Verifying concurrent software using movers in CSPEC

Tej Chajed, Frans Kaashoek, Butler Lampson*, Nickolai Zeldovich
MIT CSAIL and *Microsoft
Concurrent software is difficult to get right

Programmer cannot reason about code in sequence...
Concurrent software is difficult to get right

Programmer cannot reason about code in sequence… instead, must consider many executions:
Concurrent software is difficult to get right

Programmer cannot reason about code in sequence... instead, must consider many executions:

...
Goal: verify concurrent software
Challenge for formal verification

- Proofs must also cover every execution

- Many approaches to managing this complexity
  - movers [Lipton, 1975]
  - rely-guarantee [1983]
  - RGSep [CONCUR 2007]
  - FCSL [PLDI 2015]
  - Iris [POPL 2017, LICS 2018, others]
  - many others
Challenge for formal verification

- Proofs must also cover every execution

- Many approaches to managing this complexity
  - movers [Lipton, 1975]
  - rely-guarantee [1983]
  - RGSep [CONCUR 2007]
  - FCSDL [PLDI 2015]
  - Iris [POPL 2017, LICS 2018, others]
  - many others

- This work: our experience using movers
Movers: reduce concurrent executions to sequential ones
Movers: reduce concurrent executions to sequential ones

1 A 2 3 B

has the same effect as

1 2 3 A B

movers

blue thread

1 2 3

green thread

A B

6
Movers: reduce concurrent executions to sequential ones

1. A 2. 3. B

has the same effect as

1. 2. 3. A B

movers

sequential reasoning

blue thread 1 2 3

green thread A B

1. 2. 3. A B
Prior systems with mover reasoning

**CIVL** [CAV ’15, CAV ’18] framework relies pen & paper proofs

**IronFleet** [SOSP ’15] only move network send/receive
Contribution: CSPEC

- Framework for verifying concurrency in systems software
  - general-purpose movers
  - patterns to support mover reasoning
  - machine checked in Coq to support extensibility
Contribution: CSPEC

- Framework for verifying concurrency in systems software
  - general-purpose movers
  - patterns to support mover reasoning
  - machine checked in Coq to support extensibility
- Case studies using CSPEC
  - Lock-free file-system concurrency
  - Spinlock on top of x86-TSO (see paper)
Case study: mail server using file-system concurrency

file system

spool              mbox 

Mail servers exploit file-system concurrency

```python
# accept
def deliver(msg):
    # spool
    create("/spool/$TID")
    write("/spool/$TID", msg)
    # store
    while True:
        t = time.time()
        if link("/spool/$TID",
                "/mbox/$t"):
            break
    # cleanup
    unlink("/spool/$TID")
```
Mail servers exploit file-system concurrency

```python
# accept
def deliver(msg):
    # spool
create("/spool/$TID")
write("/spool/$TID", msg)
    # store
while True:
    t = time.time()
    if link("/spool/$TID",
             "/mbox/$t"):
        break
    # cleanup
unlink("/spool/$TID")
```
Spooling avoids reading partially-written messages

$TID = 10

# accept
def deliver(msg):
    # spool
    create("/spool/$TID")
    write("/spool/$TID", msg)

# store
while True:
    t = time.time()
    if link("/spool/$TID", "/mbox/$t"):
        break
    # cleanup
    unlink("/spool/$TID")
Spooling avoids reading partially-written messages

$TID = 10

# accept
def deliver(msg):
# spool
    create("/spool/$TID")
write("/spool/$TID", msg)
# store
while True:
    t = time.time()
    if link("/spool/$TID",
            "/mbox/$t"):
        break
# cleanup
unlink("/spool/$TID")

file system

spool

10

mbox

1 2 3
Threads use unique IDs to avoid conflicts

$TID = 10 \quad TID = 11$

```python
# accept
def deliver(msg):
    # spool
    create("/spool/$TID")
    write("/spool/$TID", msg)
    # store
    while True:
        t = time.time()
        if link("/spool/$TID", "/mbox/$t"):
            break
    # cleanup
    unlink("/spool/$TID")
```

