browser kernel

Kernel

Tab₁
  google

Tab₂
  gmail

Tab₃
  facebook
Example: Web browser kernel

---

Kernel

Cookie Manager
- google

Cookie Manager
- facebook

Tab_1
- google

Tab_2
- gmail

Tab_3
- facebook
Example: Web browser kernel

Kernel

Cookie Manager
googlet

Cookie Manager
facebook

Tab₁
googlet

Tab₂
gmailt

Tab₃
facebook
Example: Web browser kernel

Kernel

Cookie Manager
  google

Cookie Manager
  facebook

Tab
  Tab\textsubscript{1}
  google

Tab
  Tab\textsubscript{2}
  gmail

Tab
  Tab\textsubscript{3}
  facebook

(1) Tabs request resources (e.g. cookies)
(2) Kernel grants access subject to access controls (e.g. domain checks)

(1) Tabs request resources (e.g. cookies)
Example: Web browser kernel
Example: Web browser kernel

Components = Tab | CookieMgr | ...

Types of components
Example: Web browser kernel

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Types of messages
Example: Web browser kernel

Components = Tab | CookieMgr | ...  
Messages = CookieSet | CookieGet | ...  

Handlers:

How the kernel responds to messages from components
Example: Web browser kernel

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
  When [Tab t] sends CookieSet(c):

When tab t sends the kernel a CookieSet message with payload c
Example: Web browser kernel

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
  When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)

Get existing cookie manager with domain of t or spawn a new one
Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
  When [Tab t] sends CookieSet(c):
      cp <- find CookieMgr(t.domain)
      send(cp, CookieSet(c))

Send the cookie c to the found cookie manager
Example: Web browser kernel

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
  When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

  When [Tab t] sends CookieGet(c):
    ...

More handlers
Properties

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
  When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

  When [Tab t] sends CookieGet(c):
    ...

Properties

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
  When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

  When [Tab t] sends CookieGet(c):
    ...

Specify allowed behaviors
Specify allowed behaviors wrt sequence of system calls

Components = Tab | CookieMgr | ...  
Messages = CookieSet | CookieGet | ...  

Handlers:
  When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

  When [Tab t] sends CookieGet(c):
    ...

Properties
Properties

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:

When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

When [Tab t] sends CookieGet(c):
    ...

Specify allowed behaviors wrt sequence of system calls
Properties

When [Tab t] sends CookieSet(c):
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls
Properties

When [Tab t] sends CookieSet(c):
   cp <- find CookieMgr(t.domain)
   send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls
Properties

When [Tab t] sends CookieSet(c):
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls
Properties

When [Tab t] sends CookieSet(c):
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls

The system calls so far

...
Properties

When [Tab t] sends CookieSet(c):

```
cp <- find CookieMgr(t.domain)
send(cp, CookieSet(c))
```

Specify allowed behaviors wrt sequence of system calls

The system calls so far
Properties

When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls

The system calls so far
Properties

When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls

The system calls so far

Recv(Tab, CookieSet(c))

...
Properties

When [Tab t] sends CookieSet(c):
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls

The system calls so far

Recv(Tab, CookieSet(c))

...
Properties

When [Tab t] sends CookieSet(c):
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls

- Recv(Tab, CookieSet(c))
- ...

The system calls so far

Time
Properties

When `[Tab t]` sends `CookieSet(c)`:  

```latex
cp <- find CookieMgr(t.domain)
send(cp, CookieSet(c))
```

Specify allowed behaviors wrt sequence of system calls

<table>
<thead>
<tr>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The system calls so far

- `Spawn CookieMgr(t.domain)`
- `Recv(Tab, CookieSet(c))`
- `...`
Properties

When [Tab t] sends CookieSet(c):
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls
Properties

When [Tab t] sends CookieSet(c):
   cp <- find CookieMgr(t.domain)
   send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls

The system calls so far

Spawn CookieMgr(t.domain)
Recv(Tab, CookieSet(c))
...

Time
Properties

When [Tab t] sends CookieSet(c):
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))

Specify allowed behaviors wrt sequence of system calls

Send(cp, CookieSet(c))

Spawn CookieMgr(t.domain)

Recv(Tab, CookieSet(c))

The system calls so far

...
### Example: Web browser kernel

<table>
<thead>
<tr>
<th>Components</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tab</code></td>
<td><code>CookieMgr</code></td>
</tr>
</tbody>
</table>

**Handlers:**

- **When `[Tab t]` sends `CookieSet(c)`:**
  
  ```
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))
  ```

- **When `[Tab t]` sends `CookieGet(c)`:**
  
  ```
  ...
  ```
Example: Web browser kernel

Components = Tab | CookieMgr | ...  
Messages = CookieSet | CookieGet | ...  

