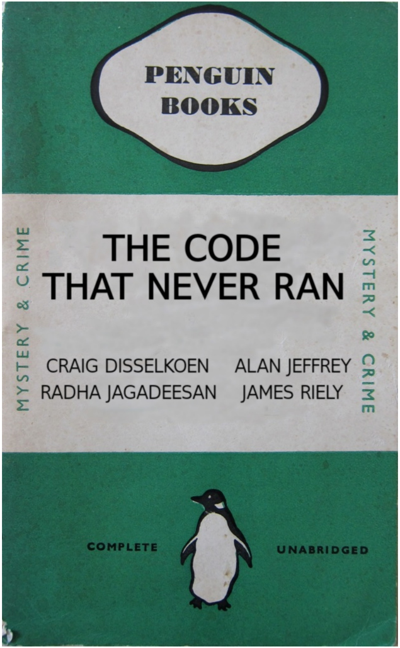
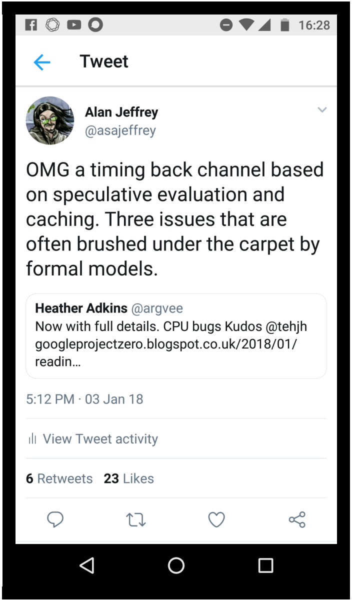


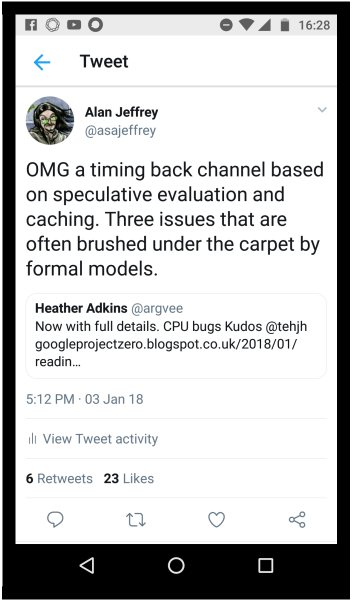
A classic locked-room mystery.
Eve was in the false branch of a conditional the whole time,
how could she do it?

Mozilla Research | DePaul University | U. California San Diego

Why? Spectre!


16:28
Tweet
Alan Jeffrey
@asajeffrey
OMG a timing back channel based on speculative evaluation and caching. Three issues that are often brushed under the carpet by formal models.
Heather Adkins @argvee
Now with full details. CPU bugs Kudos @tehjh
googleprojectzero.blogspot.co.uk/2018/01/
readin...
5:12 PM · 03 Jan 18
View Tweet activity
6 Retweets 23 Likes

Why? Spectre!



Allows reading whole process address space.

Attacks bypass dynamic security checks:

```
if canRead(SECRET) {
  doStuffWith(SECRET);
}
```

Most formal models ignore code in branches that aren't taken.

Models that include speculation?

There are some models that include speculation *relaxed memory models*:

- *The Java Memory Model*
  Manson, Pugh and Adve, 2005.
- *Generative Operational Semantics for Relaxed Memory Models*
  Jagadeesan, Pitcher and Riely, 2010.
- *A promising semantics for relaxed-memory concurrency*
  Kang, Hur, Lahav, Vafeiadis and Dreyer, 2017.

Models that include speculation?

There are some models that include speculation *relaxed memory models*:

- *The Java Memory Model*
  Manson, Pugh and Adve, 2005.
- *Generative Operational Semantics for Relaxed Memory Models*
  Jagadeesan, Pitcher and Riely, 2010.
- *A promising semantics for relaxed-memory concurrency*
  Kang, Hur, Lahav, Vafeiadis and Dreyer, 2017.

Question: is there a simple model similar to those of relaxed memory, that can model speculation?

Information flow attacks on speculation

Speculation happens in many places:

- *Speculation in hardware* (branch prediction,. . .)
- *Transactions* (transactional memory,. . .)
- *Relaxed memory* (compiler optimizations,. . .)