[Diagram showing file system operations]
Threads use unique IDs to avoid conflicts

$TID = 10 \quad TID = 11$

```python
# accept
def deliver(msg):
    # spool
    create("/spool/$TID")
    write("/spool/$TID", msg)
# store
while True:
    t = time.time()
    if link("/spool/$TID", "/mbox/$t"):
        break
    # cleanup
    unlink("/spool/$TID")
```

file system

<table>
<thead>
<tr>
<th>spool</th>
<th>mbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1 2 3</td>
</tr>
</tbody>
</table>
Threads use unique IDs to avoid conflicts

$TID = 10 \quad $TID = 11

```python
# accept
def deliver(msg):
    # spool
create("/spool/$TID")
write("/spool/$TID", msg)
    # store
while True:
    t = time.time()
    if link("/spool/$TID", "mbox/$t"):
        break
    # cleanup
unlink("/spool/$TID")
```
def deliver(msg):
    # spool
    create("/spool/$TID")
    write("/spool/$TID", msg)
    # store
    while True:
        t = time.time()
        if link("/spool/$TID", "/mbox/$t"):
            break
    # cleanup
    unlink("/spool/$TID")
def deliver(msg):
    # spool
    create("/spool/$TID")
    write("/spool/$TID", msg)
    # store
    while True:
        t = time.time()
        if link("/spool/$TID",
                "/mbox/$t"):
            break
    # cleanup
    unlink("/spool/$TID")
# accept
def deliver(msg):
    # spool
    create("/spool/$TID")
    write("/spool/$TID", msg)
    # store
    while True:
        t = time.time()
        if link("/spool/$TID",
            "/mbox/$t"):
            break
    # cleanup
    unlink("/spool/$TID")
Delivery concurrency does not use locks

```python
# accept
def deliver(msg):
    # spool
    create("/spool/$TID")
    write("/spool/$TID", msg)
    # store
    while True:
        t = time.time()
        if link("/spool/$TID",
                "/mbox/$t"):
            break
    # cleanup
    unlink("/spool/$TID")
```
Delivery concurrency does not use locks

```python
# accept
def deliver(msg):
    # spool
create("/spool/$TID")
write("/spool/$TID", msg)
    # store
while True:
    t = time.time()
    if link("/spool/$TID",
             "/mbox/$t"):
        break
    # cleanup
unlink("/spool/$TID")
```

file system

<table>
<thead>
<tr>
<th>spool</th>
<th>mbox</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="1" alt="Envelope" /> <img src="2" alt="Envelope" /> <img src="3" alt="Envelope" /> <img src="4" alt="Envelope" /> <img src="5" alt="Envelope" /></td>
<td><img src="1" alt="Envelope" /> <img src="2" alt="Envelope" /> <img src="3" alt="Envelope" /> <img src="4" alt="Envelope" /> <img src="5" alt="Envelope" /></td>
</tr>
</tbody>
</table>
Proving delivery correct in CSPEC

CSPEC provides supporting definitions and theorems
def deliver(msg):
    create("/spool/$TID", msg)
    while True:
        t = time.time()
        if link("/spool/$TID",
                "/mbox/$t"):
            break
    unlink("/spool/$TID")
Proof engineer reasons about file-system operations

def deliver(msg):
    create("/spool/$TID", msg)
    while True:
        t = time.time()
        if link("/spool/$TID", "/mbox/$t"):
            break
    unlink("/spool/$TID")

collapsed to one operation
create("/spool/$TID")
write("/spool/$TID", msg)

create(/sp/$TID, msg)
link(/sp/$TID, /mbox/$t)
link(/sp/$TID, /mbox/$t)
unlink(/sp/$TID)

