Run your Research
On the Effectiveness of Lightweight Mechanization

C Klein, J Clements, C Dimoulas, C Eastlund, M Felleisen, M Flatt,
J A McCarthy, J Rafkind, S Tobin-Hochstadt, R B Findler
One day, Koala decided to build an ftp server

ftp> user anonymous
331 Guest login ok
Password:
230-Welcome to λ.com

Moral: bugs are everywhere
The Koala, the Orangutan, and the Walrus

and made the unfortunate choice to use the programming language C.

```c
int main () {
    if (!(q = 0))
        *((int*)p)=12;
}
```
The Koala, the Orangutan, and the Walrus

We must not be surprised by this choice, however, as C is well-known to be a programming language that is effective for building systems software.
The Koala, the Orangutan, and the Walrus

After a few months of effort, Koala produced a functioning server that was rapidly adopted across the internet and widely used.

```c
int main () {
    if (!(q = 0))
        *((int*)p)=12;
}
```
The Koala, the Orangutan, and the Walrus

One day, Orangutan decided to apply a new, automated testing technique to Koala’s ftp server and, sure enough, found multiple bugs —

```c
int main () {
    if (!(q = 0))
        *((int*)p)=12;
}
```
The Koala, the Orangutan, and the Walrus

```c
int main () {
    if (!q)
        *((int*)p)=12;
    p = 0 ∨ *p = *q
}
```

unsurprising for software of that complexity implemented in a programming language like C. After all, C is designed for performance and provides no help to maintain invariants of data structures or to detect errors early, when they are easy to fix.
The Koala, the Orangutan, and the Walrus

\[ \Gamma \vdash (\lambda x:\text{\tau}_2.e) : \text{\tau}_1 \rightarrow \text{\tau}_2 \]

So, Orangutan decided to write a paper that explained the mathematical techniques it used to uncover the bugs and made the unfortunate choice to use the programming language LaTeX.
The Koala, the Orangutan, and the Walrus

We must not be surprised by this choice, however, as LaTeX is well-known to be a programming language that is effective for typesetting mathematical formulas.
The Koala, the Orangutan, and the Walrus

After a few months of effort, Orangutan produced a paper extolling the virtues of its new techniques, and the ideas were adopted across the software engineering community and the paper was widely cited.
One day, Walrus decided to apply a new, lightweight mechanized metatheory technique to Orangutan’s paper and, sure enough, found multiple bugs —
The Koala, the Orangutan, and the Walrus

\[
\Gamma \vdash \lambda x : \tau_{2.e} : \tau_1 \rightarrow \tau_2
\]

unsurprising for a piece of mathematics of that complexity implemented in a programming language like LaTeX. After all, LaTeX is designed for beautiful output and provides no help to check invariants of mathematical formulas or to run examples to ensure they illustrate the intended points.
Moral: bugs are everywhere
A niche for mechanized metatheory:

• lightweight: high level of expressiveness (think scripting language)

• supports the entire semantics lifecycle:
The Semantics Lifecycle

- Prototype model
- Robust model
- Write-up

15
The Semantics Lifecycle

Prototype model

Write-up

Robust model

misrenamed non-terminal
The Semantics Lifecycle

Prototype model

Robust model

Write-up

misrenamed non-terminal

forgot typing rule
The Semantics Lifecycle

- misrenamed non-terminal
- forgot typing rule
- lost a case in a helper function

Prototype model

Write-up

Robust model
The Semantics Lifecycle

misrenamed non-terminal

forgot typing rule

lost a case in a helper function

added a case to wrong fn

Prototype model

Write-up

Robust model
misrenamed non-terminal
forgot typing rule
lost a case in a helper function
added a case to wrong fn

The Semantics Lifecycle

Prototype model

Write-up

Robust model

swappped args
The Semantics Lifecycle

- misrenamed non-terminal
- forgot typing rule
- lost a case in a helper function
- added a case to wrong fn
- swappped args
- misused the inductive hyp.

Prototype model

Write-up

Robust model
The Semantics Lifecycle

misrenamed non-terminal

forgot typing rule

lost a case in a helper function

added a case to wrong fn

Prototype model

Write-up

Robust model

swapped args

misused the inductive hyp.

didn’t recheck a lemma
The Semantics Lifecycle

misrenamed non-terminal
forgot typing rule
lost a case in a helper function
added a case to wrong fn

Prototype model

swapped args
misused the inductive hyp.
didn’t recheck a lemma
transcribed math wrong

Robust model

Write-up
The Semantics Lifecycle

misrenamed non-terminal
forgot typing rule
lost a case in a helper function
added a case to wrong fn

Prototype model

Write-up

Robust model

swapped args
misused the inductive hyp.
didn’t recheck a lemma
transcribed math wrong
forgot to recheck example
Redex

our tool designed to fill this niche
Our study:

• Can random testing find bugs in an existing, well-tested Redex model?

