Software Infrastructure
Software Infrastructure is Shaky

The New York Times
Cars’ Computer Systems Called at Risk to Hackers
By JOHN MARKOFF
Published: May 14, 2010

Automobiles, which will be increasingly connected to the Internet in the near future, could be vulnerable to hackers.

Now, two teams of computer scientists are alerting automotive manufacturers to the potential threat of hackers who may be able to remote-control critical functions, from disabling the brakes to stealing the car.

“We demonstrated that the automotive industry is unprepared for adversarial computer hacking attacks,” said Dr. Edward W. Felten, a computer science professor at Princeton University.

Dr. Felten and his colleague Dr. Robert Lee demonstrated this week in the paper, which will be presented at the USENIX Security Symposium, in San Francisco, that it is possible to remotely initiate the car’s engine, get inside the car and even remotely disable the brakes.

In the paper, which will be presented at the conference, the researchers presented an example of how a hacker could remotely disable the brakes.

For example, the researchers created a program that could remotely disable the brakes of a car.

Bloomberg Businessweek
Software Bug Made Swedish Exchange Go Bork, Bork, Bork
By Karen Weise on November 29, 2012

A computer error stalks the markets—again. An order on a relatively obscure derivatives index in Stockholm yesterday was asking to buy futures contracts on Swedish stocks valued at 131 times the country’s entire GDP. The order made the exchange go “bananas” and caused Nasdaq OMX to stop trading in Swedish derivatives for four hours.

This was no “fat finger” incident, where a trader accidentally types an extra few digits or the wrong numbers in an order. Instead, a software glitch magnified an order, Nasdaq OMX spokesman Carl Norell told Bloomberg News. “Our system misinterpreted a certain order category and communicated a value that was way too high into the book,” he said.

The interruption was in a small corner of the market, but it’s just the latest in a string of technical problems that have halted trading. As more trading is driven by the algorithms of high-frequency traders, one glitch or bad order can spark major disruptions. The 2010 flash crash caused $862 billion in stock values to vanish from the market temporarily, and technical problems still knock the market offline from time to time. 
Software Infrastructure is Shaky
Software Infrastructure is Shaky

prog error

⇒

patch

∞
When exhaustive testing is impossible, our trust can only be based on proof.

Edsger W. Dijkstra
Under the Spell of Leibniz's Dream

... not just a dream!

Social Processes and Proofs of Theorems and Programs
Richard A. De Millo
Georgia Institute of Technology
Richard J. Lipton and Alan J. Perlis
Yale University
Proof Assistant Based Verification

Code in language suited for reasoning

Develop correctness proof in synch

Fully formal, *machine checkable* proof
Proof Assistant Based Verification

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

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

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

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

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

[Yang et al. PLDI 11]  
[Vu et al. PLDI 14]

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

realistic implementation guaranteed bug free
Proof Assistant Based Verification

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

- **Compiler**
- **Bugs Found**

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

**Proof** $\Rightarrow$ **no prog errors**

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

realistic implementation guaranteed bug free
Promise

Proof

↓

no prog errors
To day

prog error

patch

Promise

Proof

no prog errors
Today

- prog error
- patch

Promise

- Proof
- no prog errors
Today

Proof

Burden

Promise

Proof

no prog errors
The Burden of Proof

1. Initial proofs require heroic effort

    CompCert: 70% proof, vast majority of effort
    seL4: 200,000 line proof for 9,000 lines of C

2. Code updates require re-proving

    CompCert: adding opts [Tristan POPL 08, PLDI 09, POPL 10]
    seL4: changing RPC took 17% of proof effort
Mitigating the Burden of Proof

1: Scaling proofs to critical infrastructure

   Formal shim verification for large apps

   QUARK: browser with security guarantees

2: Evolving formally verified systems

   Reflex DSL exploits domain for proof auto
Fully Formal Verification
Fully Formal Verification

Proof Assistant

Coq Theorem Prover
Fully Formal Verification

Code

in language suited to reasoning

Proof Assistant
Fully Formal Verification

Code

Spec

Proof Assistant

logical properties characterizing correctness
Fully Formal Verification

Code

Spec

Proof Assistant

Grad

interactively show code satisfies specification
Fully Formal Verification

- Code
- Spec
- Proof Assistant
- Grad
- ML
- x86

compile down to machine code
Fully Formal Verification

Proof Assistant

Code

Spec

ML

x86

Grad

Extremely strong guarantees about actual system!
Fully Formal Verification
Fully Formal Verification

program in a purely functional language

```latex
Fixpoint factorial n :=
  match n with
  | 0 => 1
  | S m => n * factorial m
end.
```
Fully Formal Verification

```ocaml
Fixpoint factorial n :=
    match n with
    | 0    => 1
    | S m  => n * factorial m
end.

Definition monotonic f :=
    forall a b,
    a <= b ->
    f a <= f b.
```

specification characterizes desired behavior
Fully Formal Verification

Fixpoint factorial n :=
  match n with
  | 0    => 1
  | S m  => n * factorial m
end.

Definition monotonic f :=
  forall a b, a <= b -> f a <= f b.

Theorem example :
  monotonic factorial.
Proof.
  ...
Fully Formal Verification

Fixpoint factorial n :=
  match n with
  | 0   => 1
  | S m => n * factorial m
end.

Definition monotonic f :=
  forall a b,
  a <= b ->
  f a <= f b.

Theorem example :
  monotonic factorial.
Proof.
  unfold monotonic. intros n1 n2 H.
  induction H. apply le_refl. simpl.
  apply le_trans with (m := factorial m); auto.
  destruct (mult_0_le (factorial m) m).
  rewrite H0; simpl. apply le_refl.
  apply le_trans with (m := m * factorial m); auto.
  rewrite plus_n_0 at 1. rewrite plus_comm.
  apply plus_le_compat. apply le_0_n. apply le_refl.
Qed.
browsers don’t look like factorial

browsers don’t have simple specs

even easy proofs grow quickly and become opaque
Fully Formal Verification

Scrap existing code, rewrite
Invest decades of person-years
Intractable for large-scale apps
Formally Verify a Browser?!
Formally Verify a Browser?!

Millions of LOC

Web Browser
Formally Verify a Browser?!

Millions of LOC

High performance
Formally Verify a Browser?!

- Millions of LOC
- High performance
- Loose access policy
Formally Verify a Browser?!

- Millions of LOC
- High performance
- Loose access policy
- Constant evolution

Resources

- JavaScript
- JPEG
- HTML
Formally Verify a Browser?!

Resources

Isolate
sandbox untrusted code

JavaScript
JPEG
HTML
Formally Verify a Browser?!

- **Resources**
  - **Shim**
    - Implement shim
      - guards resource access
    - Isolate
      - sandbox untrusted code
  - **JavaScript**
  - **JPEG**
  - **HTML**
Formally Verify a Browser?!

- Implement shim
  - guards resource access

- Verify shim
  - prove security policy

- Isolate
  - sandbox untrusted code
Formal Shim Verification

- Implement shim
guards resource access
- Verify shim
prove security policy

Resources

- Shim
  - Isolate
    sandbox untrusted code
  - Implement shim
    guards resource access
  - Verify shim
    prove security policy

- JavaScript
- JPEG
- HTML
Formal Shim Verification

- Implement shim
- Verify shim

Resources

Shim

Sandbox

Untrusted Code

Isolate

Implement shim

Verify shim

Applies when:
1. sys fits architecture
2. policy over resources
   browser, httpd, sshd, ...

✔

Untrusted Code

Formal Shim Verification

Applies when:
1. sys fits architecture
2. policy over resources
   browser, httpd, sshd, ...
Key Insight: Focus Effort

- Guarantee sec props for entire system
- Only implement and prove small shim
- Radically ease verification burden
- Prove actual code correct
Mitigating the Burden of Proof

1: Scaling proofs to critical infrastructure

Formal shim verification for large apps

QUARK: browser with security guarantees

2: Evolving formally verified systems

Reflex DSL exploits domain for proof auto
Mitigating the Burden of Proof

1: Scaling proofs to critical infrastructure

*Formal shim verification for large apps*

*QUARK: browser with security guarantees*

2: Evolving formally verified systems

*Reflex DSL exploits domain for proof auto*
Browsers: Critical Infrastructure
Browsers: Vulnerable

Defenses / Policies:
[Jang et al. W2SP]
[Stamm et al. WWW]
[Jackson et al. W2SP]
[Barth et al. CCS]
[Singh et al. OAKLAND]