✓ ✓ ✓ ✓
Proof engineer reasons about interleaving of file-system operations

```python
def deliver(msg):
    create("/spool/$TID", msg)
    while True:
        t = time.time()
        if link("/spool/$TID", "/mbox/$t"):
            break
    unlink("/spool/$TID")
```

We assume file-system operations are atomic
Proving atomicity of delivery

atomicity: concurrent deliveries appear to execute all at once (in some order)
Proving atomicity of delivery

atomicity: concurrent deliveries appear to execute all at once (in some order)

Step 1: developer identifies commit point
Proving atomicity of delivery

**atomicity**: concurrent deliveries appear to execute all at once (in some order)

Step 1: developer identifies commit point
Step 2: prove operation occurs logically at commit point
Example of movers for this execution

create  create  link ✓  link ✓  unlink  unlink
Example of movers for this execution
Example of movers for this execution

create
create
link ✓
link ×
link ✓
unlink
unlink

create
link ✓
create
link ×
link ✓
unlink
unlink

create
link ✓
unlink
create
link ×
link ✓
unlink

Right mover can be reordered after any green thread operation
Right mover can be reordered after any green thread operation

left movers are the converse
Movers need to consider only *possible* operations from other threads

A is a *right mover* if for all *green* operations, 

*left movers* are the converse

- `create(/sp/$TID, msg)`
- `link(/sp/$TID, /mbox/$t)`
- `link(/sp/$TID, /mbox/$t)`
- `unlink(/sp/$TID)`
Example mover proof: failing \texttt{link} is a \textit{right mover}

Proof sketch (only \texttt{link} case):

\begin{center}
\begin{tabular}{c}
\texttt{link}(/sp/$TID$, /mbox/$t$) \rightarrow \texttt{EEXISTS X} \\
\texttt{link}(/sp/$TID$, /mbox/$t$) \rightarrow \checkmark \\
\texttt{link}(/sp/$TID$, /mbox/$t$) \rightarrow \checkmark \\
\texttt{link}(/sp/$TID$, /mbox/$t$) \rightarrow \texttt{EEXISTS X}
\end{tabular}
\end{center}
Example mover proof: failing link is a right mover

Proof sketch (only case):

\[
\text{link}(\text{/sp/} \langle \text{TID}, \text{/mbox/} \langle t \rangle) \xrightarrow{\text{✓}} \text{link}(\text{/sp/} \langle \text{TID}, \text{/mbox/} \langle t \rangle) \xrightarrow{\text{✓}} \text{link}(\text{/sp/} \langle \text{TID}, \text{/mbox/} \langle t \rangle) \xrightarrow{\text{✓}} \text{link}(\text{/sp/} \langle \text{TID}, \text{/mbox/} \langle t \rangle) \xrightarrow{\text{✗}} \text{EEXISTS} \xrightarrow{\text{✗}} \text{EEXISTS} \xrightarrow{\text{✗}} \text{EEXISTS} \xrightarrow{\text{✗}} \text{EEXISTS}
\]

\[ t \neq \langle t \rangle \quad \text{(otherwise link then link is impossible)} \]
Example mover proof: failing link is a right mover

Proof sketch (only link case):

\[ \text{link}(\text{/sp/}TID, \text{/mbox/t}) \xrightarrow{EEXISTS} \exists \]

\[ \text{link}(\text{/sp/}TID, \text{/mbox/t}) \xrightarrow{✓} \]

\[ \text{link}(\text{/sp/}TID, \text{/mbox/t}) \xrightarrow{✓} \]

\[ \text{link}(\text{/sp/}TID, \text{/mbox/t}) \xrightarrow{EEXISTS} \]

\[ t \neq t \] (otherwise \( \text{link} \) then \( \text{link} \) is impossible)

\[ \Rightarrow \text{link} \text{ operations are independent} \]
Failing link does not move left
Failing link does not move left

\[
\text{link(} /sp/\$TID, /mbox/\$t) \rightarrow \checkmark
\]

\[
\text{link(} /sp/\$TID, /mbox/\$t) \rightarrow \text{EEXISTS } \times
\]

\[
\text{link(} /sp/\$TID, /mbox/\$t) \rightarrow \text{EEXISTS } \times
\]

\[
\text{link(} /sp/\$TID, /mbox/\$t) \rightarrow \checkmark
\]

\[
\text{if } \$t = \$t
\]
Challenge: how to limit what other operations to consider in mover proofs?