Information flow attacks on speculation

Speculation happens in many places:

- *Speculation in hardware* (branch prediction,. . .)
  Attacked by Spectre (Kocher *et al.* 2019).
- *Transactions* (transactional memory,. . .)
  Attacked by Prime+Abort (Disselkoen *et al.* 2017).
- *Relaxed memory* (compiler optimizations,. . .)
  No known attacks.

Question: are there information flow attacks against compiler optimizations?

Contributions

- A simple compositional model.
- Attacks (including a new attack on relaxed memory).
- Experiments (testing practicality of new attacks).

Pomsets

C11-style models are based on *events*
with *labels* (e.g. $(\mathsf{R}\,x\,3)$ or $(\mathsf{W}\,x\,3)$)
and *relations* (e.g. happens-before or reads-from).

Pomsets

C11-style models are based on *events*
with *labels* (e.g. $(\mathsf{R}\,x\,3)$ or $(\mathsf{W}\,x\,3)$)
and *relations* (e.g. happens-before or reads-from).

Simplest such is *partially ordered multisets* (Gisher, 1988).

Only one relation, a partial order modeling dependency

Pomsets

C11-style models are based on *events*
with *labels* (e.g. $(\mathsf{R}\,x\,3)$ or $(\mathsf{W}\,x\,3)$)
and *relations* (e.g. happens-before or reads-from).

Simplest such is *partially ordered multisets* (Gisher, 1988).

Only one relation, a partial order modeling dependency, e.g.

$$\begin{tikzcd} \boxed{\mathsf{R}\,x\,1} \arrow[rr, bend left] & \boxed{\mathsf{W}\,y\,1} & \boxed{\mathsf{W}\,z\,2} \end{tikzcd}$$

is an execution of $(r := x\,;\; y := 1\,;\; z := r + 1)$.

Pomsets

C11-style models are based on *events*
with *labels* (e.g. $(\mathsf{R}\,x\,3)$ or $(\mathsf{W}\,x\,3)$)
and *relations* (e.g. happens-before or reads-from).

Simplest such is *partially ordered multisets* (Gisher, 1988).

Only one relation, a partial order modeling dependency, e.g.

$$\begin{tikzcd} \mathsf{R}\,x\,1 \arrow[r] \arrow[dr] & \mathsf{W}\,y\,1 \\ & \cancel{\mathsf{W}\,y\,2} \end{tikzcd}$$

is an execution of $(\texttt{if}\,(x)\,\{\,y := 1\,\}\,\texttt{else}\,\{\,y := 2\,\})$.

Compositional pomset model

First off, straight-line code.

Compositional pomset model

First off, straight-line code.

New idea: put preconditions on events

Compositional pomset model

First off, straight-line code.

New idea: put preconditions on events, e.g.

$$\boxed{r = 1 \mid \mathsf{W}\, z\, 2}$$

is an execution of ($z := r + 1$).

Compositional pomset model

First off, straight-line code.

New idea: put preconditions on events, e.g.

$$\boxed{\mathsf{W}\,y\,1} \quad \boxed{r = 1 \mid \mathsf{W}\,z\,2}$$

is an execution of ($y := 1;\ z := r + 1$).

Note: no dependency because r does not depend on $y := 1$.

Compositional pomset model

First off, straight-line code.

New idea: put preconditions on events, e.g.

$$\boxed{\mathsf{R}\,x\,1} \quad \boxed{\mathsf{W}\,y\,1} \quad \boxed{1 = 1 \mid \mathsf{W}\,z\,2}$$

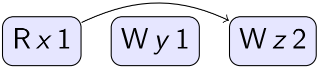
is an execution of $(r := x\,;\, y := 1\,;\, z := r + 1)$.

Note: dependency because r depends on $r := x$.
Also note: performing a substitution $[1/r]$.

Compositional pomset model

First off, straight-line code.

New idea: put preconditions on events, e.g.



is an execution of $(r := x\,;\ y := 1\,;\ z := r + 1)$.

Visualize: elide tautologies

Compositional pomset model

Next, conditionals.

Compositional pomset model

Next, conditionals.

New idea: an execution of $\texttt{if}\, M \,\{\, C \,\}\, \texttt{else} \,\{\, D \,\}$ comes from an execution of C *and* an execution of D

Compositional pomset model

Next, conditionals.