Handlers:  
  When [Tab t] sends CookieSet(c):  
    cp <- find CookieMgr(t.domain)  
    send(cp, CookieSet(c))  

  When [Tab t] sends CookieGet(c):  
    ...
Example: Web browser kernel

```
Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
  When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

  When [Tab t] sends CookieGet(c):
    ...
```

Specify cookie integrity

Example: Web browser kernel

```
Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handers:
  When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

  When [Tab t] sends CookieGet(c):
    ...
```

Specify cookie integrity
Example: Web browser kernel

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
  When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

  When [Tab t] sends CookieGet(c):
    ...

Example: Web browser kernel
Example: Web browser kernel

forall d c,

For any domain \( d \) and cookie \( c \)

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:

When [Tab t] sends CookieSet(c):
   cp <- find CookieMgr(t.domain)
   send(cp, CookieSet(c))

When [Tab t] sends CookieGet(c):
   ...

Example: Web browser kernel

forall d c,
Example: Web browser kernel

forall d c,

\[
\text{[Send(CookieMgr(d), CookieSet(c))]}\]

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
  When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

  When [Tab t] sends CookieGet(c):
    ...

The kernel sends the cookie manager for domain d a cookie c
Example: Web browser kernel

forall d c,

Enables
[Send(CookieMgr(d), CookieSet(c))]

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:

When [Tab t] sends CookieSet(c):
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))

When [Tab t] sends CookieGet(c):
  ...

Only if
Example: Web browser kernel

forall d c,
    [Recv(Tab(d), CookieSet(c))] Enables
    [Send(CookieMgr(d), CookieSet(c))]

The kernel already received a cookie c from a tab of domain d
Example: Web browser kernel

forall d c,
    [Recv(Tab(d), CookieSet(c))] Enables
    [Send(CookieMgr(d), CookieSet(c))]

A Enables B iff every sys call B is preceded by sys call A

when [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

When [Tab t] sends CookieGet(c):
    ...

Example: Web browser kernel

for all d, c,
    [Recv(Tab(d), CookieSet(c))]
Enables
    [Send(CookieMgr(d), CookieSet(c))]

Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
    When [Tab t] sends CookieSet(c):
        cp <- find CookieMgr(t.domain)
        send(cp, CookieSet(c))

    When [Tab t] sends CookieGet(c):
        ...

Example: Web browser kernel

```
for d c,
  [Recv(Tab(d), CookieSet(c))]
  Enables
  [Send(CookieMgr(d), CookieSet(c))]
```
Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))

When [Tab t] sends CookieGet(c):
...

Example: Web browser kernel
forall d c,
[Recv(Tab(d), CookieSet(c))]
Enables [Send(CookieMgr(d), CookieSet(c))]

REFLEX Benefits:
Components = Tab | CookieMgr | ...

Messages = CookieSet | CookieGet | ...

Handlers:

When [Tab t] sends CookieSet(c):

\[ cp <- \text{find CookieMgr(t.domain)} \]
\[ \text{send}(cp, \text{CookieSet}(c)) \]

When [Tab t] sends CookieGet(c):

\[ \ldots \]

Example: Web browser kernel

forall d c,
\[ \text{[Recv(Tab(d), CookieSet(c))]} \]
\[ \text{[Send(CookieMgr(d), CookieSet(c))]} \]

REFLEX Benefits:

Proofs fully automated
Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
When [Tab t] sends CookieSet(c):
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))
When [Tab t] sends CookieGet(c):
...

Example: Web browser kernel
forall d c,
[Recv(Tab(d), CookieSet(c))]
Enables
[Send(CookieMgr(d), CookieSet(c))]

REFLEX Benefits:
Proofs fully automated
No lemmas
Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:
When [Tab t] sends CookieSet(c):
    cp <- find CookieMgr(t.domain)
    send(cp, CookieSet(c))
When [Tab t] sends CookieGet(c):
...

Example: Web browser kernel
forall d c,
[Recv(Tab(d), CookieSet(c))]
Enables
[Send(CookieMgr(d), CookieSet(c))]

REFLEX Benefits:

Proofs fully automated
No lemmas
No invariants
Components = Tab | CookieMgr | ...
Messages = CookieSet | CookieGet | ...