- deliver
- create(f, d)
- link(f1, f2)
- unlink(f)
- rename(f1, f2)
Challenge: how to limit what other operations to consider in mover proofs?

Delivery

- deliver

mover proof?

File system

- create(f, d)
- link(f1, f2)
- unlink(f)
- rename(f1, f2)

create(f1, d) create(f2, d) create(f2, d) create(f1, d)

if filenames are identical
Layers enable mover reasoning

Layers **limit** what operations are available

\[\implies\text{use *multiple layers* to make operations movers}\]

- **Delivery**
  - `deliver`

- **File system**
  - `create(f, d)`
  - `link(f1, f2)`
  - `unlink(f)`
  - `rename(f1, f2)`
Layers enable mover reasoning

Layers limit what operations are available

⇒ use multiple layers to make operations movers

- create(/spool/$TID, d)
- link(/spool/$TID, /mbox/$t)
- unlink(/spool/$TID)

mover proof ✓

restrict arguments to include $TID
Layers enable mover reasoning

Layers limit what operations are available

⇒ use multiple layers to make operations movers

Delivery

Restricted file system

File system

upper layers can only use restricted operations

- create(/spool/$TID, d)
- link(/spool/$TID, /mbox/$t)
- unlink(/spool/$TID)

mover proof ✓
Movers are a layer proof pattern

Obligation for developer: movers for each implementation
Movers are a layer proof pattern

Obligation for developer: movers for each implementation

```python
def foo:
    A → B → C → D

def bar:
    B → A → C
```

layer 1

```python
foo
bar
```

layer 2

```
A → B → C → D
```
Movers are a layer proof pattern

Obligation for developer: movers for each implementation

```
def foo:
    A B C D
```

```
def bar:
    B A C
```

CSPEC theorem: entire layer implementation is atomic

```
layer 1

foo  bar

layer 2

A B C D
```
CSPEC provides other patterns to support mover reasoning

(see paper for details)

- Abstraction / forward simulation
- Invariants
- Error state
- Protocols
- Retry loops
- Partitioning
Using CSPEC to verify CMAIL

Coq

CMAIL (Coq)
- mail library spec
- implementation layers
- patterns
- file-system spec

CSPEC

auto generated framework
Using CSPEC to verify CMAIL

Coq

CMAIL (Coq)

- mail library spec
- implementation layers
- patterns
- file-system spec

CSPEC

CMAIL (Haskell)

extracted implementation

calls to file-system

SMTP + POP3

auto generated framework
Using CSPEC to verify CMAIL

CMAIL (Coq)
- mail library spec
- implementation layers
- patterns
- file-system spec

CMAIL (Haskell)
- extracted implementation
- calls to file-system
- SMTP + POP3

GHC
- executable
- Linux

CSPEC

Coq calls to file-system

Haskell

extracted implementation

auto generated framework
What is proven vs. assumed correct?

- CMAIL (Coq)
  - mail library spec
  - implementation layers
  - patterns
  - file-system spec
  - Coq proof checker
    - ok ✓

- CMAIL (Haskell)
  - extracted implementation
    - calls to file-system
    - SMTP + POP3
  - GHC
    - executable
    - Linux

- CSPEC

- auto generated
- proven
- assumed correct
Concurrency inside CMAIL is proven

- CMAIL (Coq)
  - mail library spec
  - implementation layers
  - patterns
  - file-system spec
- Coq extraction
- GHC
  - executable
  - Linux