New idea: an execution of $\texttt{if}\, M \,\{\, C \,\}\, \texttt{else} \,\{\, D \,\}$
comes from an execution of C *and* an execution of D, e.g.

$$\boxed{r \neq 0 \mid \mathsf{W}\, y\, 1}$$

is an execution of ($y := 1$)
when $r \neq 0$

Compositional pomset model

Next, conditionals.

New idea: an execution of $\texttt{if}\, M \,\{\, C \,\}\, \texttt{else} \,\{\, D \,\}$
comes from an execution of C *and* an execution of D, e.g.

$$\boxed{r = 0 \mid \mathsf{W}\, y\, 2}$$

is an execution of ($y := 2$)
when $r = 0$

Compositional pomset model

Next, conditionals.

New idea: an execution of $\texttt{if}\, M\, \{\, C\, \}\, \texttt{else}\, \{\, D\, \}$
comes from an execution of C *and* an execution of D, e.g.

$$\boxed{r \neq 0 \mid \mathsf{W}\, y\, 1}$$

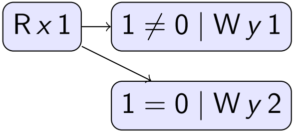
$$\boxed{r = 0 \mid \mathsf{W}\, y\, 2}$$

is an execution of ($\texttt{if}\,(r)\,\{\, y := 1\,\}\,\texttt{else}\,\{\, y := 2\,\}$)

Compositional pomset model

Next, conditionals.

New idea: an execution of $\texttt{if}\,M\,\{\,C\,\}\,\texttt{else}\,\{\,D\,\}$
comes from an execution of C *and* an execution of D, e.g.

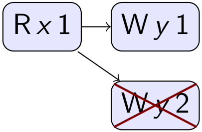


is an execution of $(r := x\,;\, \texttt{if}\,(r)\,\{\,y := 1\,\}\,\texttt{else}\,\{\,y := 2\,\})$

Compositional pomset model

Next, conditionals.

New idea: an execution of $\texttt{if}\, M \,\{\, C \,\}\, \texttt{else} \,\{\, D \,\}$
comes from an execution of C *and* an execution of D, e.g.



is an execution of $(r := x\,;\, \texttt{if}\,(r)\,\{\, y := 1 \,\}\, \texttt{else} \,\{\, y := 2 \,\})$

Visualize: elide tautologies and cross out unsatisfiables

Compositional pomset model

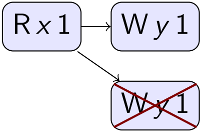
But. . .

Compositional pomset model

But. . . any execution of C should be an execution of $\texttt{if}\, M \,\{\, C \,\}\, \texttt{else} \,\{\, C \,\}$

Compositional pomset model

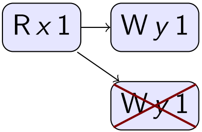
But... any execution of C should be
an execution of $\texttt{if}\, M \,\{\, C \,\}\, \texttt{else} \,\{\, C \,\}$, e.g.



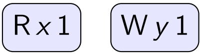
is an execution of $(\texttt{if}\, x \,\{\, y := 1 \,\}\, \texttt{else} \,\{\, y := 1 \,\})$

Compositional pomset model

But. . . any execution of C should be
an execution of $\texttt{if}\,M\,\{\,C\,\}\,\texttt{else}\,\{\,C\,\}$, e.g.

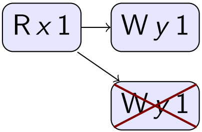


is an execution of $(\texttt{if}\,x\,\{\,y := 1\,\}\,\texttt{else}\,\{\,y := 1\,\})$, but so is

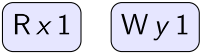


Compositional pomset model

But… any execution of C should be
an execution of $\texttt{if}\, M \,\{\, C \,\}\, \texttt{else} \,\{\, C \,\}$, e.g.



is an execution of $(\texttt{if}\, x \,\{\, y := 1 \,\}\, \texttt{else} \,\{\, y := 1 \,\})$, but so is



New idea: events from different branches can merge.

Compositional pomset model

Lastly, concurrency.

Compositional pomset model

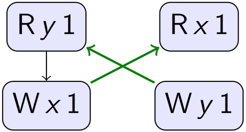
Lastly, concurrency.