Handlers:

When [Tab t] sends CookieSet(c):
  cp <- find CookieMgr(t.domain)
  send(cp, CookieSet(c))

When [Tab t] sends CookieGet(c):

Example: Web browser kernel

forall d c,
[Recv(Tab(d), CookieSet(c))] Enables [Send(CookieMgr(d), CookieSet(c))]

REFLEX Benefits:

Proofs fully automated
No lemmas
No invariants
No manual proofs
Proof Automation
Proof Automation

Prove kernel code satisfies properties
Proof Automation

Prove kernel code satisfies properties

by induction on sys call sequences kernel can produce
Proof Automation

Prove kernel code satisfies properties

by induction on sys call sequences kernel can produce
Proof Automation

Prove kernel code satisfies properties

*by induction on sys call sequences kernel can produce*
Proof Automation

Prove kernel code satisfies properties

*by induction on sys call sequences kernel can produce*
Proof Automation

Prove kernel code satisfies properties

by induction on sys call sequences kernel can produce
Proof Automation

Prove kernel code satisfies properties
by induction on sys call sequences kernel can produce

Handlers create structure for induction
Proof Automation

Prove kernel code satisfies properties

*by induction on sys call sequences kernel can produce*
Proof Automation

Prove kernel code satisfies properties

by induction on sys call sequences kernel can produce

Induction hypothesis: property holds up to this point
Proof Automation

Prove kernel code satisfies properties

by induction on sys call sequences kernel can produce

Run a single handler

Induction hypothesis: property holds up to this point
Proof Automation

Prove kernel code satisfies properties

\textit{by induction on sys call sequences kernel can produce}

Proof obligation: does property still hold?

Induction hypothesis: property holds up to this point
Proof Automation

Prove kernel code satisfies properties

by induction on sys call sequences kernel can produce

Proof obligation: does property still hold?

Induction hypothesis: property holds up to this point

Case analysis on handler
Proof Automation

Prove kernel code satisfies properties

by induction on sys call sequences kernel can produce

Proof obligation: does property still hold?

Induction hypothesis: property holds up to this point

Case analysis on handler
Symbolically run all paths
Proof Automation

Prove kernel code satisfies properties

*by induction on sys call sequences kernel can produce*

- Proof obligation: does property still hold?
- Induction hypothesis: property holds up to this point
- Case analysis on handler
  - Symbolically run all paths
  - **Prove automatically**
Proof Automation

Single domain insights:
Proof Automation

Single domain insights:

Small number of carefully designed property primitives
Proof Automation

Single domain insights:

Small number of carefully designed property primitives

Loop free handlers allowed symbolic eval of all paths
Proof Automation

**Single domain insights:**

- Small number of carefully designed property primitives
- Loop free handlers allowed symbolic eval of all paths
- Domain-specific heuristics for non-local reasoning
Evaluation
Evaluation

- Web browser
- SSH server
- Web server
Evaluation

Web browser

SSH server

Web server

Auto verified 33 properties (80% in < 2 minutes)
## Evaluation

<table>
<thead>
<tr>
<th>Web browser</th>
<th>Domains do not interfere, Cookie integrity, …</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH server</td>
<td>No PTY access before authentication, At most 3 authentication attempts, …</td>
</tr>
<tr>
<td>Web server</td>
<td>Clients only spawned after successful login, File requests guarded by access control, …</td>
</tr>
</tbody>
</table>

Auto verified 33 properties (80% in < 2 minutes)
## Evaluation

<table>
<thead>
<tr>
<th>Web browser</th>
<th>SSH server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domains do not interfere, Cookie integrity, ...</td>
<td>No PTY access before authentication, at most 3 authentication attempts, ...</td>
</tr>
<tr>
<td><strong>Able to automate proofs of non-interference</strong></td>
<td>Clients only spawned after successful login, File requests guarded by access control, ...</td>
</tr>
</tbody>
</table>

Auto verified 33 properties (80% in < 2 minutes)
<table>
<thead>
<tr>
<th></th>
<th>Domains do not interfere, Cookie integrity, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Web browser</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SSH server</strong></td>
<td>No PTY access before authentication, At most 3 authentication attempts, ...</td>
</tr>
<tr>
<td><strong>Web server</strong></td>
<td>Clients only spawned after successful login, File requests guarded by access control, ...</td>
</tr>
</tbody>
</table>

Auto verified 33 properties (80% in < 2 minutes)
**Evaluation**

<table>
<thead>
<tr>
<th>Web browser</th>
<th>Domains do not interfere, Cookie integrity, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH server</td>
<td>No PTY access before authentication, At most 3 authentication attempts, ...</td>
</tr>
<tr>
<td>Web server</td>
<td>Clients only spawned after successful login, File requests guarded by access control, ...</td>
</tr>
</tbody>
</table>

