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

The Koala, the Orangutan, and the Walrus

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

One day, Koala decided to build an ftp server

The Koala, the Orangutan, and the Walrus

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

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

The Koala, the Orangutan, and the Walrus

```
{
0))
)p)=12;
```

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

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

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

The Koala, the Orangutan, and the Walrus

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

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

The Koala, the Orangutan, and the Walrus

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

$p == 0 \vee *p == *q$

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

```
\[\Gamma\ \vdash\
  (\lambda x:\tau_2.e)
   : \tau_1\rightarrow
     \tau_2 \]
```

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

```
\ \vdash\
da x:\tau_2.e)
u_1\rightarrow
u_2 \]
```

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

```
\[\Gamma\ \vdash\
  (\lambda x:\tau_2.e)
   : \tau_1\rightarrow
     \tau_2 \]
```

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.

The Koala, the Orangutan, and the Walrus

```
\[\Gamma\ \vdash\
  (\lambda x:\tau_2.e)
   : \tau_1\rightarrow
     \tau_2 \]
```

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

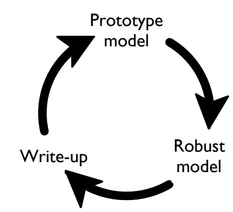
```
\[\Gamma\ \vdash\
  (\lambda x:\tau_2.e)
   : \tau_1\rightarrow
     \tau_2 \]
```

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.

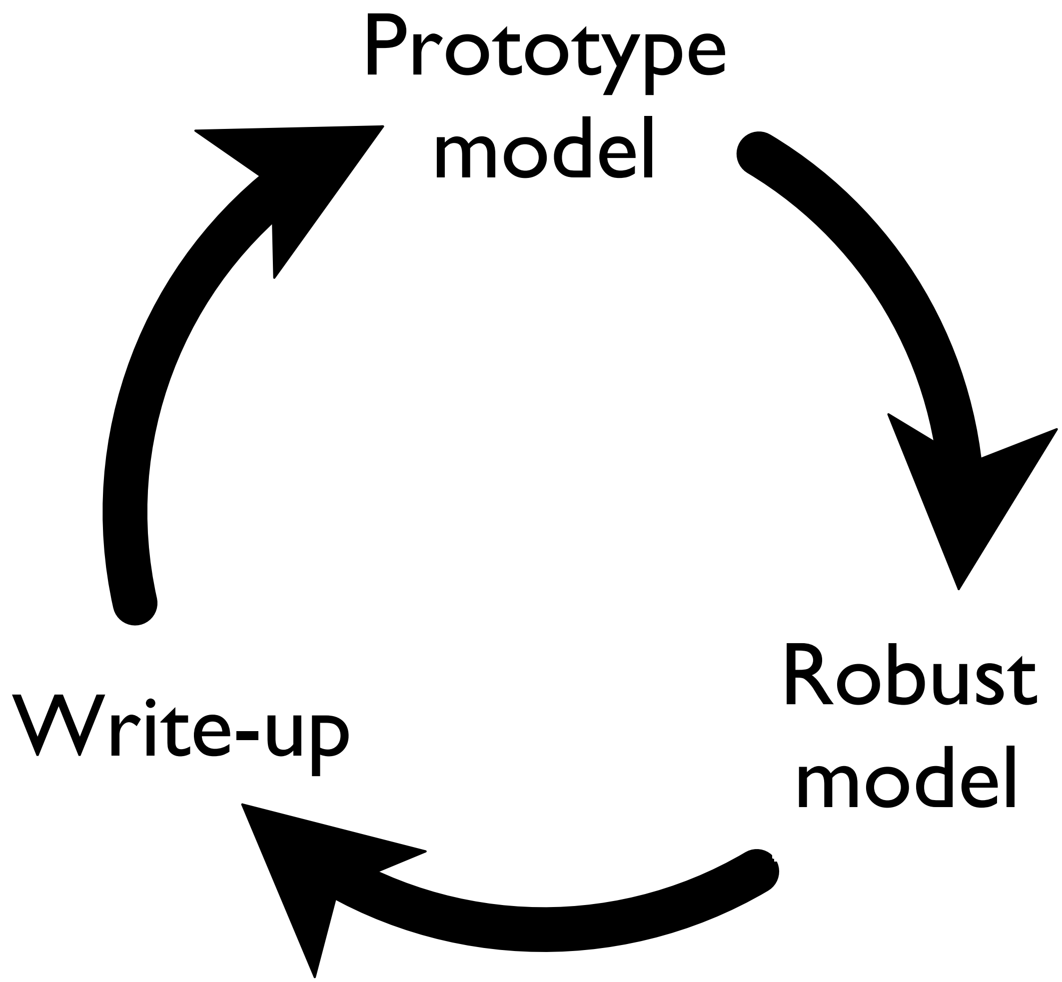
Moral: bugs are **everywhere**

A niche for mechanized metatheory:

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

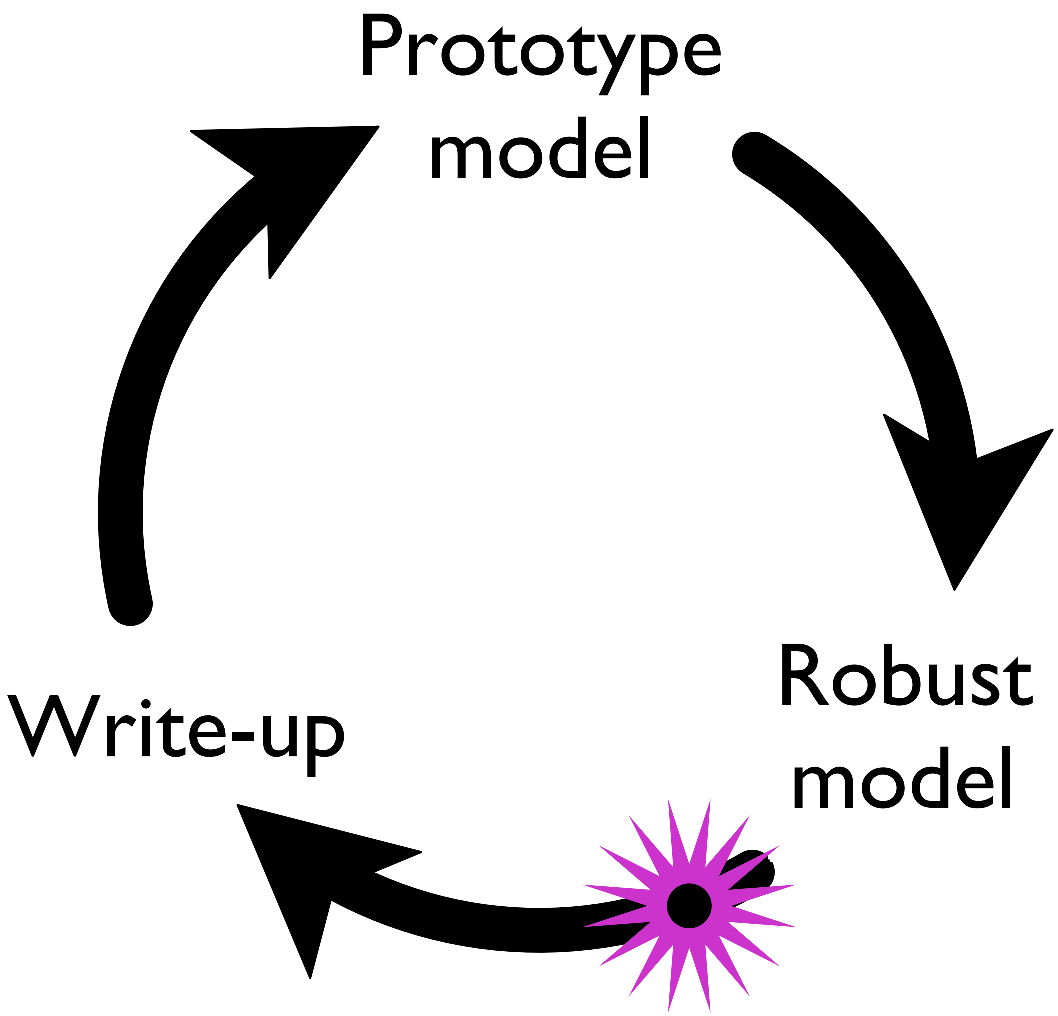
- supports the entire semantics lifecycle: 