• Can Redex find bugs in published papers?
Our study:

• Can random testing find bugs in an existing, well-tested Redex model?  
  Yes

• Can Redex find bugs in published papers?  
  Yes
10 papers in Redex
9 ICFP ’09 papers
8 written by others
2 mechanically verified
10 papers with errors

10 papers in Redex
9 ICFP ’09 papers
8 written by others
2 mechanically verified
Your papers have errors too.
Copy & Paste Typesetting Error:

\[
\begin{align*}
\text{sf} & \xrightarrow{\Delta t_{\varphi_1}} (sf_\varphi, \text{NoEvent} :: bs) \\
\text{switch sf x} & \xrightarrow{\Delta t_{\varphi_1}} (\text{switch}_\varphi \text{sf}_\varphi x, bs) \\
\Phi_1\text{-SW-NOEV} \\
\text{sfs} & \xrightarrow{\Delta t_{\varphi_1}} (sfs_\varphi, \text{Event e} :: bs) \\
& \quad f e \mapsto \text{sfr} \\
& \quad \text{sfr} \xrightarrow{0} (sfr_\varphi, bsr) \\
\text{switch sfs x} & \xrightarrow{\Delta t_{\varphi_1}} (sfr_\varphi, bsr) \\
\Phi_1\text{-SW-EV} \\
\text{sf} & \xrightarrow{\Delta t_{\varphi_1}} (sf_\varphi, \text{NoEvent} :: bs) \\
\text{dswitch sf x} & \xrightarrow{\Delta t_{\varphi_1}} (\text{dswitch}_\varphi \text{sf}_\varphi x, bs) \\
\Phi_1\text{-DSW-NOEV} \\
\text{sfs} & \xrightarrow{\Delta t_{\varphi_1}} (sfs_\varphi, \text{Event e} :: bs) \\
& \quad f e \mapsto \text{sfr} \\
& \quad \text{sfr} \xrightarrow{0} (sfr_\varphi, bsr) \\
\text{switch sfs x} & \xrightarrow{\Delta t_{\varphi_1}} (sfr_\varphi, bss) \\
\Phi_1\text{-DSW-EV}
\end{align*}
\]
Copy & Paste Typesetting Error:

\[
\frac{s f \xrightarrow{\Delta L} \phi_1 (s f_\phi, \text{NoEvent} :: \text{bs})}{\text{switch } s f \xrightarrow{\Delta L} \phi_1 (\text{switch}_\phi s f_\phi f, \text{bs})} \Phi_{1\text{-SW-NOEV}}
\]

\[
\frac{s f s \xrightarrow{\Delta L} \phi_1 (s f s_\phi, \text{Event } e :: \text{bs}s)}{f e \leftrightarrow s f r \quad s f r \xrightarrow{0} \phi_1 (s f r_\phi, \text{bsr})} \Phi_{1\text{-SW-EV}}
\]

\[
\frac{\text{switch } s f s \xrightarrow{\Delta L} \phi_1 (s f r_\phi, \text{bsr})}{\Phi_{1\text{-SW-EV}}}
\]

\[
\frac{s f \xrightarrow{\Delta L} \phi_1 (s f_\phi, \text{NoEvent} :: \text{bs})}{d \text{switch } s f \xrightarrow{\Delta L} \phi_1 (d \text{switch}_\phi s f_\phi f, \text{bs})} \Phi_{1\text{-DSW-NOEV}}
\]

\[
\frac{s f s \xrightarrow{\Delta L} \phi_1 (s f s_\phi, \text{Event } e :: \text{bs}s)}{f e \leftrightarrow s f r \quad s f r \xrightarrow{0} \phi_1 (s f r_\phi, \text{bsr})} \Phi_{1\text{-DSW-EV}}
\]

\[
\frac{\text{switch } s f s \xrightarrow{\Delta L} \phi_1 (s f r_\phi, \text{bs})}{\Phi_{1\text{-DSW-EV}}}
\]
Typesetting should be automatic
Erroneous Example:

\[ \Sigma; \cdot \vdash (\lambda y : \text{Lazy Int.} \ y + 1)(\lambda x : \text{Unit.} \ e) \sim\]
\[ (\lambda y : \text{Lazy Int.} \ (\text{force[Int] } y) + 1) \]
\[ (\text{lzy } \lambda x : \text{Unit.} \ e) : \text{Int} \]
Erroneous Example:

\[ \Sigma; \vdash (\lambda y : \text{Lazy Int. } y + 1)(\lambda x : \text{Unit. } e) \sim \\
(\lambda y : \text{Lazy Int. } \text{force[Int] } y + 1) \\
(\text{lazy } \lambda x : \text{Unit. } e) : \text{Int} \]
Erroneous Example:

\[ \Sigma; \cdot \vdash (\lambda y : \text{Lazy Int.} \ y + 1)(\lambda x : \text{Unit.} \ e) \leadsto (\lambda y : \text{Lazy Int.} \ (\text{force[Int]} \ y) + 1) \]
\[ (\text{lazy[Int]} \ \lambda x : \text{Unit.} \ m) : \text{Int} \]

where \( \Sigma; \{x : \text{Unit}\} \vdash e \leadsto m \)
Erroneous Example:

\[ \Sigma; \cdot \vdash (\lambda y : \text{Lazy Int.} \ y + 1)(\lambda x : \text{Unit.} \ e) \sim \]
\[ (\lambda y : \text{Lazy Int.} \ (\text{force}[\text{Int}] \ y) + 1) \]
\[ (\text{lazy}[\text{Int}] \ \lambda x : \text{Unit.} \ m) : \text{Int} \]

where \[ \Sigma; \{x : \text{Unit}\} \vdash e \sim m \]

Examples can be tested
Unexpected Behavior:

\[ \text{select}(c, \overline{c}) \]
Unexpected Behavior:

\[ \text{compile} \quad \text{select}(c, \overline{c}) \]

\[ \bigcirc_c \mid \text{select}(c, \overline{c}) \]
**Unexpected Behavior:**

\[
\text{compile} \leftarrow \odot \triangleright \text{select}(c, \overline{c}) \quad \text{– stuck}
\]

\[
\odot_c \mid \text{select}(c, \overline{c}) \quad \text{– loops forever}
\]

Deadlock in source but busy waiting in target
**Unexpected Behavior:**

\[
\text{select}(c, \overline{c}) \quad - \quad \text{stuck}
\]

\[
\text{compile} \quad \Leftrightarrow \quad \circ_c \mid \text{select}(c, \overline{c}) \quad - \quad \text{loops forever}
\]

Deadlock in source but busy waiting in target

*Found this by playing with examples*
False Theorem:

If a term reduces with a memo store, then the program without the memo store reduces the same way.
False Theorem:

If a term reduces with a memo store, then the program without the memo store reduces the same way.

Counterexample:
If $\sigma = \{(\delta, 1) \rightarrow 2\}$ then

$$(\lambda_{\delta} \, x. \, x) \, 1, \, \sigma \Rightarrow^{\ast} 2, \, \sigma,$$

but $$(\lambda_{\delta} \, x. \, x) \, 1 \rightarrow 1$$

Not a fly-by-night proof; 12 typeset pages in a dissertation chapter.
**False Theorem:**

If a term reduces with a memo store, then the program without the memo store reduces the same way.

Counterexample:

If $\sigma = \{(\delta,1) \rightarrow 2\}$ then

$$\left(\lambda_\delta x. x\right) 1, \sigma \Rightarrow^* 2, \sigma,$$

but $\left(\lambda_\delta x. x\right) 1 \mapsto 1$

Random testing easily finds this.
Recap:

• Automatic typesetting
• Unit Testing
• Exploring Examples
• Random testing
\[ p ::= (e \ldots) \]
\[ e ::= (e e \ldots) \]
\[ | (\lambda \ (x:t \ldots) \ e) \]
\[ | x \]
\[ | (+ e \ldots) \]
\[ | \text{number} \]
\[ | (\text{amb} \ e \ldots) \]
\[ t ::= (\to \ t \ \ldots \ t) | \text{num} \]

\[ P ::= (e \ldots \ E \ e \ldots) \]
\[ E ::= (v \ldots \ E \ e \ldots) \]
\[ | (+ v \ldots \ E \ e \ldots) \]
\[ | [] \]
\[ v ::= (\lambda \ (x:t \ldots) \ e) \]
\[ | \text{number} \]
\[ \Gamma ::= \cdot | (x : t \ \Gamma) \]