Old idea: match reads with matching writes (à la C11)

Compositional pomset model

Lastly, concurrency.

Old idea: match reads with matching writes (à la C11), e.g.



is an execution of $(x := y \ || \ r := x ; y := 1)$.

Compositional pomset model

Glossed over some details:

- 3-valued pomsets for negative constraints $d \not\leq e$,
- sanity conditions on reads-from,
- precise rules for dependency,
- variable declaration,
- ...

All in the paper!

Information flow example

Imagine a `SECRET`, protected by a run-time security check:

$$\texttt{if canRead(SECRET)}\,\{\ldots\text{use }\texttt{SECRET}\ldots\}\,\texttt{else}\,\{\ldots\}$$

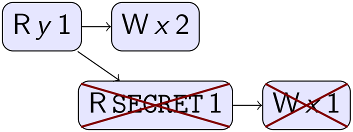
For attacker code `canRead(SECRET)` is always false

Information flow example

Imagine a $\texttt{SECRET}$, protected by a run-time security check:

$$\texttt{if}\ \texttt{canRead}(\texttt{SECRET})\ \{\ \dots\ \text{use}\ \texttt{SECRET}\ \dots\ \}\ \texttt{else}\ \{\ \dots\ \}$$

For attacker code $\texttt{canRead}(\texttt{SECRET})$ is always false, e.g.



is an execution of

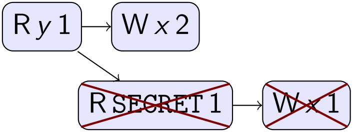
$\texttt{if}\ y\ \{\ \texttt{if}\ \texttt{canRead}(\texttt{SECRET})\ \{\ x := \texttt{SECRET}\ \}\ \texttt{else}\ \{\ x := 2\ \}\ \}$.

Information flow example

Imagine a `SECRET`, protected by a run-time security check:

$$\texttt{if}\ \texttt{canRead}(\texttt{SECRET})\ \{\ \ldots \text{use}\ \texttt{SECRET} \ldots\ \}\ \texttt{else}\ \{\ \ldots\ \}$$

For attacker code `canRead(SECRET)` is always false, e.g.



is an execution of
$\texttt{if}\ y\ \{\ \texttt{if}\ \texttt{canRead}(\texttt{SECRET})\ \{\ x := \texttt{SECRET}\ \}\ \texttt{else}\ \{\ x := 2\ \}\ \}$.

Attacker goal: learn if `SECRET` is 0 or 1.

Modeling Spectre attack

Spectre uses cache timing to discover if a memory location has been touched.

Modeling Spectre attack

Spectre uses cache timing to discover if a memory location has been touched.

Glossing over a lot of details, this is

$$\texttt{if}\,\texttt{touched}\,(x)\,\{\,\cdots\}\,\texttt{else}\,\{\,\cdots\}$$

Modeling Spectre attack

Spectre uses cache timing to discover if a memory location has been touched.

Glossing over a lot of details, this is

$$\mathtt{if}\,\mathtt{touched}\,(x)\,\{\cdots\}\,\mathtt{else}\,\{\cdots\}$$

Modeled with a new action $(\mathsf{T}\,x)$

Modeling Spectre attack

Spectre uses cache timing to discover if a memory location has been touched.

Glossing over a lot of details, this is

$$\texttt{if}\,\texttt{touched}\,(x)\,\{\,\cdots\,\}\,\texttt{else}\,\{\,\cdots\,\}$$

Modeled with a new action $(\mathsf{T}\,x)$, e.g.

$$\boxed{\mathsf{T}\,x} \longrightarrow \boxed{\mathsf{W}\,y\,1}$$

is an execution of $\texttt{if}\,\texttt{touched}\,(x)\,\{\,y := 1\,\}$.

Modeling Spectre attack

Spectre uses cache timing to discover if a memory location has been touched.

Glossing over a lot of details, this is

$$\texttt{if}\,\texttt{touched}\,(x)\,\{\cdots\}\,\texttt{else}\,\{\cdots\}$$

Modeled with a new action $(\mathsf{T}\,x)$, e.g.

$$\boxed{\mathsf{T}\,x} \longrightarrow \boxed{\mathsf{W}\,y\,1}$$

is an execution of $\texttt{if}\,\texttt{touched}\,(x)\,\{\,y := 1\,\}$.