The Semantics Lifecycle



The Semantics Lifecycle

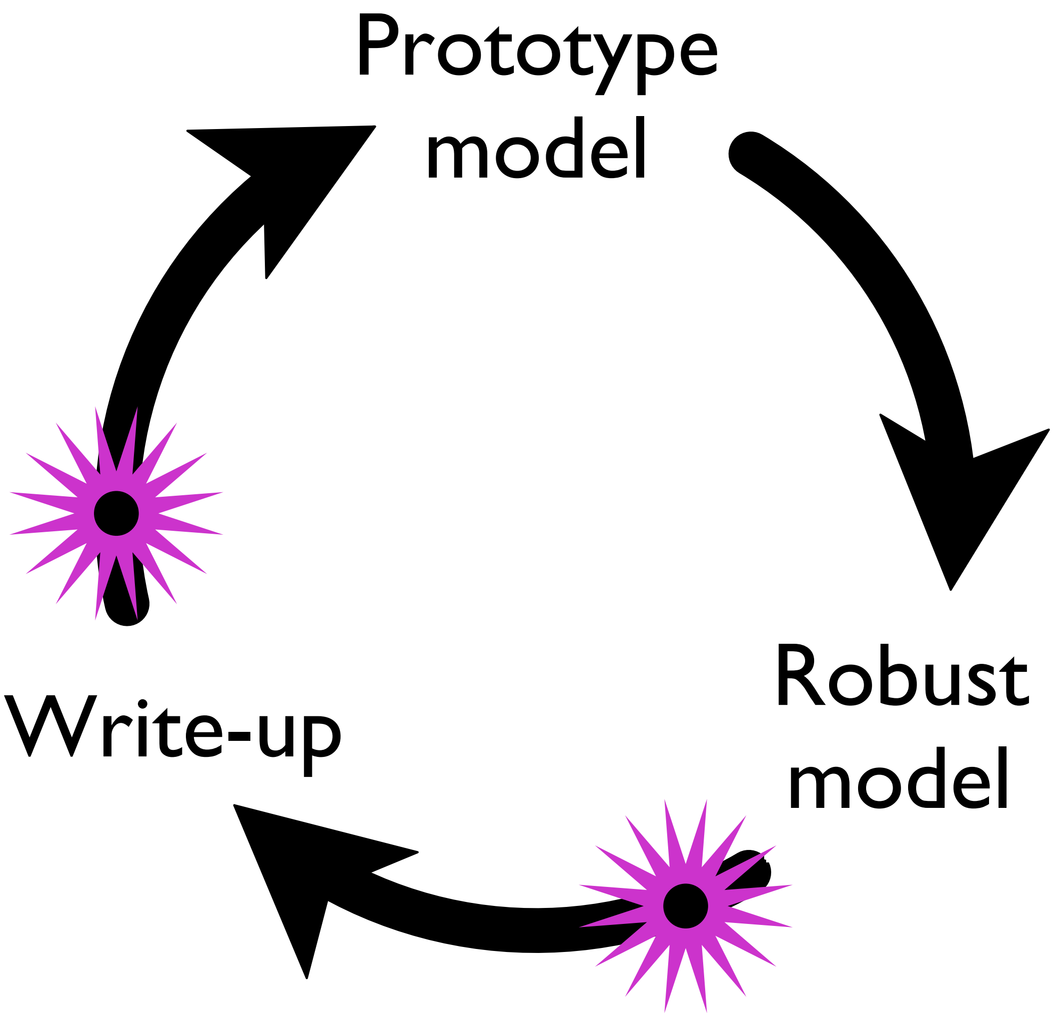
misrenamed
non-terminal



The Semantics Lifecycle

misrenamed
non-terminal

forgot
typing
rule

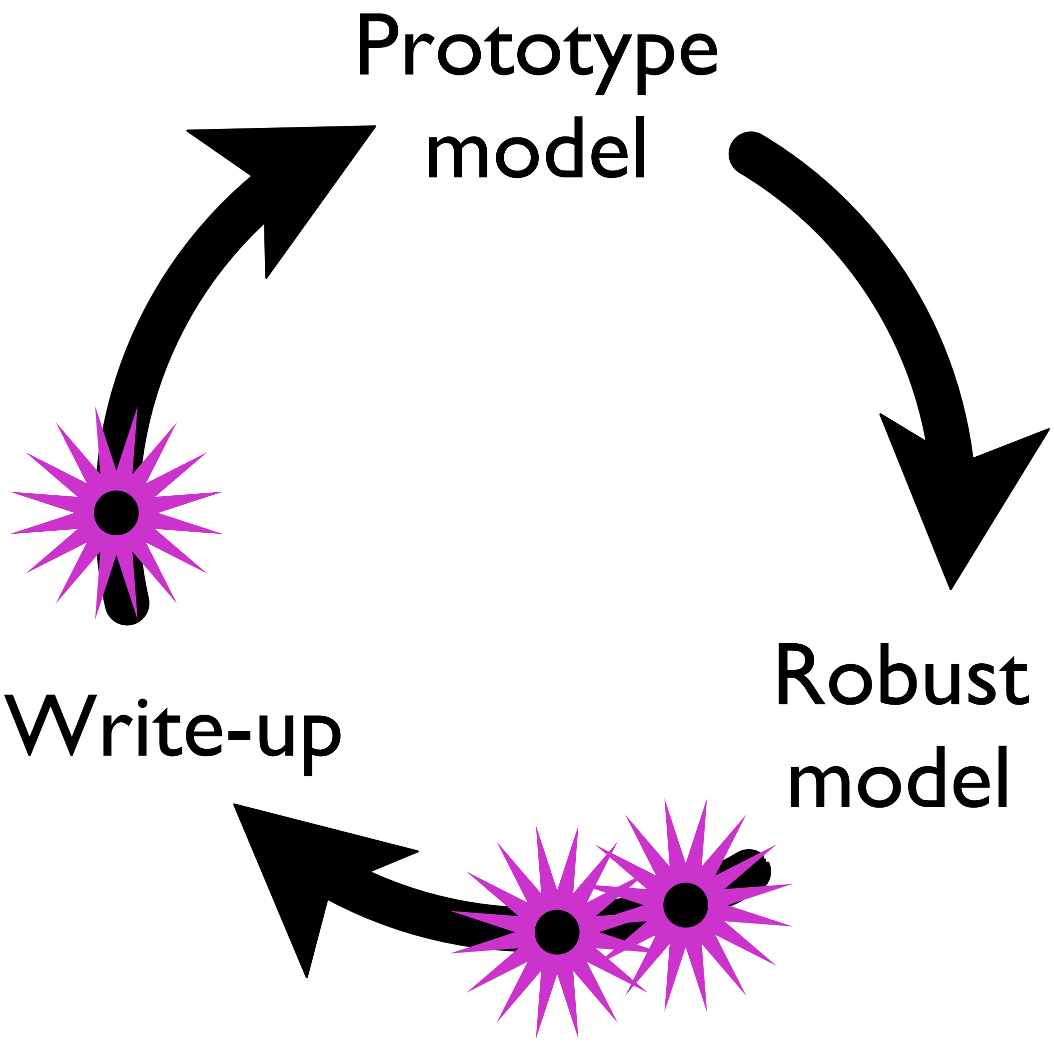


The Semantics Lifecycle

misrenamed non-terminal

forgot typing rule

lost a case in a helper function



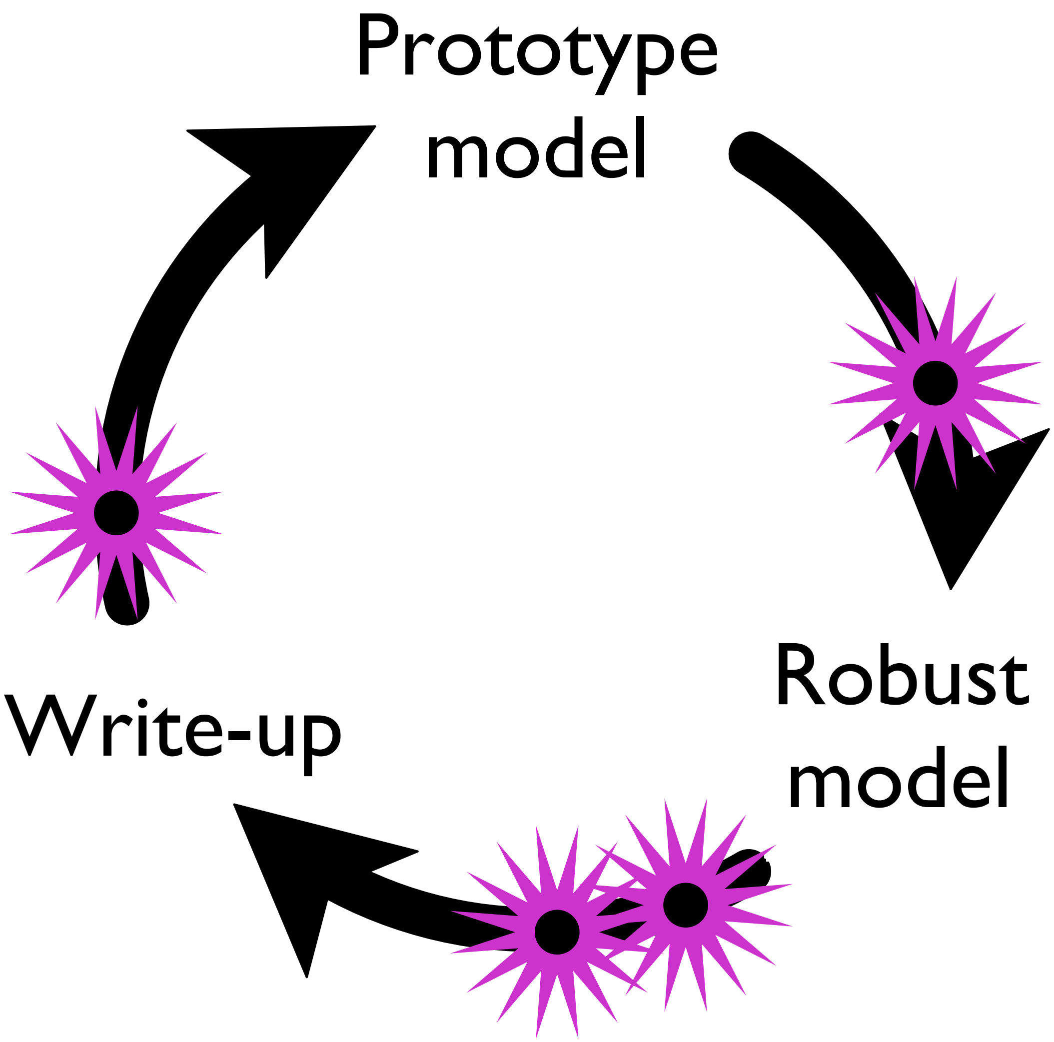
The Semantics Lifecycle