Complex + Implementation Bugs
Quark: Verified Browser

- Resources
  - Shim
    - Sandbox
      - Untrusted Code
Quark: Verified Browser

Resources

Shim

Sandbox

Untrusted Code
Quark: Verified Browser

Resources

- network
- persistent storage
- user interface

Net

Shim

Sandbox

Untrusted Code
Quark: Verified Browser

Sandbox

Untrusted Code

Net

Shim

Resources
**Quark: Verified Browser**

- **Quark Kernel**
  - Code, spec, proof in Coq

- **Sandbox**
  - Untrusted Code

- **Resources**
  - Shim

- **Net**
Quark: Verified Browser

Quark Kernel

Resources
Shim

Net

Sandbox
Untrusted Code
Quark: Verified Browser

- Quark Kernel
  - Resources
  - Shim
  - Untrusted Code
    - *browser components*
    - *run as separate procs*
    - *strictly sandboxed*

- Sandbox
  - Untrusted Code

- Net
Quark: Verified Browser

Resources
Shim
Untrusted Code

browser components
run as separate procs
strictly sandboxed
talk to kernel over pipe
Quark: Verified Browser

- Quark Kernel
- Resources
- Shim
- Untrusted Code

two component types

Sandbox

Untrusted Code
Quark: Verified Browser

Quark Kernel

Resources
Shim
Untrusted Code

two component types

WebKit
Tab
modified WebKit, intercept accesses
Quark: Verified Browser

Quark Kernel

WebKit Tab

Net

Resources

Shim

Untrusted Code

two component types
Quark: Verified Browser

- **Net**
- **WebKit Tab**
- **Quark Kernel**
- **Cookie Manager**
- **Resources Shim**
- **Untrusted Code**

- *two component types*

- *written in Python, manages single domain*
Quark: Verified Browser

Resources
Shim
Untrusted Code

*two component types*

WebKit tabs

*cookie managers*
Quark: Verified Browser

Resources
Shim
Untrusted Code

two component types
WebKit tabs
cookie managers
several instances each
Quark: Verified Browser

Net

Quark Kernel

WebKit Tab

Cookie Manager
Quark: Verified Browser

Quark Kernel
Quark Kernel
Quark Kernel: Code, Spec, Proof
Quark Kernel: *Code*, Spec, Proof
Quark Kernel: Code, Spec, Proof
Definition kstep ...
Definition kstep(focused_tab, tabs) :=
...

kernel state
Definition kstep(focused_tab, tabs) :=
    f <- select(stdin, tabs);
...

Unix-style select to find a component pipe ready to read
Definition kstep(focused_tab, tabs) :=
  f <- select(stdin, tabs);
match f with
  | Stdin =>
    ...
  | Tab t =>
    ...

  case: f is user input
  case: f is tab pipe
Definition \texttt{kstep(focused\_tab, \textit{tabs}) :=}
\begin{verbatim}
f \leftarrow \text{select}(\textit{stdin}, \textit{tabs});
\text{match} \ f \ \text{with}
    | \text{Stdin} => 
        \text{cmd} \leftarrow \text{read\_cmd}(\textit{stdin});
        ...
    | \text{Tab} \ t \Rightarrow 
        ...
\end{verbatim}

\textit{read command from user over stdin}
Definition kstep(focused_tab, tabs) :=
  f <- select(stdin, tabs);
match f with
  | Stdin =>
    cmd <- read_cmd(stdin);
    match cmd with
      | AddTab =>
        ...
      | ...    
      | Tab t =>
        ...
user wants to create and focus a new tab
Definition kstep(focused_tab, tabs) :=
    f <- select(stdin, tabs);
    match f with
    | Stdin =>
      cmd <- read_cmd(stdin);
      match cmd with
      | AddTab =>
        t <- mk_tab();
        ...
      | ...
      | Tab t =>
        ...
create a new tab
Quark Kernel: **Code**, Spec, Proof

**Definition** `kstep(focused_tab, tabs) :=`

```plaintext
f <- select(stdin, tabs);
match f with
| Stdin =>
    cmd <- read_cmd(stdin);
    match cmd with
    | AddTab =>
        t <- mk_tab();
        write_msg(t, Render);
        ...
    | ...
| Tab t =>
    ...
```

tell new tab to render itself
Definition kstep(focused_tab, tabs) :=
    f <- select(stdin, tabs);
    match f with
    | Stdin =>
        cmd <- read_cmd(stdin);
        match cmd with
        | AddTab =>
            t <- mk_tab();
            write_msg(t, Render);
            return (t, t::tabs)
        | ...
    | Tab t =>
        ...

return updated state
Definition kstep(focused_tab, tabs) :=
  f <- select(stdin, tabs);
  match f with
  | Stdin =>
    cmd <- read_cmd(stdin);
    match cmd with
    | AddTab =>
      t <- mk_tab();
      write_msg(t, Render);
      return (t, t::tabs)
    | ...
  | Tab t =>
    ...
  | ...
Definition kstep(focused_tab, tabs) :=
  f <- select(stdin, tabs);
  match f with
  | Stdin =>
    cmd <- read_cmd(stdin);
    match cmd with
    | AddTab =>
      t <- mk_tab();
      write_msg(t, Render);
      return (t, t::tabs)
    | ...
  | Tab t =>
    ...
handle other user commands
Definition kstep(focused_tab, tabs) :=
    f <- select(stdin, tabs);
    match f with
    | Stdin =>
      cmd <- read_cmd(stdin);
      match cmd with
      | AddTab =>
        t <- mk_tab();
        write_msg(t, Render);
        return (t, t::tabs)
      | ...
    | Tab t =>
      ...
    handle requests from tabs
Definition kstep(focused_tab, tabs) :=
    f <- select(stdin, tabs);
match f with
    | Stdin =>
        cmd <- read_cmd(stdin);
        match cmd with
            | AddTab =>
                t <- mk_tab();
                write_msg(t, Render);
                return (t, t::tabs)
            | ...
    | Tab t =>
        ...

Quark Kernel: Code, Spec, Proof
Quark Kernel: Code, \textit{Spec}, Proof
Quark Kernel: Code, *Spec*, Proof

Safety properties to mitigate attacks

*restrict kernel behavior to only safe executions*

Example: mitigate phishing attacks

*prevent tricks that get users to divulge secrets*
Quark Kernel: Code, Spec, Proof

Safety properties to mitigate attacks

restrict kernel behavior to only safe executions

Example: mitigate phishing attacks

prevent tricks that get users to divulge secrets

spoofed!
Quark Kernel: Code, Spec, Proof

Specify correct behavior wrt syscall seqs

read(), write(), open(), write(), ...
Specify correct behavior wrt syscall seqs

trace: all syscalls made by Quark kernel during execution
Quark Kernel: Code, Spec, Proof

Specify correct behavior wrt syscall seqs

kstep()  kstep()  kstep()  kstep()
Quark Kernel: Code, Spec, Proof

Specify correct behavior wrt syscall seqs

structure of produceable traces supports spec & proof
Specify correct behavior wrt syscall seqs

structure of produceable traces supports spec & proof

Example: address bar correctness
Specify correct behavior wrt syscall seqs
structure of produceable traces supports spec & proof

Example: address bar correctness

forall trace tab domain,

for any trace, tab, and domain
Specify correct behavior wrt syscall seqs

structure of produceable traces supports spec & proof

Example: address bar correctness

forall trace tab domain,
  quark_produced(trace) \land 
  ...

if Quark could have produced this trace
Specify correct behavior wrt syscall seqs structure of produceable traces supports spec & proof

Example: address bar correctness

\[
\text{foreach trace tab domain,}
\]
\[
\text{quark\_produced(trace)} \land
\]
\[
\text{tab} = \text{cur\_tab(trace)} \land
\]
\[
\ldots
\]

and \text{tab} is the selected tab in this trace
Specify correct behavior wrt syscall seqs
structure of produceable traces supports spec & proof

Example: address bar correctness

forall trace tab
quark_produced(trace) /
  tab = cur_tab(trace) /
  domain = addr_bar(trace) -> ...

and domain displayed in address bar for this trace
Specify correct behavior wrt syscall seqs
structure of produceable traces supports spec & proof

Example: address bar correctness

forall trace tab do
  quark_produced(trace) /
  tab = cur_tab(trace) /
  domain = addr_bar(trace) /
  domain = tab_domain(tab)

then domain is the domain of the focused tab
Specify correct behavior wrt syscall seqs

structure of produceable traces supports spec & proof

Example: address bar correctness

forall trace tab domain,
  quark_produced(trace) \land
  tab = cur_tab(trace) \land
  domain = addr_bar(trace) \rightarrow
  domain = tab_domain(tab)
Formal Security Properties

Tab Non-Interference
no tab affects kernel interaction with another tab

Cookie Confidentiality and Integrity
cookies only accessed by tabs of same domain

Address Bar Integrity and Correctness
address bar accurate, only modified by user action
Quark Kernel: Code, Spec, Proof
Quark Kernel: Code, Spec, Proof
Prove kernel code satisfies sec props by induction on traces Quark can produce
Prove kernel code satisfies sec props

by induction on traces Quark can produce

induction hypothesis:
trace valid up to this point
Prove kernel code satisfies sec props

by induction on traces Quark can produce

induction hypothesis: trace valid up to this point

proof obligation: still valid after step?
Quark Kernel: Code, Spec, Proof

induction hypothesis: trace valid up to this point

proof obligation: still valid after step?

Proceed by case analysis on kstep() what syscalls can be appended to trace? will they still satisfy all security properties? prove each case interactively in proof assistant
Quark Kernel: Code, Spec, *Proof*

Proving required diverse range of tools

- **monads** encoding I/O in functional language
- **Hoare logic** reasoning about imperative programs
- **op. semantics** defining correctness of Quark kernel
- **linear logic** proving resources created / destroyed

YNot

[Naneveski et al. ICFP 08]
Key Insight: FSV Effective

- Guarantee sec props for browser
- Use state-of-the-art components
- Only prove simple browser kernel
Formally Verified Browser!
Extending Quark

Filesystem access, sound, history
could be implemented w/out major redesign

Finer grained resource accesses
support mashups and plugins

Liveness properties
no blocking, kernel eventually services all requests
Trusted Computing Base

Infrastructure we assume correct

*bugs here can invalidate our formal guarantees*

---

**Fundamental**

- Statement of security properties
- Coq (soundness, proof checker)

---

**Eventually Verified**

- OCaml [VeriML]
- Tab Sandbox [RockSalt]
- Operating System [seL4]
- ...

[active research]
Quark Development Effort

150 lines of security props
900 lines of kernel code
4,500 lines of proofs
1,000,000 lines of WebKit
Quark Development Effort

150 lines of security props
900 lines of kernel code
4,500 lines of proofs
1,000,000 lines of WebKit

week
months
Mitigating the Burden of Proof

1: Scaling proofs to critical infrastructure
   *Formal shim verification for large apps*
   QUARK: browser with security guarantees

2: Evolving formally verified systems
   Reflex DSL exploits domain for proof auto
1: Scaling proofs to critical infrastructure

Formal shim verification for large apps

QUARK: browser with security guarantees

2: Evolving formally verified systems

Reflex DSL exploits domain for proof auto
Struggle Against Formality Inertia

Adding cookies to Quark quite difficult

all the pieces already there, still took over a month

Proof updates repetitive and shallow

sensitive proof scripts, changes not mechanical

match svec_ith PAYREST i as _vi return
  forall (EQ: (svec_ith (projT2 (existT vcdesc' ENVD_SIZE PAYREST)) i) = _vi),
  match _vi as __d return (base_term (existT vcdesc' ENVD_SIZE PAYREST) __d -> Prop) with
  | Desc d => fun _ => True
  | Comp c => fun b => FdSet.In
    (comp_fd (projT1 (eval_base_term (envd:=existT _ ENVD_SIZE PAYREST) erest b))) fds end
  match EQ in _ = __vi return base_term __vi with Logic.eq_refl =>
    Var (existT vcdesc' ENVD_SIZE PAYREST) i end
->
match _vi as __d return (base_term (existT vcdesc' (S ENVD_SIZE) (PAY0, PAYREST)) __d -> Prop) with
  | Desc d => fun _ => True
  | Comp c => fun b =>
    FdSet.In (comp_fd (projT1 (eval_base_term (envd:=existT _ (S ENVD_SIZE) (PAY0, PAYREST)) (e0, erest) b))) fds end
  match EQ in _ = __vi return base_term __vi with Logic.eq_refl =>
    Var (existT vcdesc' (S ENVD_SIZE) (PAY0, PAYREST)) (Some i) end
with
  | Desc d => _ | Comp c => _ end (Logic.eq_refl _)
Division of Labor (to scale)
Division of Labor

Ideal?

Spec

Code

Proof
Division of Labor

just application specific bits

(no manual proof)
Division of Labor

Spec → ? → Spec

Code

Proof
Division of Labor

Spec

Code

DSL

Spec

Code

Proof
Division of Labor

- Spec → DSL → Code

- Easier to implement, verify, and maintain

- Does not demand verification expertise

- Proof
**Reflex**: a DSL for Reactive Systems

“kernel based archs, well suited to FSV design”

Exploit structure of app domain

**Components** = ...
**Messages** = ...

*example*
- *e.g.* tabs, cookie managers
- *e.g.* GetCookie, MouseClick
Reflex: a DSL for Reactive Systems

Exploit structure of app domain

kernel based archs, well suited to FSV design

Components = ...
Messages = ...

Handlers:

When C sends M:
    ...
    ... react by:
        updating state
        accessing resources
        sending messages

when component C sends message M ...

loop free!
**Reflex**: a DSL for Reactive Systems [PLDI 14]

Exploit structure of app domain

*kernel based archs, well suited to FSV design*

Provide expressive spec language

*subset of LTL and non-interference properties*

forall d c,

\[
\text{[Recv(Tab(d), CookieSet(c))] Enables [Send(CookieMgr(d), CookieSet(c))]}
\]
Reflex: a DSL for Reactive Systems [PLDI 14]

Exploit structure of app domain

*kernel based archs, well suited to FSV design*

Provide expressive spec language

*subset of LTL and non-interference properties*

Auto prove user-provided specs

*exploit domain, ensure all traces match spec*

Counterexample-driven search discovers invariants.
Reflex: a DSL for Reactive Systems

Reflex Effective:

- Prototype sshd, browser, httpd
- Specify basic access controls
- Auto prove user-provided specs
# Reflex: 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)*
**Reflex: Development Effort**

<table>
<thead>
<tr>
<th>Reflex :</th>
<th>Many reactive systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>7500 lines of Coq</td>
<td></td>
</tr>
</tbody>
</table>

| Web browser | SSH server | Web server |

**Quark Web browser :**

5500 lines of Coq

Single reactive system
Mitigating the Burden of Proof

1: Scaling proofs to critical infrastructure

Formal shim verification for large apps

QUARK: browser with security guarantees

2: Evolving formally verified systems

Reflex DSL exploits domain for proof auto
AND NOW FOR SOMETHING COMPLETELY DIFFERENT
Double Trouble

\[
\begin{align*}
x &= 0.1 + 0.2; \\
\text{if} (x \neq 0.3) \\
\text{printf(“wat. \\
\n”);} \\
\end{align*}
\]

\[
\frac{(-b) - \sqrt{b^2 - 4(a \cdot c)}}{2 \cdot a}
\]
Less Double Trouble
Neutron Beams
1. Quark: A secure Web Browser with a Formally Verified Kernel (ucsd.edu)
   141 points by herbal 4 hours ago | 38 comments

2. Writing an nginx authentication module in Lua and Go (stavros.io)
   41 points by stavrosK 2 hours ago | 7 comments

3. Code & Conquer: A War Game for Coders (codeandconquer.co)
   60 points by elliottcarson 2 hours ago | 13 comments

4. Proposal to Change the Default TLS Ciphersuites Offered by Browsers (briansmith.org)
   78 points by lep 5 hours ago | 44 comments

5. The backlash against running firms like progressive schools has begun (economist.com)
   26 points by akr 2 hours ago | 16 comments