Able to automate proofs of non-local properties

Auto verified 33 properties (80% in < 2 minutes)
Development Effort: Framework
Development Effort: Framework

**Reflex:**

7500 lines of Coq
Development Effort: Framework

**Reflex:**

7500 lines of Coq

| Web browser | SSH server | Web server |
# Development Effort: Framework

## Reflex:

- 7500 lines of Coq

<table>
<thead>
<tr>
<th>Web browser</th>
<th>SSH server</th>
<th>Web server</th>
</tr>
</thead>
</table>

## Quark Web browser:

- 5500 lines of Coq
Development Effort: Framework

Reflex:
7500 lines of Coq

<table>
<thead>
<tr>
<th>Web browser</th>
<th>SSH server</th>
<th>Web server</th>
</tr>
</thead>
</table>

Quark Web browser:
5500 lines of Coq

Single reactive system
Development Effort: Framework

Reflex:
- Many reactive systems
- 7500 lines of Coq

<table>
<thead>
<tr>
<th>Web browser</th>
<th>SSH server</th>
<th>Web server</th>
</tr>
</thead>
</table>

Quark Web browser:
- Single reactive system
- 5500 lines of Coq
Development Effort: Systems
## Development Effort: Systems

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language</th>
<th>Lines of Code/Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Browser</strong></td>
<td>REFLEX</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>C++, Python</td>
<td>970,240</td>
</tr>
<tr>
<td><strong>SSH server</strong></td>
<td>REFLEX</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>C, Python</td>
<td>89,567</td>
</tr>
<tr>
<td><strong>Web server</strong></td>
<td>REFLEX</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Python</td>
<td>386</td>
</tr>
<tr>
<td>Benchmark</td>
<td>Language</td>
<td>Lines of Code/Spec</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Browser</td>
<td>REFLEX</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>C++, Python</td>
<td>970,240</td>
</tr>
<tr>
<td>SSH server</td>
<td>REFLEX</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>C, Python</td>
<td>89,567</td>
</tr>
<tr>
<td>Web server</td>
<td>REFLEX</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Python</td>
<td>386</td>
</tr>
</tbody>
</table>
## Development Effort: Systems

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language</th>
<th>Lines of Code/Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser</td>
<td>REFLEX</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>C++, Python</td>
<td>970,240</td>
</tr>
<tr>
<td>SSH server</td>
<td>REFLEX</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>C, Python</td>
<td>89,567</td>
</tr>
<tr>
<td>Web server</td>
<td>REFLEX</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Python</td>
<td>386</td>
</tr>
</tbody>
</table>

- Webkit
- OpenSSH
Limitations
Limitations

Expressiveness:

Incompleteness:
Limitations

Expressiveness:

Strict subset of temporal logic + non-interference

Incompleteness:
Limitations

Expressiveness:

- Strict subset of temporal logic + non-interference
- Loop free handlers

Incompleteness:
Limitations

Expressiveness:
- Strict subset of temporal logic + non-interference
- Loop free handlers
- No user-defined unbounded data structures

Incompleteness:
Limitations

Expressiveness:

- Strict subset of temporal logic + non-interference
- Loop free handlers
- No user-defined unbounded data structures

Incompleteness:

- Unable to infer some inductive invariants
Limitations

Expressiveness:
- Strict subset of temporal logic + non-interference
- Loop free handlers
- No user-defined unbounded data structures

Incompleteness:
- Unable to infer some inductive invariants
- Low level incompleteness in automation tactics
SSH Server Kernel in REFLEX
SSH Server Kernel in REFLEX
Web Browser Kernel in REFLEX
Web Browser Kernel in REFLEX
Conclusion

Proof assistant based verification
Conclusion

Proof assistant based verification

Expressiveness

Automation

Coq
Conclusion

Proof assistant based verification

Automation

Expressiveness

Ynot

Coq
Conclusion

Proof assistant based verification

Automation

Expressiveness

Bedrock
Ynot
Coq
Conclusion

Proof assistant based verification

Automation

Expressiveness

Reflex

Bedrock

Ynot

Coq
Conclusion

Reflex
Conclusion

Reflex

DSL expressive enough for entire domain
Conclusion

Reflex

- DSL expressive enough for entire domain
- Automation eliminates manual proof burden
Conclusion

Reflex

- DSL expressive enough for entire domain
- Automation eliminates manual proof burden

http://goto.ucsd.edu/reflex/
Thank You!

Reflex

- DSL expressive enough for entire domain
- Automation eliminates manual proof burden
- http://goto.ucsd.edu/reflex/