misrenamed non-terminal

forgot typing rule

lost a case in a helper function

added a case to wrong fn



The Semantics Lifecycle

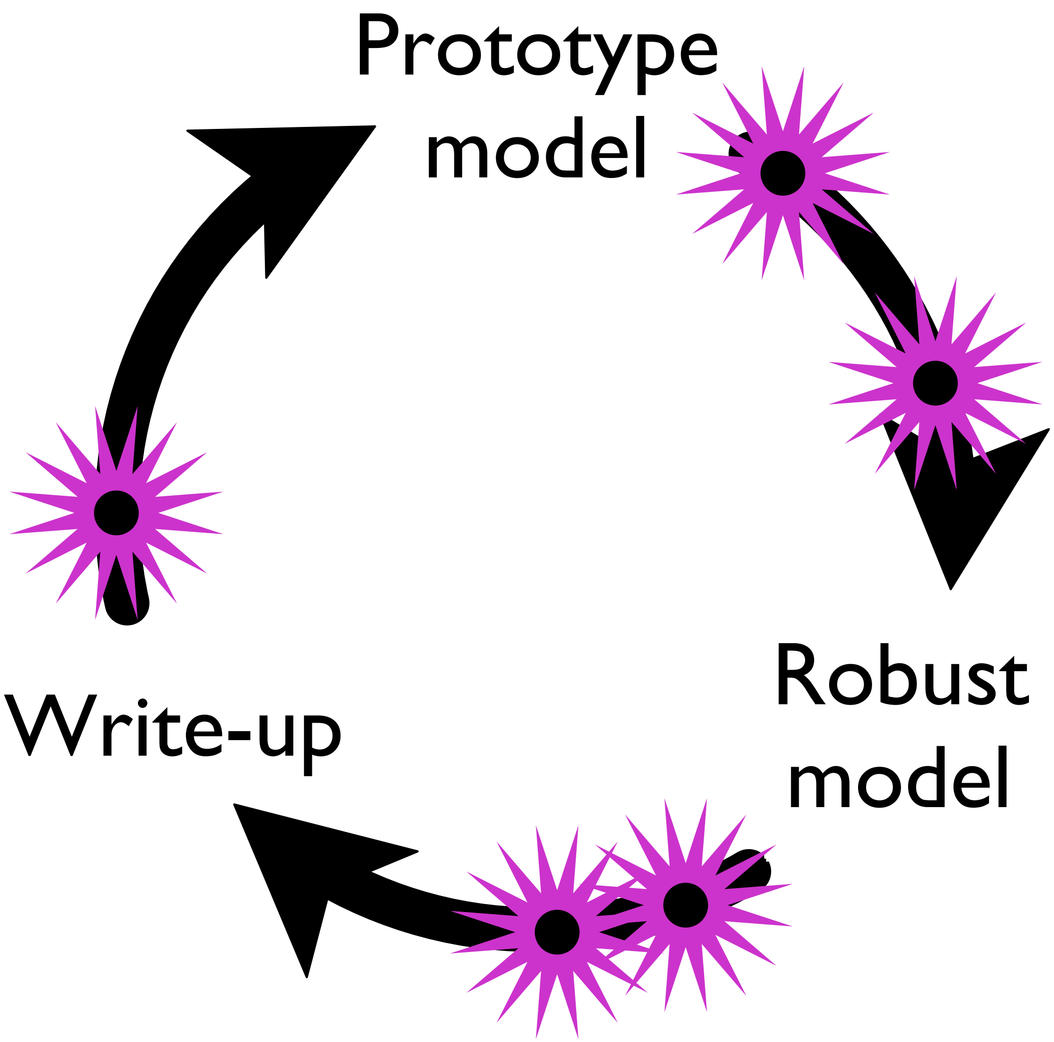
misrenamed non-terminal

forgot typing rule

lost a case in a helper function

added a case to wrong fn

swappped args



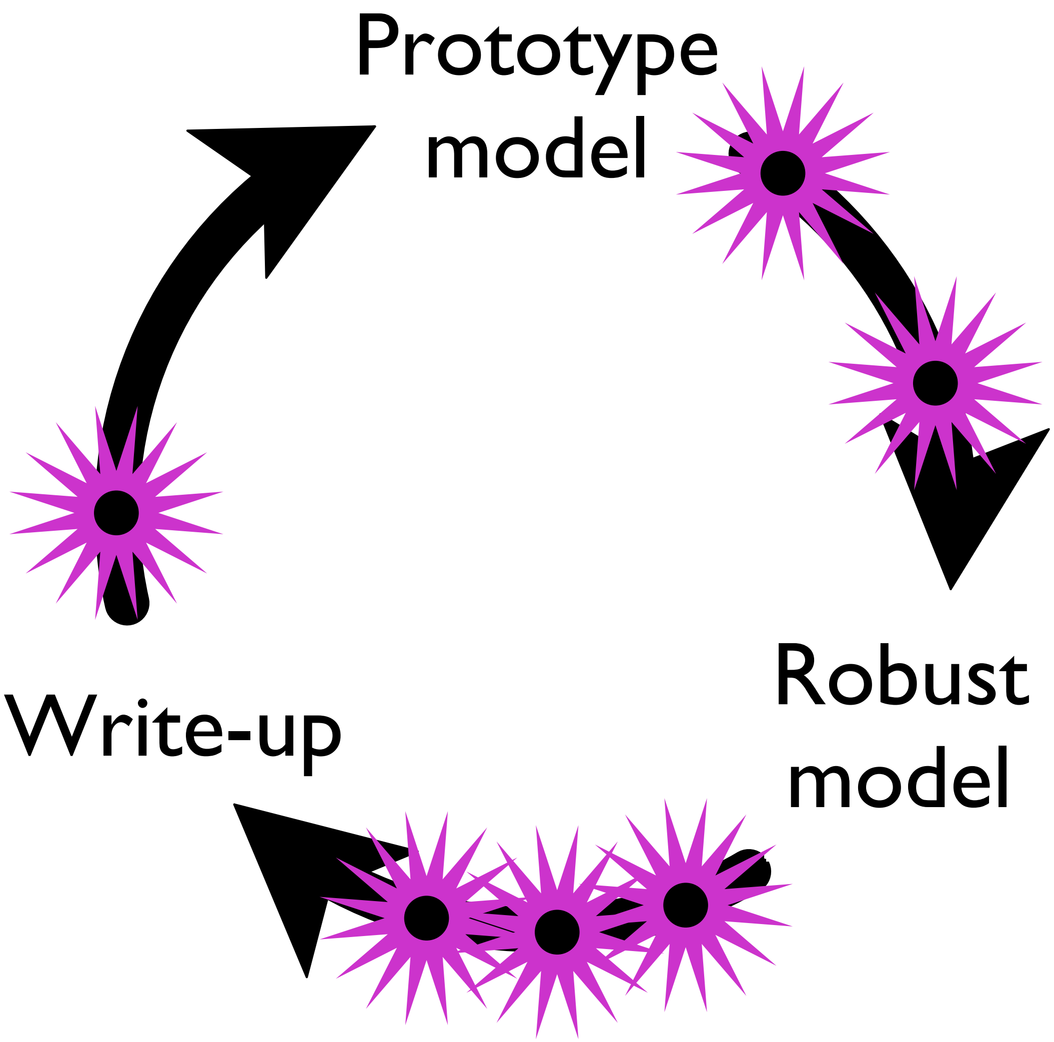
The Semantics Lifecycle

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.