6. Darkness (wegnerdesign.com)
   41 points by yesplorer 4 hours ago | 14 comments

7. Big Data and the Soviet Ghosts (mempko.wordpress.com)
   36 points by mempko 4 hours ago | 8 comments

8. Startup Ideas Every Nerd Has (That Never Work) (swombat.com)
   5 points by luu 23 minutes ago | discuss

9. GCP - cp with a progress bar (hecticgeek.com)
   18 points by dannyr 2 hours ago | 14 comments

10. Doing Good in the Addiction Economy (kajsotala.fi)
    58 points by kaj_sotala 6 hours ago | 7 comments

11. Yahoo says U.S. sought data on 48,332 user accounts in 2013 ( wired.com)
    5 points by tareqak 35 minutes ago

12. Setup a Docker Container (coolbae.com)
    16 points by mbj 5 hours ago

Achievement unlocked
Goal: mitigate formality inertia

address scaling and evolving formally verified systems

1. Extend verification frontier

develop techniques to verify critical “pinch points”

2. Make verification accessible

equip domain experts with effective tools
Verifying Optimizations

Rich compiler correctness history:

*McCarthy 67, Samet 75, Cousot 77, …*

Already solved?

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

[Yang et al. PLDI 11]

many optimization bugs

lacks many optimizations
Verifying Optimizations
Verifying Optimizations
Verifying Optimizations

Proof original and opt code equivalent.
Verifying Optimizations

Proof original and opt code equivalent.

Construct bisimulation relation:
Verifying Optimizations

Proof original and opt code equivalent.

Construct bisimulation relation: if orig and opt in equal states

CompCert

C

Asm
Verifying Optimizations

Proof original and opt code equivalent.

Construct a bisimulation relation: if orig and opt in equal states and orig prog can take some action.
Verifying Optimizations

Proof original and opt code equivalent.

Construct bisimulation relation:

\[ P \xlongequal{\text{bis}} P' \]

then opt prog can take same action to another equal state
Verifying Optimizations

Proof original and opt code equivalent.

Construct bisimulation relation:

\[ P \sim P' \implies \text{anything orig can do, opt can do too} \]
Verifying Optimizations

Proof original and opt code equivalent.

Construct bisimulation relation:

\[ \vdash P \Rightarrow P' \wedge P' \Rightarrow P \]

... also prove inverse

CompCert

C \rightarrow \checkmark \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow S

Asm
Verifying Optimizations

Proof original and opt code equivalent.

Construct bisimulation relation:

\[ \begin{align*}
P & \quad P' \\
P' & \quad P \\
\Rightarrow & \quad \land \\
\Rightarrow & \quad P \\
\Rightarrow & \quad P'
\end{align*} \]

together, implies indistinguishability

CompCert

\[
\begin{array}{ccccccc}
C & \xrightarrow{\text{CompCert}} & 1 & \xrightarrow{\text{CompCert}} & 2 & \xrightarrow{\text{CompCert}} & 3 & \xrightarrow{\text{CompCert}} & 4 & \xrightarrow{\text{CompCert}} & \text{ ASM}
\end{array}
\]
Verifying Optimizations

Proof original and opt code equivalent.

Construct bisimulation relation:

\[ P \quad P' \quad P \quad P' \quad P \quad P' \quad P \quad P' \]

\[ \implies \wedge \implies \]
Verifying Optimizations

Formally Proved:
Rewrites locally correct
\[ \Rightarrow \exists \text{ bisimulation relation} \]

CompCert
XCert
CompCert

C

Asm

Semantic Proofs
Rewrite
Local Proofs
Verifying Optimizations

Rewrite Rule

PEC

Rewrite
Local Proofs

CompCert

XCert

Verifying Optimizations
Verifying Optimizations

Auto prove complex opts:
- software pipelining
- loop fusion / distribution
- loop unswitching
- ...

Rewrite Rule

PEC

Local Proofs

Rewrite

CompCert

XCert

C

Asm
Verifying Optimizations

Rewrite Rule → PEC → Rewrite

Local Proofs

CompCert → XCert → Asm
Future Work

Generating and evaluating specs

*techniques to ensure spec matches intuition*

Even perfect program verification can only establish that a program meets its specification... Much of the essence of building a program is in fact the debugging of the specification.

*Frederick P. Brooks, Jr.*
*No Silver Bullet*
Software Infrastructure
Quark Usability

And we have lift off! Celebrate 50 years of the Kennedy Space Center with Google Maps.
Browsers: Critical Infrastructure
Browsers: Critical Infrastructure
Browsers: Critical Infrastructure
Browsers: Critical Infrastructure
Browsers: Critical Infrastructure