Lecture 23
Verified Systems
Software Infrastructure
Software Infrastructure is Shaky

The New York Times

Cars’ Computer Systems Called at Risk to Hackers

By JOHN MARKOFF

Automobiles, which will be increasingly vulnerable in the near future, could be vulnerable to hackers, two teams of computer scientists at the University of California, San Diego, reported yesterday in a draft they presented next week.

The scientists said that remote control functions of an automobile can be hacked and that it was possible to undermine its security by altering its software.

“We demonstrated adversarial attacks on two different automobiles, including disabling the brakes, selectively turning off the engine, and so on,” they wrote in the draft paper, "Modern Automobile."

In the paper, which will be presented at the University of California, San Diego, reported engineers in the design of their computers to the potential threat of hackers who may increasingly control modern cars.

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 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 are expected at the NYSE and Nasdaq OMX unless they can fix some of these issues.
Software Infrastructure is Shaky
Software Infrastructure is Shaky
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!
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:**  

<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]

[CompCert] Leroy POPL 06
**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>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]

<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
Today

- prog error
- patch

Promise

- Proof

- no prog errors
Today

Proof

Burden

Promise

Proof

no prog errors

prog error

patch

↓

↓
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 ➞ Proof Assistant

Spec ➞ Proof Assistant

Proof Assistant ➞ Grad

Grad ➞ interactively show code satisfies specification
Fully Formal Verification

Code

Spec

Proof Assistant

Grad

ML

x86

compile down to machine code
Fully Formal Verification

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

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

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

claim program satisfies spec
construct proof interactively
Fully Formal Verification

```coq
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 O at 1. rewrite plus_comm.
  apply plus_le_compat. apply le 0 n. apply le_refl.
Qed.
```
Fully Formal Verification

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

JavaScript

JPEG

HTML
Formally Verify a Browser?!

Resources

- JavaScript
- JPEG
- HTML

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

Resources

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

JavaScript

JPEG

HTML
Formally Verify a Browser?!

Resources

Isolate

sandbox untrusted code

JavaScript

JPEG

HTML
Formally Verify a Browser?!

Isolate
sandbox untrusted code

Implement shim
guards resource access
Formally Verify a Browser?!

**Resources**

- **Shim**
  - Guard resource access
  - Isolate untrusted code

**JavaScript**

- Implement shim
  - Prove security policy

**JPEG**

**HTML**
Formal Shim Verification

- **Isolate**
  - sandbox untrusted code

- **Implement shim**
  - guards resource access

- **Verify shim**
  - prove security policy

Diagram:
- Resources
- Shim
- JavaScript
- JPEG
- HTML

- ✔
Formal Shim Verification

- Implement shim
- Verify shim
- Isolate
- Untrusted Code
- Sandbox

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
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

Sandbox

Untrusted Code

Shim

Net
**Quark: Verified Browser**

- **Quark Kernel**: code, spec, proof in Coq
- **Shim**: Quark browser kernel
- **Resources**: Net, Untrusted Code, Sandbox

The diagram illustrates the architecture of the Quark Verified Browser, emphasizing the verified Quark kernel and its interaction with untrusted resources through a secure shim.
Quark: Verified Browser

Resources

Shim

Quark Kernel

Net

Sandbox

Untrusted Code
Quark: Verified Browser

Quark Kernel

Resources
Shim

Untrusted Code

browser components
run as separate procs
strictly sandboxed
Quark: Verified Browser

Resources
Shim
Untrusted Code

Quark Kernel

Sandbox
Untrusted Code

untrusted code
run as separate procs
strictly sandboxed
talk to kernel over pipe
Quark: Verified Browser

Resources
Shim
Untrusted Code
two component types

Quark Kernel

Sandbox
Untrusted Code
Quark: Verified Browser

Resources
Shim
Untrusted Code

two component types

Quark Kernel

WebKit
Tab

modified WebKit, intercept accesses
Quark: Verified Browser

Resources
Shim
Untrusted Code

two component types
Quark: Verified Browser

Quark Kernel

WebKit Tab

Cookie Manager

two component types written in Python, manages single domain

Resources
Shim
Untrusted Code

two component types
Quark: Verified Browser

Resources
Shim
Untrusted Code

two component types
WebKit tabs
cookie managers

Quark Kernel

Net

WebKit Tab
Cookie Manager
Quark: Verified Browser

Resources
Shim
Untrusted Code
two component types
WebKit tabs
cookie managers
several instances each
Quark: Verified Browser
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 =>
    ...  \(\text{case: } f \text{ is user input}\)
  | Tab t =>
    ...  \(\text{case: } f \text{ is tab pipe}\)
Definition kstep(focused_tab, tabs) :=
    f <- select(stdin, tabs);
    match f with
    | Stdin =>
        cmd <- read_cmd(stdin);
        ...
    | Tab t =>
        ...

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 \( \text{kstep}(\text{focused\_tab}, \text{tabs}) := \)
\[
\begin{align*}
f & \leftarrow \text{select}(\text{stdin}, \text{tabs}); \\
\text{match } f \text{ with } & \\
\mid \text{Stdin} & \Rightarrow \\
& \phantom{\text{match } f \text{ with }} \text{cmd} \leftarrow \text{read\_cmd}(\text{stdin}); \\
& \phantom{\text{match } f \text{ with }} \text{match } \text{cmd} \text{ with } \\
\mid \text{AddTab} & \Rightarrow \\
& \phantom{\text{match } f \text{ with }} t \leftarrow \text{mk\_tab}(); \\
& \ldots \\
\mid \ldots & \\
\mid \text{Tab} t & \Rightarrow \\
& \ldots
\end{align*}
\]
create 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();
      write_msg(t, Render);
      ...
    |
    | 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 =>
      ...
  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, 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*
Specify correct behavior wrt syscall seqs

read(), write(), open(), write(), ...
Quark Kernel: Code, Spec, Proof

Specify correct behavior wrt syscall seqs

(trace: all syscalls made by Quark kernel during execution)
Specify correct behavior wrt syscall seqs
Quark Kernel: Code, Spec, Proof

Specify correct behavior wrt syscall seqs

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

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
Quark Kernel: Code, Spec, Proof

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

forall trace tab domain,
  quark_produced(trace) ∧
  tab = cur_tab(trace) ∧
  ...

and 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, domain

\(
\text{quark\_produced}(\text{tab} = \text{cur\_tab}(\text{trace})) \land \text{domain} = \text{addr\_bar}(\text{trace}) \rightarrow
\)

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

Example: address bar correctness

\[
\text{forall trace } \text{tab do } \\text{quark_produced}(\text{trace}) \\text{ \tab = cur_tab(\text{trace})} \\text{ \domain = addr_bar(\text{trace})} \\text{ \domain = tab_domain(\text{tab})}
\]

then \text{domain} is the domain of the focused \text{tab}
Specify correct behavior wrt syscall seqs

structure of producible traces supports spec & proof

Example: address bar correctness

\[
\text{forall trace tab domain,}
\]
\[
\text{quark\_produced}(\text{trace}) \land \\
\text{tab} = \text{cur\_tab}(\text{trace}) \land \\
\text{domain} = \text{addr\_bar}(\text{trace}) \rightarrow \\
\text{domain} = \text{tab\_domain}(\text{tab})
\]
Quark Kernel: Code, Spec, Proof

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
Quark Kernel: Code, Spec, *Proof*

Prove kernel code satisfies sec props

*by induction on traces Quark can produce*

*induction hypothesis:
  trace valid up to this point*
Quark Kernel: Code, Spec, Proof

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 \texttt{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)

**EventuallyVerified**

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

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*

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*
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
Code

? 

Spec
Code

Proof
Division of Labor

Spec
Code

DSL

Spec
Code

Proof
Division of Labor

Easier to implement, verify, and maintain

Does not demand verification expertise

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

Exploit structure of app domain

*kernel based archs, well suited to FSV design*

Components = ...

Messages = ...

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,
    [Recv(Tab(d), CookieSet(c))] Enables [Send(CookieMgr(d), CookieSet(c))] cookie integrity
```
Reflex: a DSL for Reactive Systems

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>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web browser</td>
<td>Domains do not interfere, Cookie integrity, ...</td>
</tr>
<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

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
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

\[ x = 0.1 + 0.2; \]
\[ \text{if } (x != 0.3) \]
\[ \text{printf("wat. \n");} \]

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

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

3. Code & Conquer: A War Game for Coders (codeandconquer.co)
   60 points by elliottcarlson 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 alexfarran 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 (mempho.wordpress.com)
   36 points by mempho 4 hours ago | 8 comments

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

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

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

11. Yahoo says U.S. sought data on 40,222 user accounts in 2013 (apbrain.com)
    5 points by tarekqaq 35 minutes ago

12. Setup a Docker Container for a "Hello, world!" application (kajisotola.fi)
    16 points by mkashenkov 9 hours ago
Thank You!

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: *McCarth 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

CompCert
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 \( P \) and \( P' \), equal states

CompCert

C

Asm
Verifying Optimizations

Proof original and opt code equivalent.

Construct bisimulation relation: if orig and opt in equal states

and orig prog can take some action
Proof original and opt code equivalent.

Construct bisimulation relation:

\[ P \quad \approx \quad P' \quad \Rightarrow \quad P \quad \approx \quad P' \]

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

Proof original and opt code equivalent.

Construct *bisimulation relation*:

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

implies: *anything orig can do, opt can do too*
Verifying Optimizations

Proof original and opt code equivalent.

Construct bisimulation relation:

\[ P \sim P' \land P \sim P' \implies P \sim P' \]

\[ \implies \]

... also prove inverse

CompCert

C → Asm
Verifying Optimizations

Proof original and opt code equivalent.

Construct \textit{bisimulation relation}: 

\begin{align*}
P & \quad \Downarrow \quad \bigwedge \quad \Downarrow \quad P' \\
& \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad P' \\
& \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad P' \\
& \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad P' \\
& \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad P' \\
\end{align*}

together, implies \textit{indistinguishability}

CompCert

C \quad \overset{\checkmark}{\longrightarrow} \quad \text{Asm}
Verifying Optimizations

Proof original and opt code equivalent.

Construct bisimulation relation:

\[ P \sim P' \quad \wedge \quad P \sim P' \quad \Rightarrow \quad P \sim P' \]
Verifying Optimizations

Formally Proved:
Rewrites locally correct
⇒ ∃ bisimulation relation
Verifying Optmizations

Rewrite Rule

PEC

Rewrite
Local Proofs

CompCert

XCert

Verifying Optimizations

CompCert

XCert

C

Asm
Verifying Optimizations

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

Rewrite Rule

CompCert

XCert

PEC

Rewrite

Local Proofs

C

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

- MasterCard
- Wells Fargo
- CHASE
- ING
- E*TRADE
- Gmail
- Calendar
- Twitter
- LinkedIn
- Hotmail
- Yahoo
- Facebook
- Pinterest
Browsers: Critical Infrastructure