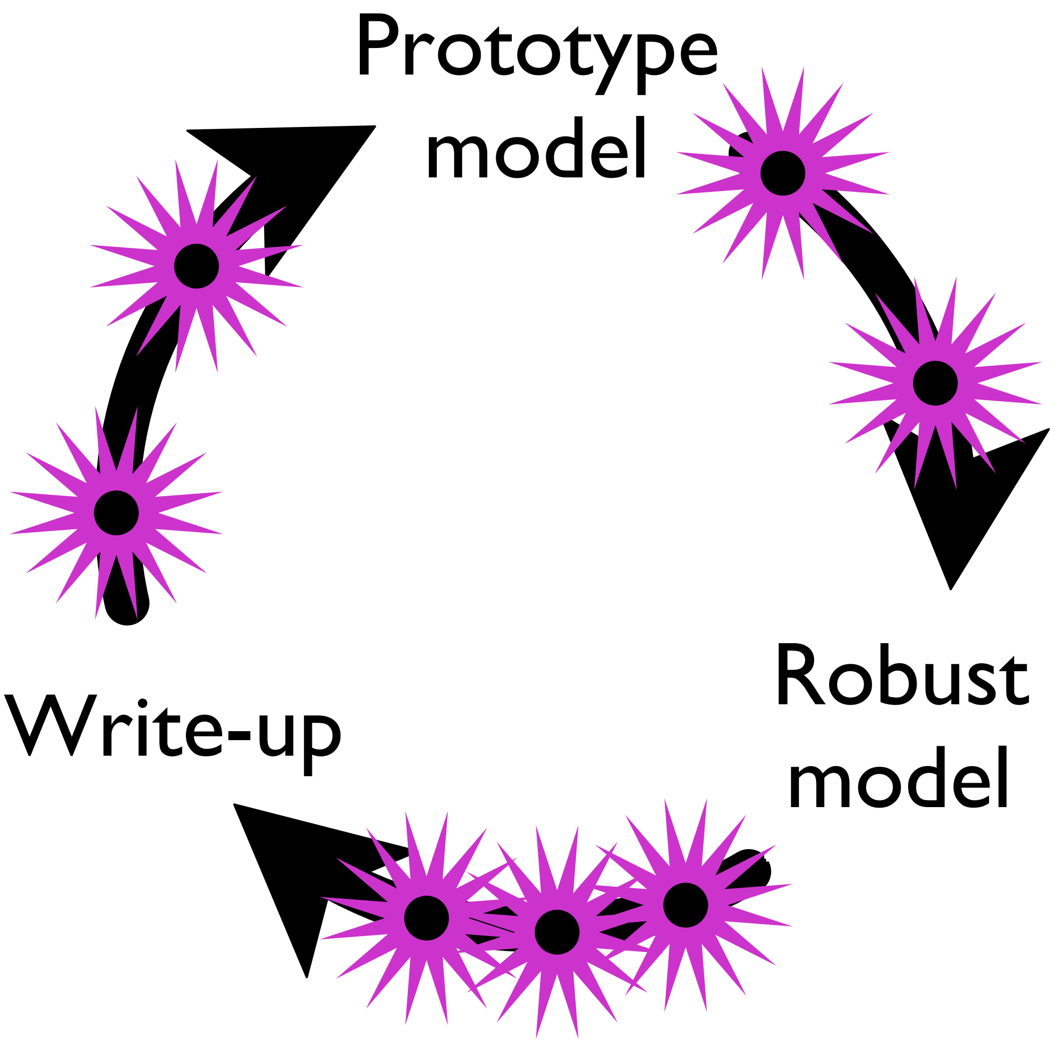
The Semantics Lifecycle

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.