CMAIL (Haskell)
- extracted implementation
  - calls to file-system
  - SMTP + POP3

Coq proof checker
- ok ✓

auto generated
proven
assumed correct
Trust that the tools and OS are correct

Coq

CMAIL (Coq)
- mail library spec
- implementation layers
- patterns
- file-system spec

Coq extraction

Coq proof checker

ok ✓

CMAIL (Haskell)
- extracted implementation

calls to file-system

SMTP + POP3

GHC

 executable

Linux

auto generated
proven
assumed correct
Mail server-specific assumptions

CMAIL (Coq)

mail library spec
implementation layers
patterns
file-system spec

CMAIL (Haskell)

extracted implementation
calls to file-system
SMTP + POP3

CSPEC

Coq extraction

GHC

executable
Linux

auto generated

proven
assumed correct

Coq proof checker
ok ✓
Evaluation

- Can CMAIL exploit file-system concurrency for speedup?
- How much effort was verifying CMAIL?
- What is the benefit of CSPEC’s machine-checked proofs?
CMAIL achieves speedup with multiple cores

The chart shows the performance of CMAIL and GoMail in terms of kreq/s (thousands of requests per second) as the number of cores increases. The x-axis represents the number of cores, ranging from 1 to 12, and the y-axis represents kreq/s, ranging from 0 to 140. The blue circles represent CMAIL, and the green squares represent GoMail. The trend indicates a linear increase in performance with the number of cores.
CMAIL was work but doable

<table>
<thead>
<tr>
<th></th>
<th>proof:code ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMAIL</td>
<td>11.5x</td>
</tr>
<tr>
<td>CertiKOS</td>
<td>13.8x</td>
</tr>
<tr>
<td>IronFleet</td>
<td>7.7x</td>
</tr>
<tr>
<td>IronClad</td>
<td>4.8x</td>
</tr>
<tr>
<td>CompCert</td>
<td>4.6x</td>
</tr>
</tbody>
</table>

Took two authors 6 months
Machine-checked proofs give confidence in framework changes

Three anecdotes of changes to CSPEC:

Machine-checked proofs ensure soundness of entire system
Machine-checked proofs give confidence in framework changes

Three anecdotes of changes to CSPEC:

- Implemented **partitioning pattern** to support multiple users

Machine-checked proofs ensure soundness of entire system
Machine-checked proofs give confidence in framework changes

Three anecdotes of changes to CSPEC:

• Implemented **partitioning pattern** to support multiple users
• Improved **mover pattern** for a CMAIL left mover proof

Machine-checked proofs ensure soundness of entire system
Machine-checked proofs give confidence in framework changes

Three anecdotes of changes to CSPEC:

• Implemented **partitioning pattern** to support multiple users
• Improved **mover pattern** for a CMAIL left mover proof
• Implemented **error-state pattern** for the x86-TSO lock proof

Machine-checked proofs ensure soundness of entire system
CSPEC is a framework for verifying concurrency in systems software

- Layers and patterns (esp. movers) make proofs manageable
- Machine-checked framework supports adding new patterns
- Evaluated by verifying mail server and x86-TSO lock

github.com/mit-pdos/cspec
CSPEC is a framework for verifying concurrency in systems software

- Layers and patterns (esp. movers) make proofs manageable
- Machine-checked framework supports adding new patterns
- Evaluated by verifying mail server and x86-TSO lock

[github.com/mit-pdos/cspec](https://github.com/mit-pdos/cspec)

poster #1
Backup slides

CMAIL perf experimental setup
Performance experiment setup for CMAIL

process

client

deliver + pickup

CMAIL

core 1

in-memory file system
Performance experiment setup for CMAIL

- Process
  - Client
  - Deliver + pickup
  - CMAIL
  - Core 1

- Process
  - Client
  - Deliver + pickup
  - CMAIL
  - Core 2

- Process
  - Client
  - Deliver + pickup
  - CMAIL
  - Core 12

In-memory file system