\[ P[(((\lambda \ (x:t \ldots) \ e) \ v \ldots))] \quad [\beta v] \]
\[ \quad \to P[e\{x:=v \ldots\}] \]
\[ P[(+ \ \text{number}_1 \ldots)] \quad [+ ] \]
\[ \quad \to P[\Sigma[[\text{number}_1, \ldots]]] \]
\[ (e_1 \ldots \ E[(\text{amb} \ e_2 \ldots)] \ e_3 \ldots) \quad [\text{amb}] \]
\[ \quad \to (e_1 \ldots \ E[e_2] \ldots \ e_3 \ldots) \]

\[
\Gamma \vdash e_1 : (\to \ t_2 \ldots \ t_3) \quad \Gamma \vdash e_2 : t_2 \ldots \\
\frac{\Gamma \vdash (e_1 \ e_2 \ldots) : t_3}{\Gamma \vdash (e_1 \ e_2 \ldots) : t_3} \\
(x_1 : t_1 \ \Gamma) \vdash (\lambda \ (x_2:t_2 \ldots) \ e) : (\to \ t_2 \ldots t) \\
\frac{\Gamma \vdash (\lambda \ (x_2:t_2 \ldots) \ e) : (\to \ t_2 \ldots t)}{\Gamma \vdash (\lambda \ (x_1:t_1 \ x_2:t_2 \ldots) \ e) : (\to \ t_1 \ t_2 \ldots t)} \\
\]

\[
\frac{\Gamma \vdash e : t}{\Gamma \vdash (\lambda \ () \ e) : (\to \ t)} \\
\frac{\Gamma \vdash (\lambda \ () \ e) : (\to \ t)}{(x : t \ \Gamma) \vdash x : t} \\
\frac{(x_1 : t_1) \ x_1 \neq x_2}{(x_2 : t_2 \ \Gamma) \vdash x_1 : t_1} \\
\frac{\Gamma \vdash e : \text{num} \ldots}{\Gamma \vdash (+ e \ldots) : \text{num}} \\
\frac{\Gamma \vdash (+ e \ldots) : \text{num}}{\Gamma \vdash \text{number} : \text{num}} \\
\frac{\Gamma \vdash e : \text{num} \ldots}{\Gamma \vdash (\text{amb} \ e \ldots) : \text{num}} \]
\[ p ::= (e \ldots) \]

\[ e ::= (e\ e\ \ldots) \]
\[ \mid (\lambda \ (x:\ t\ \ldots)\ e) \]
\[ \mid x \]
\[ \mid (+\ e\ \ldots) \]
\[ \mid \text{number} \]
\[ \mid (\text{amb}\ e\ \ldots) \]

\[ t ::= (\rightarrow\ t\ \ldots\ t) \]

\[ P ::= (e\ \ldots\ E\ e\ \ldots) \]
\[ E ::= (v\ \ldots\ E\ e\ \ldots) \]
\[ \mid (+\ v\ \ldots\ E\ e\ \ldots) \]
\[ \mid [] \]
\[ v ::= (\lambda\ (x:\ t\ \ldots)\ e) \]
\[ \mid \text{number} \]

\[ \Gamma ::= \bullet\ \mid (x:\ t\ \Gamma) \]

\[ p ::= (e\ \ldots) \]

\[ \mid (\text{amb}\ e\ \ldots) \]

\[ \frac{\Gamma \vdash e : \text{num} \ldots}{\Gamma \vdash (\text{amb}\ e\ \ldots) : \text{num}} \]

\[ (e_1\ \ldots\ E[(\text{amb}\ e_2\ \ldots)]\ e_3\ \ldots)\ [\text{amb}] \]
\[ \rightarrow (e_1\ \ldots\ E[e_2]\ \ldots\ e_3\ \ldots) \]

\[ (e_1\ \ldots\ E[(\text{amb}\ e_2\ \ldots)]\ e_3\ \ldots)\ [\text{amb}] \]
\[ \rightarrow (e_1\ \ldots\ E[e_2]\ \ldots\ e_3\ \ldots) \]
Recap:

✓ Automatic typesetting
✓ Unit Testing
✓ Exploring Examples
✓ Random testing
Takeaways:

• Nobody will produce error-free papers
• Errors introduce friction into our communication
• Redex can help reduce the errors — with about as much effort as LaTeX requires
Thank you.