didn’t recheck a lemma

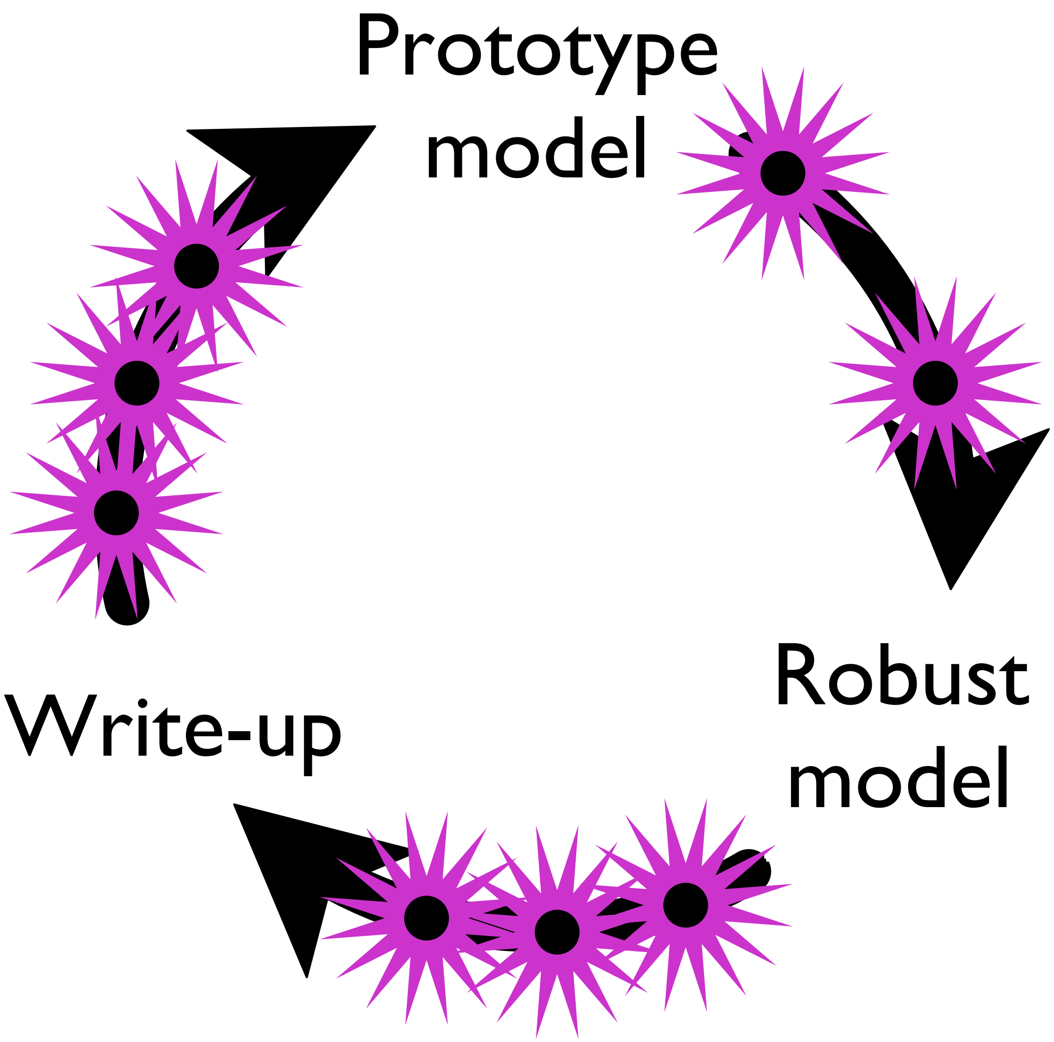
The Semantics Lifecycle

misrenamed non-terminal

forgot typing rule

lost a case in a helper function

added a case to wrong fn



swappped args

misused the inductive hyp.

didn't recheck a lemma

transcribed math wrong

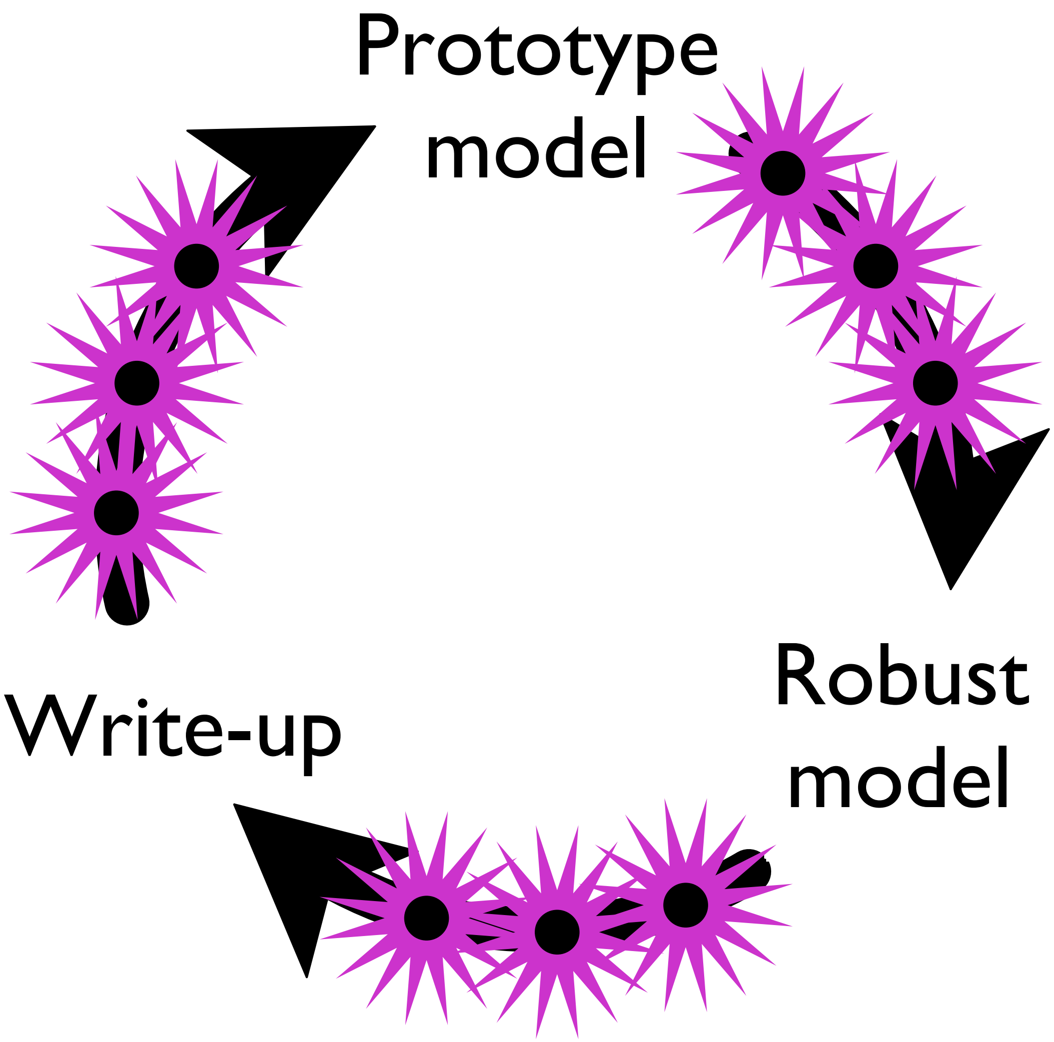
The Semantics Lifecycle

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.

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

10 papers in Redex
9 ICFP '09 papers
8 written by others
2 mechanically verified

$$\frac{10}{10}$$

papers with errors

10 papers in Redex
9 ICFP ’09 papers
8 written by others
2 mechanically verified

$$\frac{10}{10}$$

Your

papers
have
errors
too

Copy & Paste Typesetting Error:

$$\frac{sf \xrightarrow{\delta t}_{\phi_1} (sf_{\phi}, NoEvent :: bs)}{switch\ sf\ f \xrightarrow{\delta t}_{\phi_1} (switch_{\phi}\ sf_{\phi}\ f, bs)}\ \Phi_1\text{-SW-NOEV}$$

$$\frac{sfs \xrightarrow{\delta t}_{\phi_1} (sfs_{\phi}, Event\ e :: bss) \qquad f\ e \mapsto sfr \qquad sfr \xrightarrow{0}_{\phi_1} (sfr_{\phi}, bsr)}{switch\ sfs\ f \xrightarrow{\delta t}_{\phi_1} (sfr_{\phi}, bsr)}\ \Phi_1\text{-SW-EV}$$

$$\frac{sf \xrightarrow{\delta t}_{\phi_1} (sf_{\phi}, NoEvent :: bs)}{dswitch\ sf\ f \xrightarrow{\delta t}_{\phi_1} (dswitch_{\phi}\ sf_{\phi}\ f, bs)}\ \Phi_1\text{-DSW-NOEV}$$

$$\frac{sfs \xrightarrow{\delta t}_{\phi_1} (sfs_{\phi}, Event\ e :: bss) \qquad f\ e \mapsto sfr \qquad sfr \xrightarrow{0}_{\phi_1} (sfr_{\phi}, bsr)}{switch\ sfs\ f \xrightarrow{\delta t}_{\phi_1} (sfr_{\phi}, bss)}\ \Phi_1\text{-DSW-EV}$$