Require that if there is an event labeled $(\mathsf{T}\,x)$ then there must be an event labeled $(\mathsf{R}\,x\,v)$ or $(\mathsf{W}\,x\,v)$.

Modeling Spectre attack

A very simplified Spectre attack:

$$\begin{array}{l}\texttt{if}\,\texttt{canRead}(\texttt{SECRET})\,\{\,a[\texttt{SECRET}] := 1\,\}\\ \texttt{else}\,\texttt{if}\,\texttt{touched}\,(a[0])\,\{\,x := 0\,\}\\ \texttt{else}\,\texttt{if}\,\texttt{touched}\,(a[1])\,\{\,x := 1\,\}\end{array}$$

Modeling Spectre attack

A very simplified Spectre attack:

$$
\begin{array}{l}
\mathtt{if}\ \mathtt{canRead}(\mathtt{SECRET})\ \{\ a[\mathtt{SECRET}] := 1\ \} \\
\mathtt{else}\ \mathtt{if}\ \mathtt{touched}\ (a[0])\ \{\ x := 0\ \} \\
\mathtt{else}\ \mathtt{if}\ \mathtt{touched}\ (a[1])\ \{\ x := 1\ \}
\end{array}
$$

e.g. with execution

$$\boxed{\xcancel{\mathsf{R}\ \mathtt{SECRET}\ 1}} \rightarrow \boxed{\xcancel{\mathsf{W}\ a[1]\ 1}} \rightarrow \boxed{\mathsf{T}\ a[1]} \rightarrow \boxed{\mathsf{W}\ x\ 1}$$

Information flow from $\mathtt{SECRET}$ to x.

Modeling Prime+Abort attack

Prime+Abort is an information flow attack on Intel's transactional memory. So first model transactions

Modeling Prime+Abort attack

Prime+Abort is an information flow attack on Intel's transactional memory. So first model transactions, e.g.

$$\boxed{\mathsf{B}} \longrightarrow \boxed{\mathsf{R}\,x\,1} \longrightarrow \boxed{\mathsf{W}\,x\,2} \longrightarrow \boxed{\mathsf{C}}$$

and

$$\boxed{\mathsf{B}} \longrightarrow \boxed{\xcancel{\mathsf{R}\,x\,1}} \longrightarrow \boxed{\xcancel{\mathsf{W}\,x\,2}} \longrightarrow \boxed{\xcancel{\mathsf{C}}}$$

are executions of `begin`; $x := x + 1$; `end`

Modeling Prime+Abort attack

Prime+Abort is an information flow attack on Intel's transactional memory. So first model transactions, e.g.

$$\mathsf{B} \longrightarrow \mathsf{R}\,x\,1 \longrightarrow \mathsf{W}\,x\,2 \longrightarrow \mathsf{C}$$

and

$$\mathsf{B} \longrightarrow \cancel{\mathsf{R}\,x\,1} \longrightarrow \cancel{\mathsf{W}\,x\,2} \longrightarrow \cancel{\mathsf{C}}$$

are executions of `begin`; $x := x + 1$; `end`, but *not*

$$\mathsf{B} \longrightarrow \mathsf{R}\,x\,1 \longrightarrow \cancel{\mathsf{W}\,x\,2} \longrightarrow \cancel{\mathsf{C}}$$

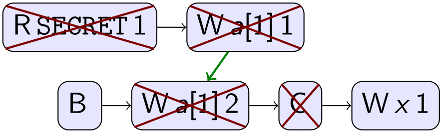
Modeling Prime+Abort attack

Transactions are fine, but not if we add a reason for an abort.

If the attacker knows an aborted transaction does so because of a read/write or write/write conflict, then in

$$\begin{array}{l}\mathtt{if}\ \mathtt{canRead}(\mathtt{SECRET})\ \{\ a[\mathtt{SECRET}] := 1\ \}\ \|\\ \mathtt{begin};\ a[1] := 2;\ \mathtt{loop};\ \mathtt{end};\ x := 1\end{array}$$

the transaction aborts only when SECRET is 1.



Information flow from SECRET to x.

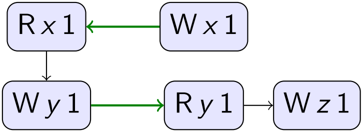
Craig Disselkoen, Radha Jagadeesan, Alan Jeffrey, James Riely

New store reordering attack

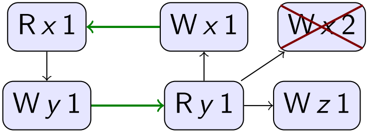
An attack on relaxed memory, *discovered from this model*.

$$y := x \;||\; \texttt{if}\,(y == 0)\,\{\, x := 1 \,\}$$
$$\texttt{else}\;\texttt{if}\,(\texttt{canRead}(\texttt{SECRET}))\,\{\, x := \texttt{SECRET} \,\}$$
$$\texttt{else}\,\{\, x := 1;\, z := 1 \,\}$$

If $\texttt{SECRET}$ is 1, there is an execution:



If $\texttt{SECRET}$ is 2, there is no execution:



New dead store elimination attack

Another attack *discovered from this model*.

$$
\begin{array}{l}
y := x \;||\; x := 1; \\
\qquad\qquad \texttt{if}\,(\texttt{canRead}(\texttt{SECRET}))\,\{\,\texttt{if}\,(\texttt{SECRET})\,\{\,x := 2\,\}\,\} \\
\qquad\qquad \texttt{else}\,\{\,x := 2\,\}
\end{array}
$$

If `SECRET` is 1, there is an execution:

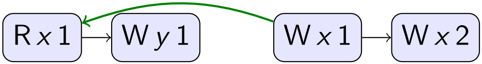
$$
\begin{tikzcd}
\boxed{\mathrm{R}\,x\,1} \arrow[r] & \boxed{\mathrm{W}\,y\,1} & \boxed{\mathrm{W}\,x\,1} \arrow[r] \arrow[ll, bend right] & \boxed{\mathrm{W}\,x\,2}
\end{tikzcd}
$$

New dead store elimination attack

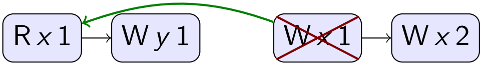
Another attack *discovered from this model*.

```
y := x || x := 1;
         if (canRead(SECRET)) { if (SECRET) { x := 2 } }
         else { x := 2 }
```

If SECRET is 1, there is an execution:



If dead store elimination is performed, there is *no* execution:



Implementing the new attacks

Spectre and Prime+Abort are implemented.
What about the attacks on compiler optimizations?

Implementing the new attacks

Spectre and Prime+Abort are implemented.
What about the attacks on compiler optimizations?

Yes

Implementing the new attacks

Spectre and Prime+Abort are implemented.
What about the attacks on compiler optimizations?

Yes, under unrealistic assumptions:

- `SECRET` is a constant known at compile-time,
- `canRead(SECRET)` is a run-time check.

Implementing load/store reordering

x86 assembly generated by gcc for the main thread of a variant of the load-store reordering attack:

If SECRET is 0:

```
  mov SECRET(%rip), %eax
  mov $1, x(%rip)
  test %eax, %eax
  je label1
  mov $0, x(%rip)
label1:
  mov y(%rip), %eax
  test %eax, %eax
  sete %eax
```

Writes x then reads y, so can read 1

If SECRET is 1:

```
  mov SECRET(%rip), %eax
  mov y(%rip), %eax
  mov $1, x(%rip)
  test %eax, %eax
  sete %eax
```

Reads y then writes x, so cannot read 1

A forwarding thread copies x to y.

Implementing load/store reordering

To make this attack more likely, introduce
a small delay between write of x and read of y,
increases probability of round trip.

Experimentally gcc will reorder load/store across 30 straight-line instructions.

Repeat attack to leak multiple bits,
and increase probability of success.

Attack is 99.9% accurate at 100Kbps.

Implementing dead store elimination attack

DSE attack is similar.

Works against clang as well as gcc.

Attack is 99.9% accurate at 400Kbps (clang), 2Mbps (gcc).

Also in the paper

Details of the model, semantics, etc.

Temporal logic for proving invariants (e.g. no thin-air read).

More examples.

Contributions

A model of program execution that includes speculation.

Examples including existing information flow attacks on branch prediction and transactional memory, and new attacks on optimizing compilers.

Experimental evidence about how practical it is to mount the new class of attacks.

A temporal logic which supports compositional proof.

`https://github.com/chicago-relaxed-memory/spec-eval`