Copy & Paste Typesetting Error:

$$\frac{sf \ \stackrel{\delta t}{\Longrightarrow}_{\phi_1} \ (sf_\phi, NoEvent :: bs)}{switch\ sf\ f \ \stackrel{\delta t}{\Longrightarrow}_{\phi_1} \ (switch_\phi\ sf_\phi\ f, bs)} \ \Phi_1\text{-SW-NOEV}$$

$$\frac{sfs \ \stackrel{\delta t}{\Longrightarrow}_{\phi_1} \ (sfs_\phi, Event\ e :: bss) \qquad f\ e \mapsto sfr \qquad sfr \ \stackrel{0}{\Longrightarrow}_{\phi_1} \ (sfr_\phi, bsr)}{switch\ sfs\ f \ \stackrel{\delta t}{\Longrightarrow}_{\phi_1} \ (sfr_\phi, bsr)} \ \Phi_1\text{-SW-EV}$$

$$\frac{sf \ \stackrel{\delta t}{\Longrightarrow}_{\phi_1} \ (sf_\phi, NoEvent :: bs)}{dswitch\ sf\ f \ \stackrel{\delta t}{\Longrightarrow}_{\phi_1} \ (dswitch_\phi\ sf_\phi\ f, bs)} \ \Phi_1\text{-DSW-NOEV}$$

$$\frac{sfs \ \stackrel{\delta t}{\Longrightarrow}_{\phi_1} \ (sfs_\phi, Event\ e :: bss) \qquad f\ e \mapsto sfr \qquad sfr \ \stackrel{0}{\Longrightarrow}_{\phi_1} \ (sfr_\phi, bsr)}{switch\ sfs\ f \ \stackrel{\delta t}{\Longrightarrow}_{\phi_1} \ (sfr_\phi, bss)} \ \Phi_1\text{-DSW-EV}$$

Copy & Paste Typesetting Error:

$$\frac{sf \xrightarrow{\delta t}_{\phi_1} (sf_\phi, NoEvent :: bs)}{switch\ sf\ f \xrightarrow{\delta t}_{\phi_1} (switch_\phi\ sf_\phi\ f, bs)}\ \Phi_1\text{-SW-NOEV}$$

$$\frac{sfs \xrightarrow{\delta t}_{\phi_1} (sfs_\phi, Event\ e :: bss) \qquad f\ e \mapsto sfr \qquad sfr \xrightarrow{0}_{\phi_1} (sfr_\phi, bsr)}{switch\ sfs\ f \xrightarrow{\delta t}_{\phi_1} (sfr_\phi, bsr)}\ \Phi_1\text{-SW-EV}$$

$$\frac{sf \xrightarrow{\delta t}_{\phi_1} (sf_\phi, NoEvent :: bs)}{dswitch\ sf\ f \xrightarrow{\delta t}_{\phi_1} (dswitch_\phi\ sf_\phi\ f, bs)}\ \Phi_1\text{-DSW-NOEV}$$

$$\frac{sfs \xrightarrow{\delta t}_{\phi_1} (sfs_\phi, Event\ e :: bss) \qquad f\ e \mapsto sfr \qquad sfr \xrightarrow{0}_{\phi_1} (sfr_\phi, bsr)}{switch\ sfs\ f \xrightarrow{\delta t}_{\phi_1} (sfr_\phi, bss)}\ \Phi_1\text{-DSW-EV}$$

Typesetting should be automatic

Erroneous Example:

$$\begin{aligned}\Sigma; \cdot \vdash (\lambda y : \mathit{Lazy\ Int}.\ y + 1)(\lambda x : \mathit{Unit}.\ e) \rightsquigarrow \\ (\lambda y : \mathit{Lazy\ Int}.(\mathit{force}[\mathit{Int}]\ y) + 1) \\ (\mathit{lazy}\ \lambda x : \mathit{Unit}.\ e) : \mathit{Int}\end{aligned}$$

Erroneous Example:

$$\Sigma; \cdot \vdash (\lambda y : Lazy\ Int.\ y + 1)(\lambda x : Unit.\ e) \rightsquigarrow$$
$$(\lambda y : Lazy\ Int. (force[Int]\ y) + 1)$$
$$(lazy\ \lambda x : Unit.\ e) : Int$$

Erroneous Example:

$$\begin{aligned}\Sigma; \cdot \vdash &(\lambda y : Lazy\ Int.\ y + 1)(\lambda x : Unit.\ e) \rightsquigarrow \\ &(\lambda y : Lazy\ Int. (force[Int]\ y) + 1) \\ &(lazy[Int]\ \lambda x : Unit.\ m) : Int\end{aligned}$$

$$\text{where} \quad \Sigma; \{x : Unit\} \vdash e \rightsquigarrow m$$

Erroneous Example:

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

Examples can be tested

Unexpected Behavior:

$$\mathrm{select}(c, \overline{c})$$

Unexpected Behavior:

$$\mathrm{select}(c, \overline{c})$$

compile

$$\odot_c \mid \widetilde{\mathrm{select}}(c, \overline{c})$$

Unexpected Behavior:

$$\text{select}(c, \overline{c}) \quad \text{– stuck}$$

compile

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

Deadlock in source but busy waiting in target

Unexpected Behavior:

$$\mathrm{select}(c, \overline{c}) \qquad \text{– stuck}$$

compile

$$\odot_c \mid \widetilde{\mathrm{select}}(c, \overline{c}) \qquad \text{– loops forever}$$

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

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

but $(\lambda_\delta\, x.\, x)\ 1 \mapsto 1$

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

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

but $(\lambda_\delta\, x.\, x)\ 1 \mapsto 1$

Not a fly-by-night proof; 12 typeset pages in a dissertation chapter

Random testing easily finds this

Recap:

- Automatic typesetting
- Unit Testing
- Exploring Examples
- Random testing

$$
\begin{aligned}
p &::= (e\ \ldots) \\
e &::= (e\ e\ \ldots) \\
&\mid (\lambda\ (x{:}t\ \ \ldots)\ e) \\
&\mid x \\
&\mid (\mathbf{+}\ e\ \ldots) \\
&\mid \mathit{number} \\
&\mid (\mathbf{amb}\ e\ \ldots) \\
t &::= (\rightarrow\ t\ \ldots\ t) \mid \mathbf{num}
\end{aligned}
$$

$$
\begin{aligned}
P &::= (e\ \ldots\ E\ e\ \ldots) \\
E &::= (v\ \ldots\ E\ e\ \ldots) \\
&\mid (\mathbf{+}\ v\ \ldots\ E\ e\ \ldots) \\
&\mid \mathbf{[]} \\
v &::= (\lambda\ (x{:}t\ \ \ldots)\ e) \\
&\mid \mathit{number} \\
\Gamma &::= \cdot \mid (x : t\ \Gamma)
\end{aligned}
$$

$$
\begin{array}{ll}
\quad P[((\lambda\ (x{:}t\ \ \ldots_1)\ e)\ v\ \ldots_1)] & \textsf{[βv]} \\
\longrightarrow P[e\{x{:=}v\ \ldots\}] &
\end{array}
$$

$$
\begin{array}{ll}
\quad P[(\mathbf{+}\ \mathit{number}_1\ \ldots)] & \textsf{[+]} \\
\longrightarrow P[\Sigma[[\mathit{number}_1, \ldots]]] &
\end{array}
$$

$$
\begin{array}{ll}
\quad (e_1\ \ldots\ E[(\mathbf{amb}\ e_2\ \ldots)]\ e_3\ \ldots) & \textsf{[amb]} \\
\longrightarrow (e_1\ \ldots\ E[e_2]\ \ldots\ e_3\ \ldots) &
\end{array}
$$

$$\frac{\Gamma \vdash e_1 : (\rightarrow\ t_2\ \ldots\ t_3) \quad \Gamma \vdash e_2 : t_2 \quad \ldots}{\Gamma \vdash (e_1\ e_2\ \ldots) : t_3}$$

$$\frac{(x_1 : t_1\ \Gamma) \vdash (\lambda\ (x_2{:}t_2\ \ \ldots)\ e) : (\rightarrow\ t_2\ \ldots\ t)}{\Gamma \vdash (\lambda\ (x_1{:}t_1\ x_2{:}t_2\ \ \ldots)\ e) : (\rightarrow\ t_1\ t_2\ \ldots\ t)}$$

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash (\lambda\ ()\ e) : (\rightarrow\ t)}$$

$$\frac{}{(x : t\ \Gamma) \vdash x : t}$$

$$\frac{\Gamma \vdash x_1 : t_1 \quad x_1 \neq x_2}{(x_2 : t_2\ \Gamma) \vdash x_1 : t_1}$$

$$\frac{\Gamma \vdash e : \mathbf{num} \quad \ldots}{\Gamma \vdash (\mathbf{+}\ e\ \ldots) : \mathbf{num}}$$

$$\frac{}{\Gamma \vdash \mathit{number} : \mathbf{num}}$$

$$\frac{\Gamma \vdash e : \mathbf{num} \quad \ldots}{\Gamma \vdash (\mathbf{amb}\ e\ \ldots) : \mathbf{num}}$$

$$| \ (\mathbf{amb}\ e\ \ldots)$$

$$\frac{\Gamma \vdash e : \mathbf{num} \quad \ldots}{\Gamma \vdash (\mathbf{amb}\ e\ \ldots) : \mathbf{num}}$$

$$p ::= (e\ \ldots)$$

$$\begin{array}{ll} & (e_1\ \ldots\ E[(\mathbf{amb}\ e_2\ \ldots)]\ e_3\ \ldots) \quad \text{[amb]} \\ \longrightarrow & (e_1\ \ldots\ E[e_2]\ \ldots\ e_3\ \ldots) \end{array}$$

$p ::= (e\ \ldots)$
$e ::= (e\ e\ \ldots)$
$\quad | \ (\lambda\ (x{:}t\ \ldots)\ e)$
$\quad | \ x$
$\quad | \ (\mathbf{+}\ e\ \ldots)$
$\quad | \ number$
$\quad | \ (\mathbf{amb}\ e\ \ldots)$
$t ::= (\rightarrow t\ \ldots\ t)\ |$

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

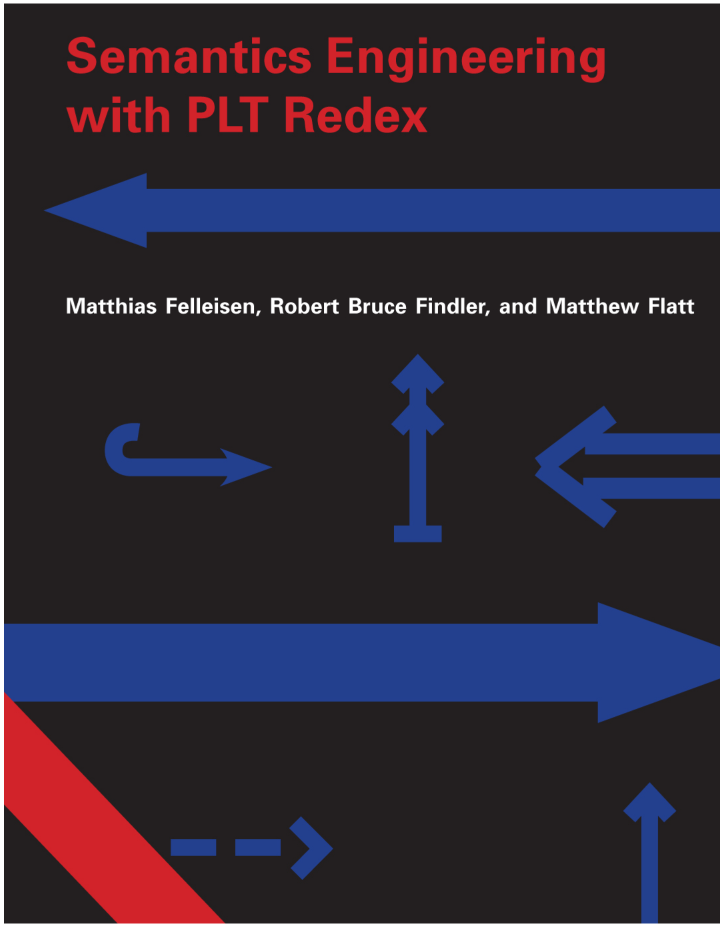
Recap:

- ✓ 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.


Semantics Engineering
with PLT Redex
Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt