Vulnerable Semantics of Solidity

Sukyoung Ryu

with PLRG@KAIST and friends

October 20, 2018

Fortress, JavaScript, and Solidity

Sukyoung Ryu

with PLRG@KAIST and friends

October 20, 2018

KAIST: SML and OCaml

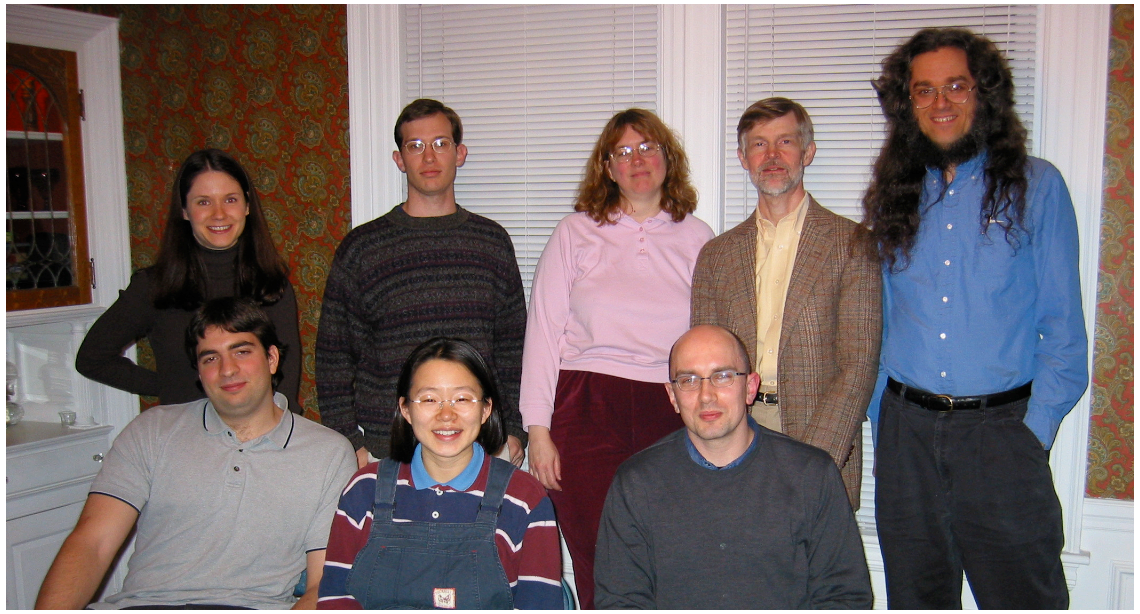
Links to other SML resources

Please feel free to send additional URLs that should be on this list to smlnj-dev-list@mailman.cs.uchicago.edu

SML Programming Resources

- Concurrent ML
- sml_tk, a library for using the TK graphical interface, now updated to version 3.0
- Martin Erwig's Functional Graph Library
- Yi and Ryu's SML/NJ exception analyzer

Harvard: Assembly, C, and Modula-3

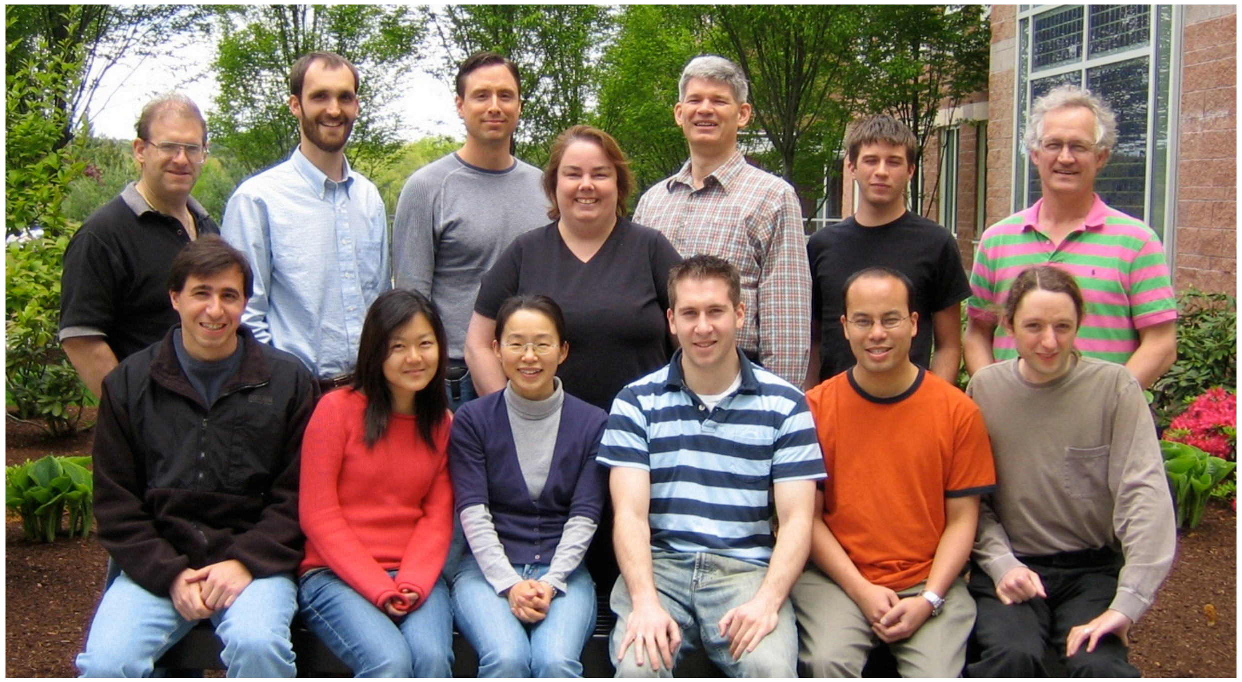


Debugging Everywhere

The goal of the *Debugging Everywhere* project is to make debugging a cheap, ubiquitous service. We intend to begin by getting compilers to emit *Active Debugging Information*, which we expect will support multi-language, multi-platform debugging much more readily than older approaches like Dwarf or dbx ``stabs."

```
ldb Fib (stopped) > t
  0 <_print:2> (Mips/mjr.c:25,2)  void _print(char *s = (0x1000008c) " ")
* 1 <fib:51+0x24> (Fib.java:23,32)
         void fib(Fib this = {int buffer = 10,
                              int[] a = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55,
                                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
                             }, int n = 10)
  2 <main:3+0x18> (Fib.java:5,23)  void main(String[] argv = {})
  3 <mAiN:end+0x1c> (mininub.c:7,9)
         int mAiN(int argc = 2, char **argv = 0x7fff7b24,
                  char **envp = 0x7fff7b34)
```

Sun Microsystems: Java and Scala

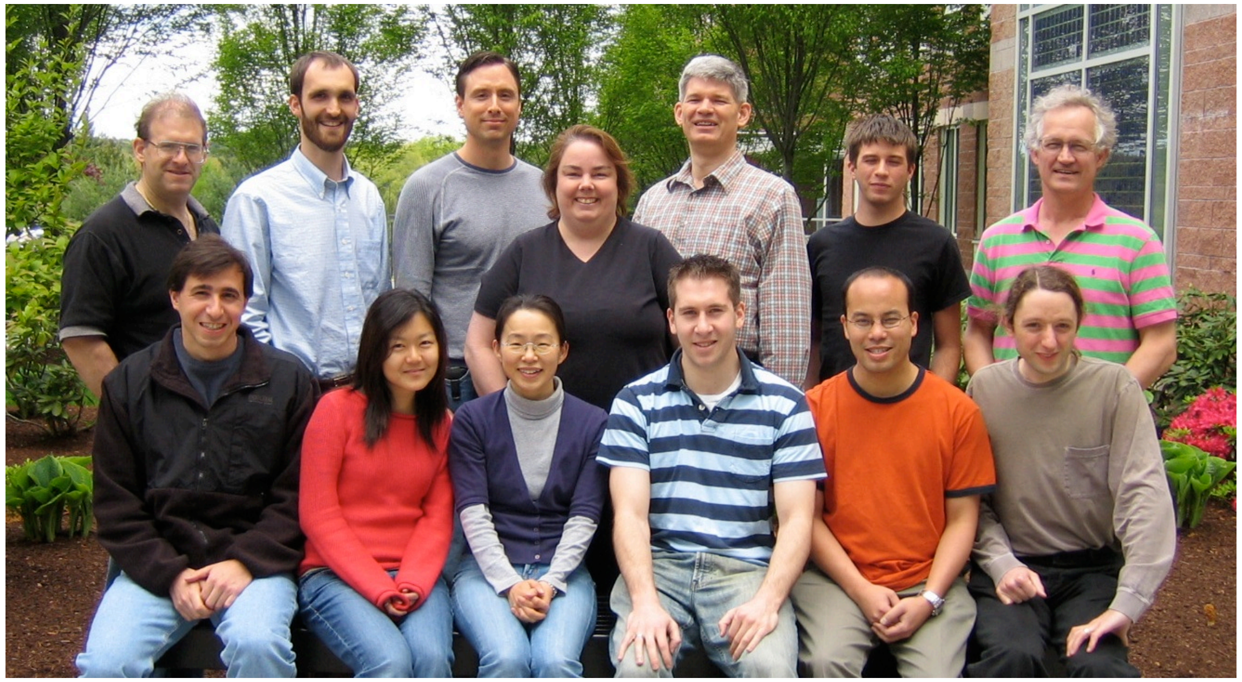


$$
\begin{aligned}
&z\colon \mathrm{Vec} := 0\\
&r\colon \mathrm{Vec} := x\\
&p\colon \mathrm{Vec} := r\\
&\rho\colon \mathrm{Elt} := r^T\, r\\
&\texttt{for } j \leftarrow seq(1 : cgit_{\max}) \texttt{ do}\\
&\quad q = A\, p\\
&\quad \alpha = \frac{\rho}{p^T\, q}\\
&\quad z := z + \alpha\, p\\
&\quad r := r - \alpha\, q\\
&\quad \rho_0 = \rho\\
&\quad \rho := r^T\, r\\
&\quad \beta = \frac{\rho}{\rho_0}\\
&\quad p := r + \beta\, p\\
&\texttt{end}\\
&(z, \|x - A\, z\|)
\end{aligned}
$$

sources / ProjectFortress / src / com / sun / fortress / parser

Filename	Author	Revision	Modified	Log Entry
..				
Compilation.rats	chf	4575	over 4 years ago	Updated Copyright notices
Declaration.rats	chf	4575	over 4 years ago	Updated Copyright notices
DelimitedExpr.rats	sukyoungryu	4921	about 4 years ago	Revising
Expression.rats	Guy Steele	5051	almost 4 years ago	Completely redid assignment and ...
Fortress.rats	chf	4575	over 4 years ago	Updated Copyright notices

Sun Microsystems: Fortress



$$v_{\mathrm{norm}} = v / \|v\|$$

$$\sum_{k \leftarrow 1:n} a_k \, x^k$$

$$C = A \cup B$$

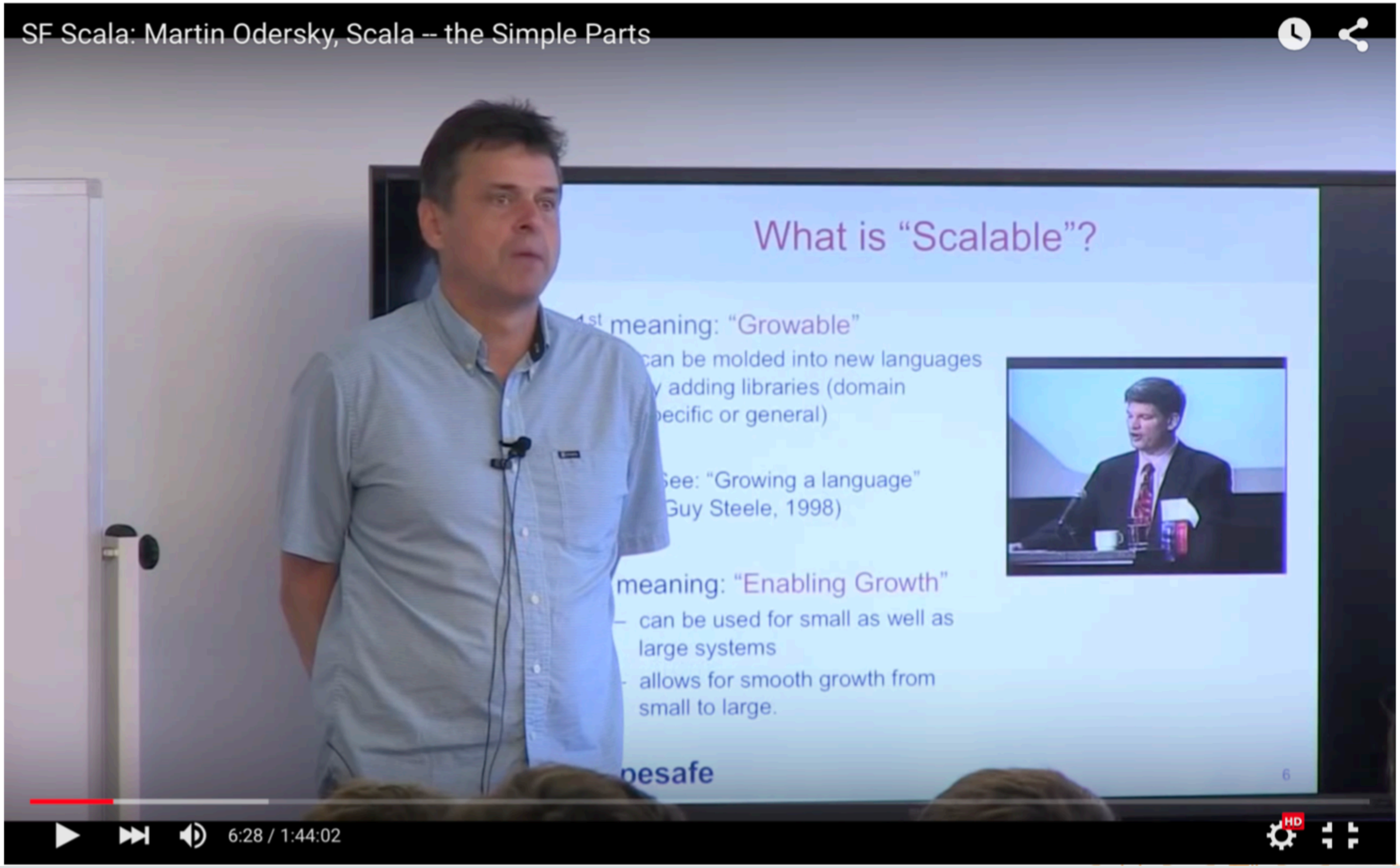
$$y = 3x \sin x \cos 2x \log \log x$$

- A multicore language for scientists and engineers
- Run your whiteboard in parallel!
- “Growing a Language” https://www.youtube.com/watch?v=_ahvzDzKdB0&t=10s
  - Guy L. Steele Jr., keynote talk, OOPSLA 1998
  - Higher-Order and Symbolic Computation 12, 221-236 (1999)

Sun Microsystems: Fortress

"It's an absolute piece of beauty"

https://www.youtube.com/watch?v=_ahvzDzKdB0



Language *Manipulation*: Fortress

- Specification
- Parsing
- Static checking
- Compilation / Interpretation
- Testing
- Analysis
- Verification

Fortress as a Language Designer

- Development from scratch
- Language evolves to experiment with new features
- Constant changes in spec., parsing, checking, ...
- Language development by 3 teams in tandem:
  - Specification team
  - Implementation team
  - Library team

Fortress as a Language Designer

- Specification team   
  - Formal concrete grammar in EBNF
  - Informal description in prose
  - Examples & formal calculi

Fortress as a Language Designer

- Specification team   
  - Formal concrete grammar in EBNF
  - Informal description in prose
  - Examples & formal calculi
- Library team  
  - Adventurous library in unimplemented Fortress features

Fortress as a Language Designer

- Specification team   
  - Formal concrete grammar in EBNF
  - Informal description in prose
  - Examples & formal calculi
- Library team  
  - Adventurous library in unimplemented Fortress features
- Implementation team  
  - Implementation extension
  - Regression tests & new feature tests

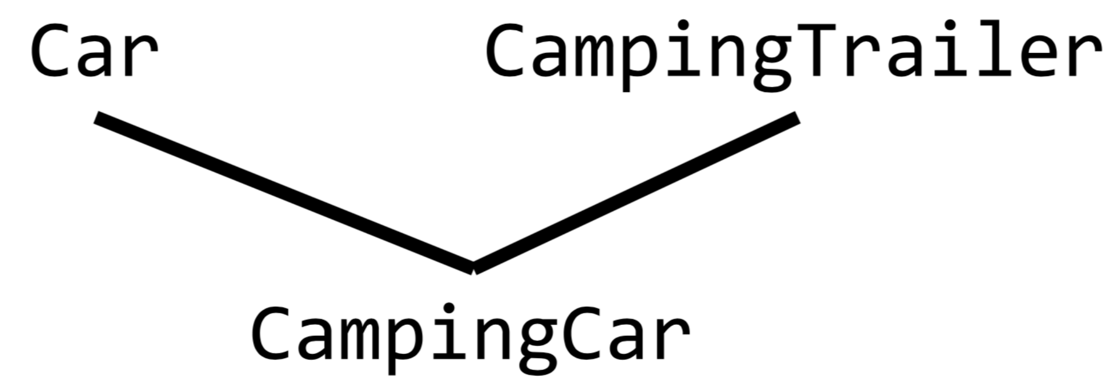
Language *Manipulation*: Fortress

- Specification: automatic extraction/test of examples
- Parsing: automatic generation of parsers & ASTs
- Static checking: parallel development
- Compilation / Interpretation: cross validation
- Testing
- Analysis
- Verification
  - FFMM (Featherweight Fortress with Multiple Dispatch and Multiple Inheritance): 3,000 LOC (Coq)

Fortress: Symmetric Multiple Dispatch

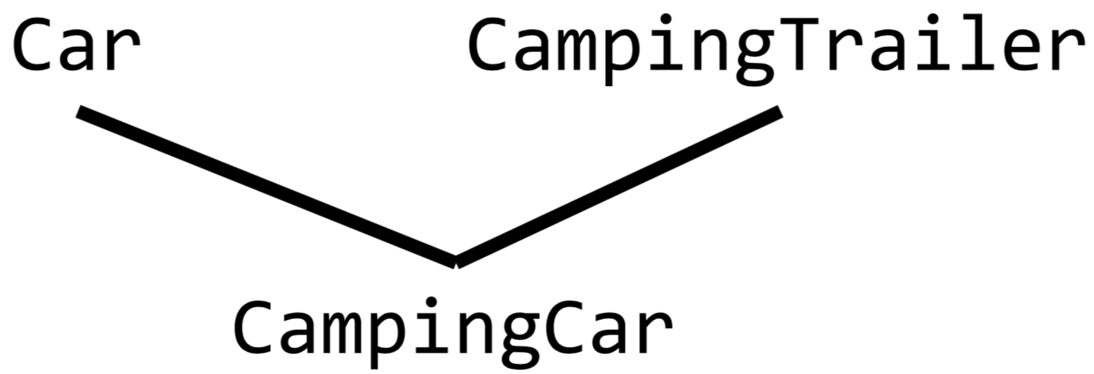
- Method overloading
  - Multiple method declarations of the same name
- Symmetric multiple dispatch
  - Selection of a method declaration at run time using all the arguments equally
- Overloading rules
  - Static rejection of ambiguous method declarations
  - No ambiguous nor undefined method call at run time!

Fortress: Symmetric Multiple Dispatch



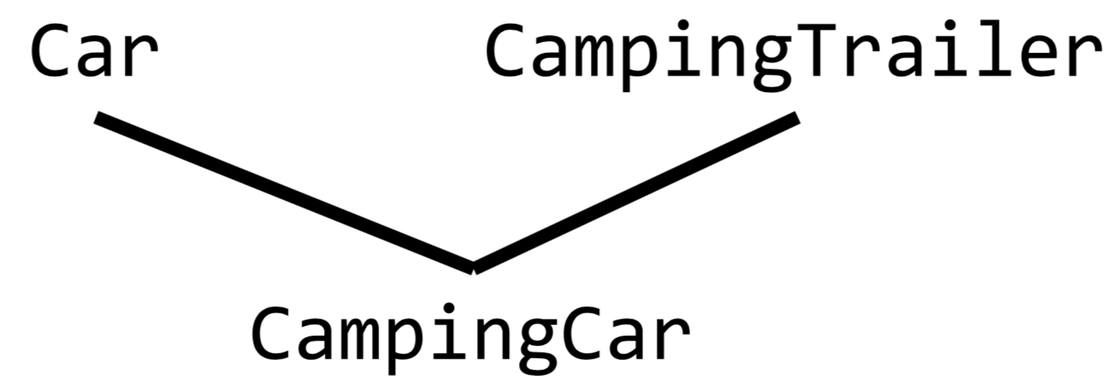

```
collide(c: Car, cc: CampingCar): Int = _
collide(cc: CampingCar, c: Car): Int = _

cc1: CampingCar = CampingCar()
cc2: CampingCar = CampingCar()
collide(cc1, cc2)
```

Fortress: Symmetric Multiple Dispatch



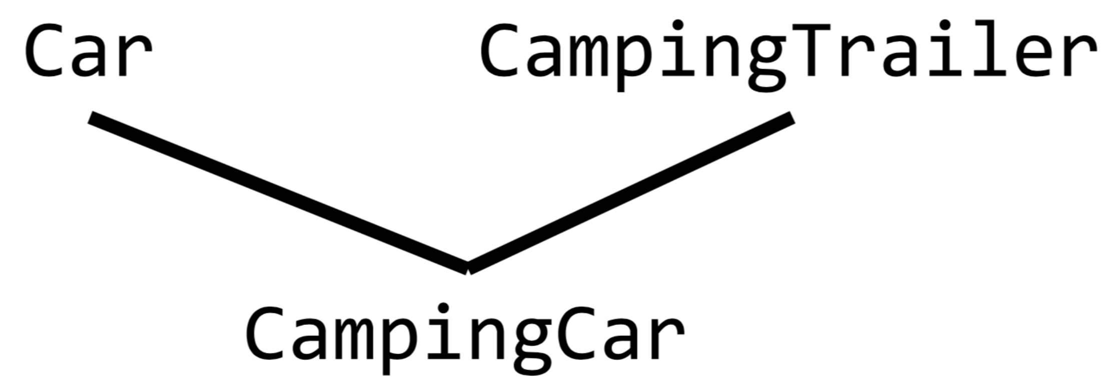

```
collide(c: Car, cc: CampingCar): Int = _
collide(cc: CampingCar, c: Car): Int = _
collide(cc: CampingCar, c: CampingCar): Int = _
cc1: CampingCar = CampingCar()
cc2: CampingCar = CampingCar()
collide(cc1, cc2)
```

Fortress: Symmetric Multiple Dispatch




```
sort[P <: Car](x: List[P]): SortedList[P] = _
sort[P <: CampingTrailer](x: List[P]): SortedList[P] = _

l: List[CampingCar] = List(CampingCar())
sort(cc)
```

Fortress: Symmetric Multiple Dispatch




```
sort[P <: Car](x: List[P]): SortedList[P] = _
sort[P <: CampingTrailer](x: List[P]): SortedList[P] = _
sort[P <: CampingCar](x: List[P]): SortedList[P] = _
l: List[CampingCar] = List(CampingCar())
sort(cc)
```

Fortress: Symmetric Multiple Dispatch with Parametric Polymorphism

Type Checking Modular Multiple Dispatch with Parametric Polymorphism and Multiple Inheritance

Eric Allen
Oracle Labs
eric.allen@oracle.com

Justin Hilburn
Oracle Labs
justin.hilburn@oracle.com

Scott Kilpatrick
University of Texas at Austin
scottk@cs.utexas.edu

Victor Luchangco
Oracle Labs
victor.luchangco@oracle.com

Sukyoung Ryu
KAIST
sryu.cs@kaist.ac.kr

David Chase
Oracle Labs
david.r.chase@oracle.com

Guy L. Steele Jr.
Oracle Labs
guy.steele@oracle.com

- OOPSLA’11
- No variance, No dynamic dispatch algorithm
- No type soundness proof

Fortress: Symmetric Multiple Dispatch with Polymorphism and Variance

Fortress: Symmetric Multiple Dispatch with Polymorphism and Variance PROVED

Polymorphic Symmetric Multiple Dispatch with Variance

GYUNGHEE PARK, Oracle Labs and KAIST
JAEMIN HONG, KAIST
GUY L. STEELE JR., Oracle Labs
SUKYOUNG RYU, KAIST

- POPL'19
- Yes variance, Yes dynamic dispatch algorithm
- Yes type soundness proof

Fortress: Symmetric Multiple Dispatch with Polymorphism and Variance PROVED

Program	Π	::=	$\overline{\psi}, e$
Class declaration	ψ	::=	$\texttt{trait}\ T[\![\overline{V\ \beta}]\!] <: \{\overline{t}\}\ \overline{\mu}\ \texttt{end} \mid \texttt{object}\ O[\![\overline{\beta}]\!](\overline{x\colon\tau}) <: \{\overline{t}\}\ \overline{\mu}\ \texttt{end}$
Method definition	μ	::=	$m[\![\overline{\kappa}]\!](\overline{x\colon\tau})\colon \tau = e$
Class type parameter binding	β	::=	$P <: \{\overline{\tau}\}$
Method type parameter binding	κ	::=	$\{\overline{\tau}\} <: P <: \{\overline{\tau}\}$
Variance mark	V	::=	$+ \mid - \mid =$
Expression	e	::=	$z \mid ((\overline{x\colon\tau})\colon \tau \Rightarrow e) \mid e@(\overline{e}) \mid O[\![\overline{\tau}]\!](\overline{e}) \mid e.m(\overline{e})$
Bindable variable	z	::=	$x \mid \texttt{self}$
Type	τ	::=	$P \mid c \mid (\overline{\tau}) \mid (\tau \rightarrow \tau) \mid \texttt{Any}$
Constructed type	c	::=	$t \mid O[\![\overline{\tau}]\!]$
Trait type	t	::=	$T[\![\overline{\tau}]\!]$

Fortress: Symmetric Multiple Dispatch with Polymorphism and Variance

PROVED

Well-formed programs: $\boxed{\vdash \Pi : g}$

$$\frac{\Pi = \overline{\psi}, e \qquad \mathit{distinct}(\overline{\mathit{name}(\psi)}) \qquad \Delta = \{\overline{\psi}\} \qquad \overline{\underline{\Delta} \vdash \psi\ \mathsf{ok}} \qquad \Delta; \bullet \vdash e : (\_, g)}{\vdash \Pi : g}\ [\textsc{T-Program}]$$

Well-formed trait and object declarations: $\boxed{\Delta \vdash \psi\ \mathsf{ok}}$

$$\frac{\begin{array}{c} \Delta' = \Delta \cup \left\{\overline{\{\} <: P <: \{\overline{\xi}\}}\right\} \qquad \mathit{distinct}(\overline{P}) \qquad \overline{\overline{\underline{\underline{\Delta'}} \vdash \xi\ \mathsf{ok}}} \qquad \overline{\underline{\Delta'} \vdash t\ \mathsf{ok}} \\ \{\overline{J}\} = \mathit{properAncestors}(\Delta', T[\![\overline{P}]\!]) \qquad \overline{\Delta'; \overline{J} \vdash J_i \text{ and } J_j \text{ ancestors ok}}^{1 \le i < j \le \#(\overline{J})} \\ \overline{\underline{T} \ne \mathit{name}(J)} \qquad \overline{\underline{\Delta'; \mathtt{self}{:}T[\![\overline{P}]\!]; \overline{V\ P}} \vdash \mu\ \mathsf{ok}} \qquad \{\overline{d}\} = \mathit{allVisible}(\Delta', T[\![\overline{P}]\!]) \\ \overline{\Delta' \vdash d_i \text{ not duplicate of } d_j}^{1 \le i < j \le \#(\overline{d})} \qquad \overline{\Delta' \vdash d_i \text{ meet } d_j \text{ wrt } \{\overline{d}\}\ \mathsf{ok}}^{1 \le i < j \le \#(\overline{d})} \\ \overline{\Delta' \vdash d_i \text{ return type wrt } d_j\ \mathsf{ok}}^{1 \le i < j \le \#(\overline{d})} \qquad \overline{\Delta' \vdash d_j \text{ return type wrt } d_i\ \mathsf{ok}}^{1 \le i < j \le \#(\overline{d})} \end{array}}{\Delta \vdash \mathtt{trait}\ T[\![\overline{V\ P <: \{\overline{\xi}\}}]\!] <: \left\{\overline{t}\right\}\ \overline{\mu}\ \mathsf{end\ ok}}\ [\textsc{D-Trait}]$$

$$\frac{\begin{array}{c} \Delta' = \Delta \cup \left\{\overline{\{\} <: P <: \{\overline{\xi}\}}\right\} \qquad \mathit{distinct}(\overline{P}) \qquad \overline{\overline{\underline{\underline{\Delta'}} \vdash \xi\ \mathsf{ok}}} \qquad \overline{\underline{\Delta'} \vdash t\ \mathsf{ok}} \\ \{\overline{J}\} = \mathit{properAncestors}(\Delta', O[\![\overline{P}]\!]) \qquad \overline{\Delta'; \overline{J} \vdash J_i \text{ and } J_j \text{ ancestors ok}}^{1 \le i < j \le \#(\overline{J})} \\ \overline{\underline{\Delta'; \mathtt{self}{:}O[\![\overline{P}]\!], \overline{x{:}\tau}; \overline{=P}} \vdash \mu\ \mathsf{ok}} \qquad \{\overline{d}\} = \mathit{allVisible}(\Delta', O[\![\overline{P}]\!]) \\ \overline{\Delta' \vdash d_i \text{ not duplicate of } d_j}^{1 \le i < j \le \#(\overline{d})} \qquad \overline{\Delta' \vdash d_i \text{ meet } d_j \text{ wrt } \{\overline{d}\}\ \mathsf{ok}}^{1 \le i < j \le \#(\overline{d})} \\ \overline{\Delta' \vdash d_i \text{ return type wrt } d_j\ \mathsf{ok}}^{1 \le i < j \le \#(\overline{d})} \qquad \overline{\Delta' \vdash d_j \text{ return type wrt } d_i\ \mathsf{ok}}^{1 \le i < j \le \#(\overline{d})} \\ \overline{\underline{\Delta'} \vdash \tau\ \mathsf{ok}} \qquad \mathit{distinct}(\overline{x}) \end{array}}{\Delta \vdash \mathtt{object}\ O[\![\overline{P <: \{\overline{\xi}\}}]\!](\overline{x{:}\tau}) <: \left\{\overline{t}\right\}\ \overline{\mu}\ \mathsf{end\ ok}}\ [\textsc{D-Object}]$$

Well-formed method declarations: $\boxed{\Delta; \Gamma; \overline{V\ P} \vdash \mu\ \mathsf{ok}}$

$$\frac{\begin{array}{c} \Delta \vdash [\![\overline{\{\overline{\zeta}\} <: Q <: \{\overline{\xi}\}}]\!]\ \mathsf{ok} \qquad \Delta' = \Delta \cup \left\{\overline{\{\overline{\zeta}\} <: Q <: \{\overline{\xi}\}}\right\} \qquad \overline{\overline{\underline{\underline{\Delta'}} \vdash \zeta\ \mathsf{ok}}} \qquad \overline{\overline{\underline{\underline{\Delta'}} \vdash \xi\ \mathsf{ok}}} \\ \mathit{distinct}(\overline{x}) \qquad \overline{\underline{\Delta'} \vdash \tau\ \mathsf{ok}} \qquad \Delta' \vdash \omega\ \mathsf{ok} \qquad \Delta'; \Gamma, \overline{x{:}\tau} \vdash e : (\_, \rho) \qquad \Delta' \vdash \rho <: \omega \\ \mathit{distinct}(\overline{P}, \overline{Q}) \qquad \overline{(V = {=}) \vee (\forall \mathcal{T}.\ ((\underline{(\overline{\tau}) \to \omega}) \ne \mathcal{T}[P]) \vee (\underline{\Delta} \vdash \mathcal{T} \text{ variance } V))} \end{array}}{\Delta; \Gamma; \overline{V\ P} \vdash m[\![\overline{\{\overline{\zeta}\} <: Q <: \{\overline{\xi}\}}]\!](\overline{x{:}\tau}){:}\ \omega = e\ \mathsf{ok}}\ [\textsc{D-Method}]$$

Fig. 3. Well-formed programs, class declarations, and method declarations

Fortress: Symmetric Multiple Dispatch with Polymorphism and Variance

PROVED

Well-formed programs: $\boxed{\vdash \Pi : g}$

$$\frac{\Pi = \overline{\psi}, e \qquad distinct(\overline{name(\psi)}) \qquad \Delta = \{\overline{\psi}\} \qquad \overline{\underline{\Delta} \vdash \psi \text{ ok}} \qquad \Delta; \bullet \vdash e : (\_, g)}{\vdash \Pi : g} \text{ [T-Program]}$$

Well-formed trait and object declarations: $\boxed{\Delta \vdash \psi \text{ ok}}$

$$\frac{\begin{array}{c} \Delta' = \Delta \cup \left\{\overline{\{\} <: P <: \{\overline{\xi}\}}\right\} \qquad distinct(\overline{P}) \qquad \overline{\overline{\underline{\underline{\Delta'}} \vdash \xi \text{ ok}}} \qquad \overline{\underline{\Delta'} \vdash t \text{ ok}} \\ \{\overline{J}\} = properAncestors(\Delta', T[\![\overline{P}]\!]) \qquad \overline{\Delta'; \overline{J} \vdash J_i \text{ and } J_j \text{ ancestors ok}}^{1 \le i < j \le \#(\overline{J})} \\ \overline{\underline{T} \neq name(J)} \qquad \overline{\underline{\Delta'; \mathtt{self}: T[\![\overline{P}]\!]; \overline{V\ P}} \vdash \mu \text{ ok}} \qquad \{\overline{d}\} = allVisible(\Delta', T[\![\overline{P}]\!]) \\ \overline{\Delta' \vdash d_i \text{ not duplicate of } d_j}^{1 \le i < j \le \#(\overline{d})} \qquad \overline{\Delta' \vdash d_i \text{ meet } d_j \text{ wrt } \{\overline{d}\} \text{ ok}}^{1 \le i < j \le \#(\overline{d})} \\ \overline{\Delta' \vdash d_i \text{ return type wrt } d_j \text{ ok}}^{1 \le i < j \le \#(\overline{d})} \qquad \overline{\Delta' \vdash d_j \text{ return type wrt } d_i \text{ ok}}^{1 \le i < j \le \#(\overline{d})} \end{array}}{\Delta \vdash \mathtt{trait}\ T[\![\overline{V\ P <: \{\overline{\xi}\}}]\!] <: \{\overline{t}\}\ \overline{\mu} \text{ end ok}} \text{ [D-Trait]}$$

$$\frac{\begin{array}{c} \Delta' = \Delta \cup \left\{\overline{\{\} <: P <: \{\overline{\xi}\}}\right\} \qquad distinct(\overline{P}) \qquad \overline{\overline{\underline{\underline{\Delta'}} \vdash \xi \text{ ok}}} \qquad \overline{\underline{\Delta'} \vdash t \text{ ok}} \\ \{\overline{J}\} = properAncestors(\Delta', O[\![\overline{P}]\!]) \qquad \overline{\Delta'; \overline{J} \vdash J_i \text{ and } J_j \text{ ancestors ok}}^{1 \le i < j \le \#(\overline{J})} \\ \overline{\underline{\Delta'; \mathtt{self}: O[\![\overline{P}]\!], \overline{x : \tau}; \overline{= P}} \vdash \mu \text{ ok}} \qquad \{\overline{d}\} = allVisible(\Delta', O[\![\overline{P}]\!]) \\ \overline{\Delta' \vdash d_i \text{ not duplicate of } d_j}^{1 \le i < j \le \#(\overline{d})} \qquad \overline{\Delta' \vdash d_i \text{ meet } d_j \text{ wrt } \{\overline{d}\} \text{ ok}}^{1 \le i < j \le \#(\overline{d})} \\ \overline{\Delta' \vdash d_i \text{ return type wrt } d_j \text{ ok}}^{1 \le i < j \le \#(\overline{d})} \qquad \overline{\Delta' \vdash d_j \text{ return type wrt } d_i \text{ ok}}^{1 \le i < j \le \#(\overline{d})} \\ \overline{\underline{\Delta'} \vdash \tau \text{ ok}} \qquad distinct(\overline{x}) \end{array}}{\Delta \vdash \mathtt{object}\ O[\![\overline{P <: \{\overline{\xi}\}}]\!](\overline{x : \tau}) <: \{\overline{t}\}\ \overline{\mu} \text{ end ok}} \text{ [D-Object]}$$

Well-formed method declarations: $\boxed{\Delta; \Gamma; \overline{V\ P} \vdash \mu \text{ ok}}$

$$\frac{\begin{array}{c} \Delta \vdash [\![\overline{\{\overline{\zeta}\} <: Q <: \{\overline{\xi}\}}]\!] \text{ ok} \qquad \Delta' = \Delta \cup \left\{\overline{\{\overline{\zeta}\} <: Q <: \{\overline{\xi}\}}\right\} \qquad \overline{\overline{\underline{\underline{\Delta'}} \vdash \zeta \text{ ok}}} \qquad \overline{\overline{\underline{\underline{\Delta'}} \vdash \xi \text{ ok}}} \\ distinct(\overline{x}) \qquad \overline{\underline{\Delta'} \vdash \tau \text{ ok}} \qquad \Delta' \vdash \omega \text{ ok} \qquad \Delta'; \Gamma, \overline{x : \tau} \vdash e : (\_, \rho) \qquad \Delta' \vdash \rho <: \omega \\ distinct(\overline{P}, \overline{Q}) \qquad \overline{(V = =) \vee (\forall \mathcal{T}. (\underline{((\overline{\tau}) \to \omega)} \neq \mathcal{T}[P]) \vee (\underline{\Delta} \vdash \mathcal{T} \text{ variance } V))} \end{array}}{\Delta; \Gamma; \overline{V\ P} \vdash m[\![\overline{\{\overline{\zeta}\} <: Q <: \{\overline{\xi}\}}]\!](\overline{x : \tau}) \colon \omega = e \text{ ok}} \text{ [D-Method]}$$

Fig. 3. Well-formed programs, class declarations, and method declarations

Well-formed ancestors: $\boxed{\Delta; \overline{J} \vdash J \text{ and } J \text{ ancestors ok}}$

$$\frac{T \neq T'}{\Delta; \overline{J} \vdash T[\![\overline{\alpha}]\!] \text{ and } T'[\![\overline{\eta}]\!] \text{ ancestors ok}} \text{ [Anc-Diff-Trait]}$$

$$\frac{T[\![\overline{\gamma}]\!] \in \{\overline{J}\} \qquad \Delta \vdash T[\![\overline{\gamma}]\!] <: T[\![\overline{\alpha}]\!] \qquad \Delta \vdash T[\![\overline{\gamma}]\!] <: T[\![\overline{\eta}]\!]}{\Delta; \overline{J} \vdash T[\![\overline{\alpha}]\!] \text{ and } T[\![\overline{\eta}]\!] \text{ ancestors ok}} \text{ [Anc-Same-Trait]}$$

Well-formed type parameter bindings: $\boxed{\Delta \vdash [\![\overline{K}]\!] \text{ ok}}$

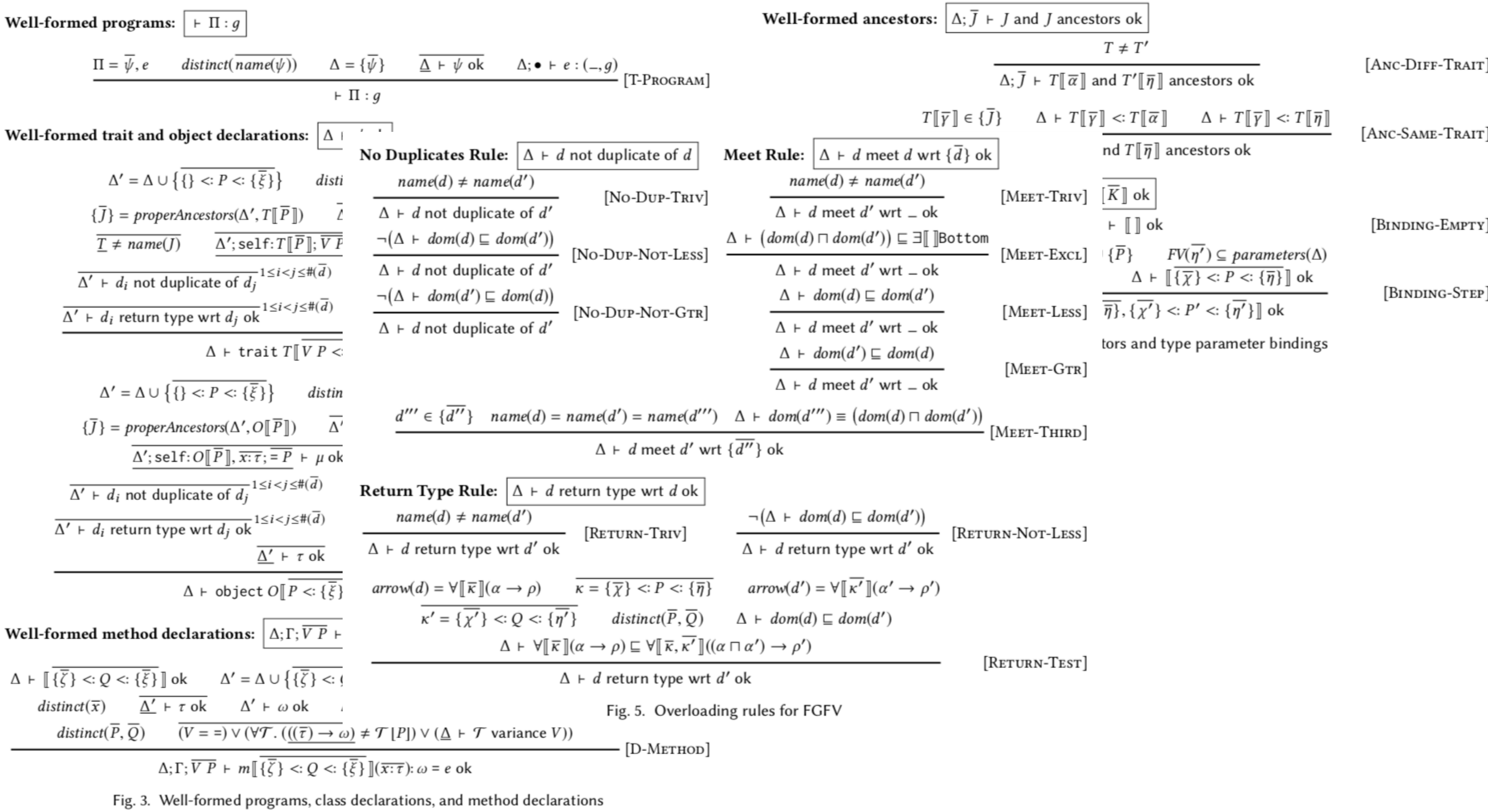
$$\Delta \vdash [\![\,]\!] \text{ ok} \qquad \text{[Binding-Empty]}$$

$$\frac{\begin{array}{c} FV(\overline{\chi'}) \subseteq parameters(\Delta) \cup \{\overline{P}\} \qquad FV(\overline{\eta'}) \subseteq parameters(\Delta) \\ \Delta \vdash \bigsqcup \{\overline{\chi'}\} <: \bigsqcap \{\overline{\eta'}\} \qquad \Delta \vdash [\![\overline{\{\overline{\chi}\} <: P <: \{\overline{\eta}\}}]\!] \text{ ok} \end{array}}{\Delta \vdash [\![\overline{\{\overline{\chi}\} <: P <: \{\overline{\eta}\}}, \{\overline{\chi'}\} <: P' <: \{\overline{\eta'}\}]\!] \text{ ok}} \text{ [Binding-Step]}$$

Fig. 4. Well-formed ancestors and type parameter bindings

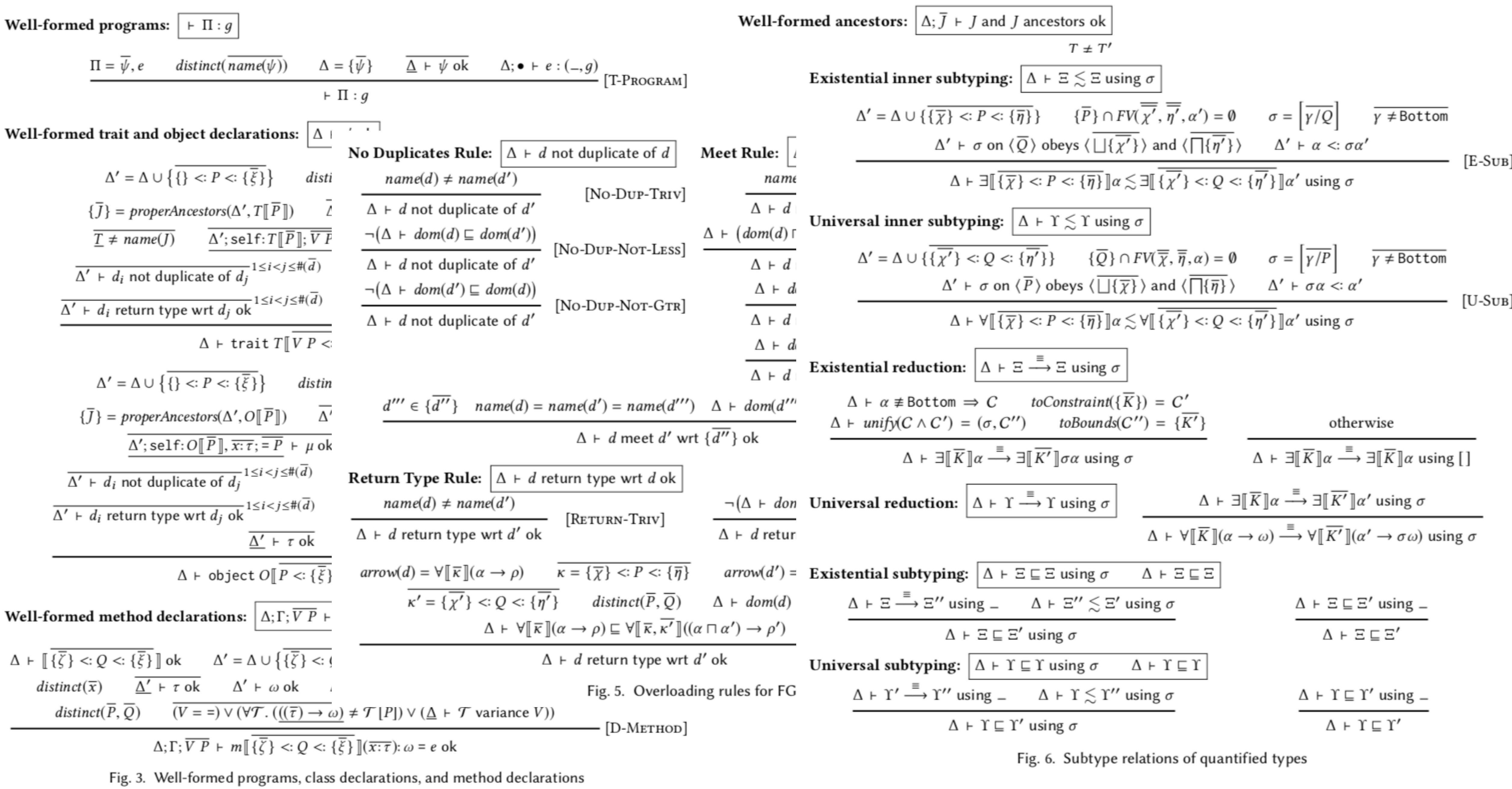
Fortress: Symmetric Multiple Dispatch with Polymorphism and Variance

PROVED



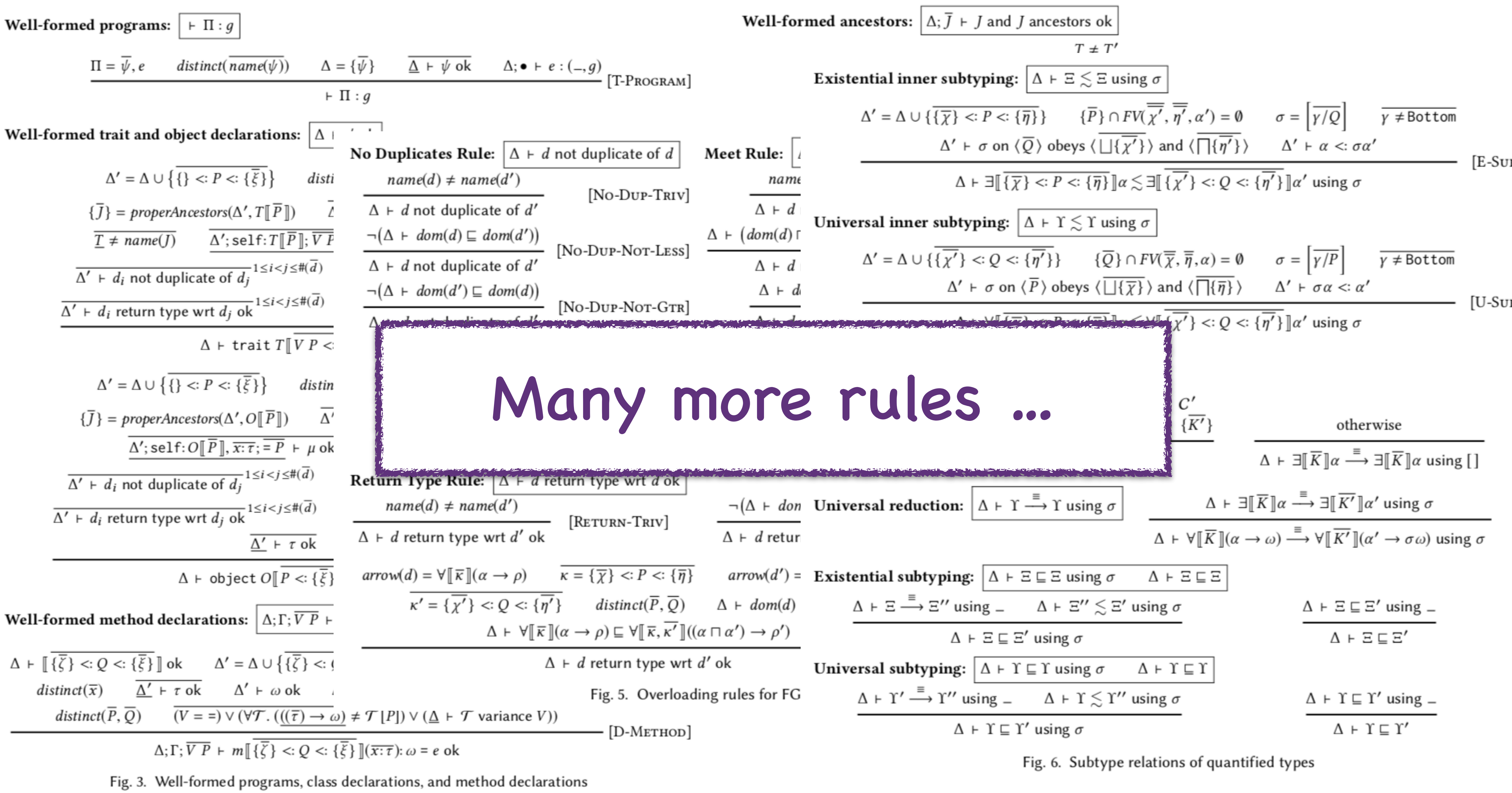
Fortress: Symmetric Multiple Dispatch with Polymorphism and Variance

PROVED



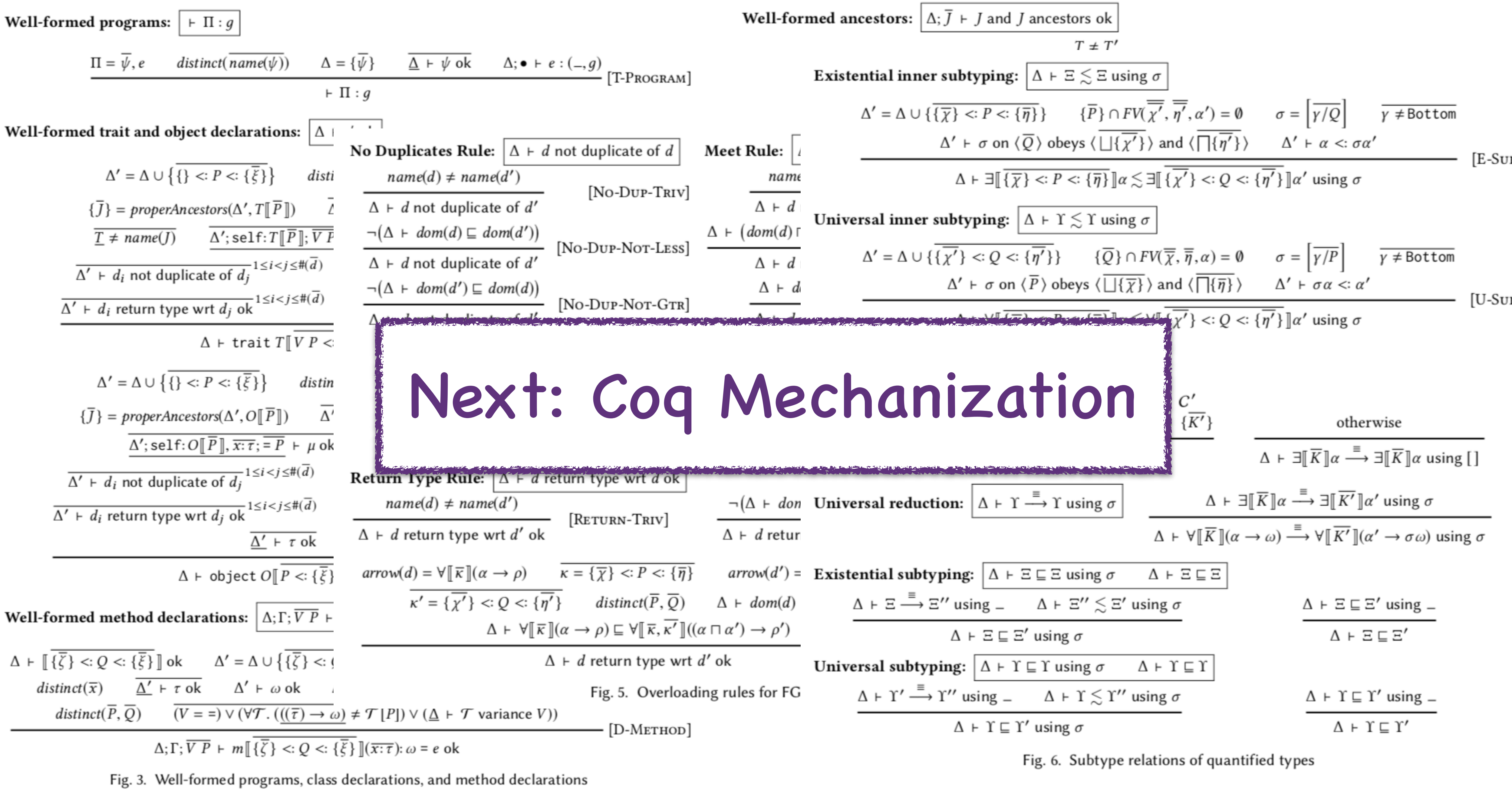
Fortress: Symmetric Multiple Dispatch with Polymorphism and Variance

PROVED



Fortress: Symmetric Multiple Dispatch with Polymorphism and Variance

PROVED



Next: Coq Mechanization

Fortress: Symmetric Multiple Dispatch with Polymorphism and Variance

PROVED

No Duplicates Rule: $\boxed{\Delta \vdash d \text{ not duplicate of } d}$

$$\frac{name(d) \neq name(d')}{\Delta \vdash d \text{ not duplicate of } d'}\ [\text{No-Dup-Triv}]$$

$$\frac{\neg(\Delta \vdash dom(d) \sqsubseteq dom(d'))}{\Delta \vdash d \text{ not duplicate of } d'}\ [\text{No-Dup-Not-Less}]$$

$$\frac{\neg(\Delta \vdash dom(d') \sqsubseteq dom(d))}{\Delta \vdash d \text{ not duplicate of } d'}\ [\text{No-Dup-Not-Gtr}]$$

Meet Rule: $\boxed{\Delta \vdash d \text{ meet } d \text{ wrt } \{\overline{d}\} \text{ ok}}$

$$\frac{name(d) \neq name(d')}{\Delta \vdash d \text{ meet } d' \text{ wrt } \_ \text{ ok}}\ [\text{Meet-Triv}]$$

$$\frac{\Delta \vdash (dom(d) \sqcap dom(d')) \sqsubseteq \exists [\![\,]\!] \texttt{Bottom}}{\Delta \vdash d \text{ meet } d' \text{ wrt } \_ \text{ ok}}\ [\text{Meet-Excl}]$$

$$\frac{\Delta \vdash dom(d) \sqsubseteq dom(d')}{\Delta \vdash d \text{ meet } d' \text{ wrt } \_ \text{ ok}}\ [\text{Meet-Less}]$$

$$\frac{\Delta \vdash dom(d') \sqsubseteq dom(d)}{\Delta \vdash d \text{ meet } d' \text{ wrt } \_ \text{ ok}}\ [\text{Meet-Gtr}]$$

$$\frac{d''' \in \{\overline{d''}\} \quad name(d) = name(d') = name(d''') \quad \Delta \vdash dom(d''') \equiv (dom(d) \sqcap dom(d'))}{\Delta \vdash d \text{ meet } d' \text{ wrt } \{\overline{d''}\} \text{ ok}}\ [\text{Meet-Third}]$$

Return Type Rule: $\boxed{\Delta \vdash d \text{ return type wrt } d \text{ ok}}$

$$\frac{name(d) \neq name(d')}{\Delta \vdash d \text{ return type wrt } d' \text{ ok}}\ [\text{Return-Triv}]$$

$$\frac{\neg(\Delta \vdash dom(d) \sqsubseteq dom(d'))}{\Delta \vdash d \text{ return type wrt } d' \text{ ok}}\ [\text{Return-Not-Less}]$$

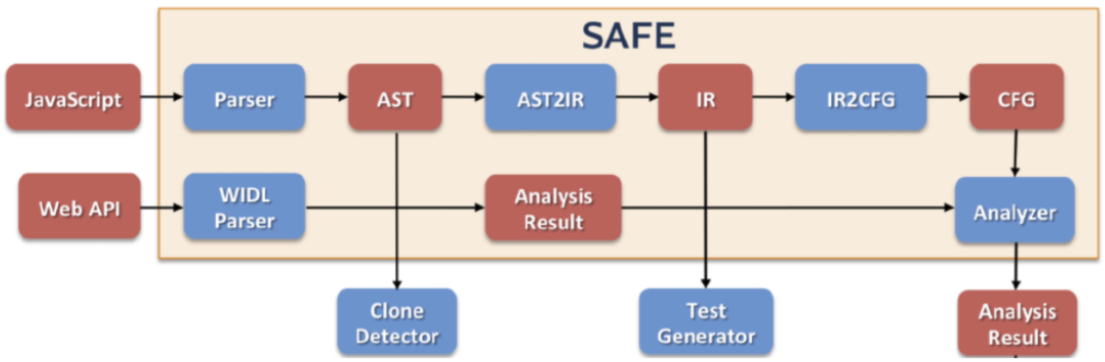
$$\frac{\begin{array}{c} arrow(d) = \forall [\![\overline{\kappa}]\!](\alpha \to \rho) \quad \overline{\kappa = \{\overline{\chi}\} <: P <: \{\overline{\eta}\}} \quad arrow(d') = \forall [\![\overline{\kappa'}]\!](\alpha' \to \rho') \\ \overline{\kappa' = \{\overline{\chi'}\} <: Q <: \{\overline{\eta'}\}} \quad distinct(\overline{P}, \overline{Q}) \quad \Delta \vdash dom(d) \sqsubseteq dom(d') \\ \Delta \vdash \forall [\![\overline{\kappa}]\!](\alpha \to \rho) \sqsubseteq \forall [\![\overline{\kappa}, \overline{\kappa'}]\!]((\alpha \sqcap \alpha') \to \rho') \end{array}}{\Delta \vdash d \text{ return type wrt } d' \text{ ok}}\ [\text{Return-Test}]$$

Fig. 5. Overloading rules for FGFV

Language *Manipulation*: JavaScript

- Specification: ECMAScript
- Parsing: automatic generation of parsers & ASTs
- Static checking: parallel development
- Compilation / Interpretation: cross validation
- Testing
- Analysis: SAFE, TAJS, WALA
- Verification

Analyzing JS



`module` $M\ \{s \cdots\}$	module definition
`module` $M = M \cdots .M$`;`	module alias
`import` $M \cdots .x$`;`	qualified import
`import` $M \cdots .x$`:` x`;`	aliased import
`import` $M \cdots .*$`;`	import all
`export var` $x\ [= e]$`;`	exported variable
`export function` $x\,(x \cdots)\ \{s \cdots\}$	exported function
`export module` $M\ \{s \cdots\}$	exported module
`export module` $M = M \cdots .M$`;`	exported module alias
`export` x`;`	exported local
`export` x`:` x`;`	exported local alias
`export` x`:` $M \cdots .x$`;`	exported qualified alias

Figure 9. Extended syntax for JavaScript modules

v	$::= \cdots \mid \alpha$	value
α	$::= \langle\!\langle v_g \rangle\!\rangle$	accessor
v_g	$::=$ func(){ return e }	getter

Figure 10. Extended syntax for λ_{JS} to allow accessors

ϕ	$::= x \cdots$	path
φ_i	$::= \phi.(x)$	internal qualified name
φ_e	$::= \phi.x$	external qualified name
φ	$::= \varphi_i \mid \varphi_e$	qualified name
$\bar{\varphi}$	$::= \phi.*$	expanded qualified name
τ	$::=$ var $\mid$ module	desugaring type
ς	$::= \epsilon \mid$ local $\mid$ export φ_e	desugaring scope
ρ	$::= \bot \mid \tau\,\varphi$	desugaring binding
$\bar{\rho}$	$::= \bot \mid \top$	expanded desugaring binding
Σ	$::= \{(\varphi, \rho\,\varsigma) \cdots\} \cup \{(\bar{\varphi}, \bar{\rho}) \cdots\}$	desugaring environment
Σ^*	$::= \epsilon \mid \Sigma^*\ \Sigma \mid \Sigma^*\ x$	desugaring environment chain

Figure 12. Desugaring environment for the modified λ_{JS}

Hongki Lee, Sooncheol Won, Joonho Jin, Junhee Cho, and Sukyoung Ryu. **SAFE: Formal Specification and Implementation of a Scalable Analysis Framework for ECMAScript** (FOOL'12)

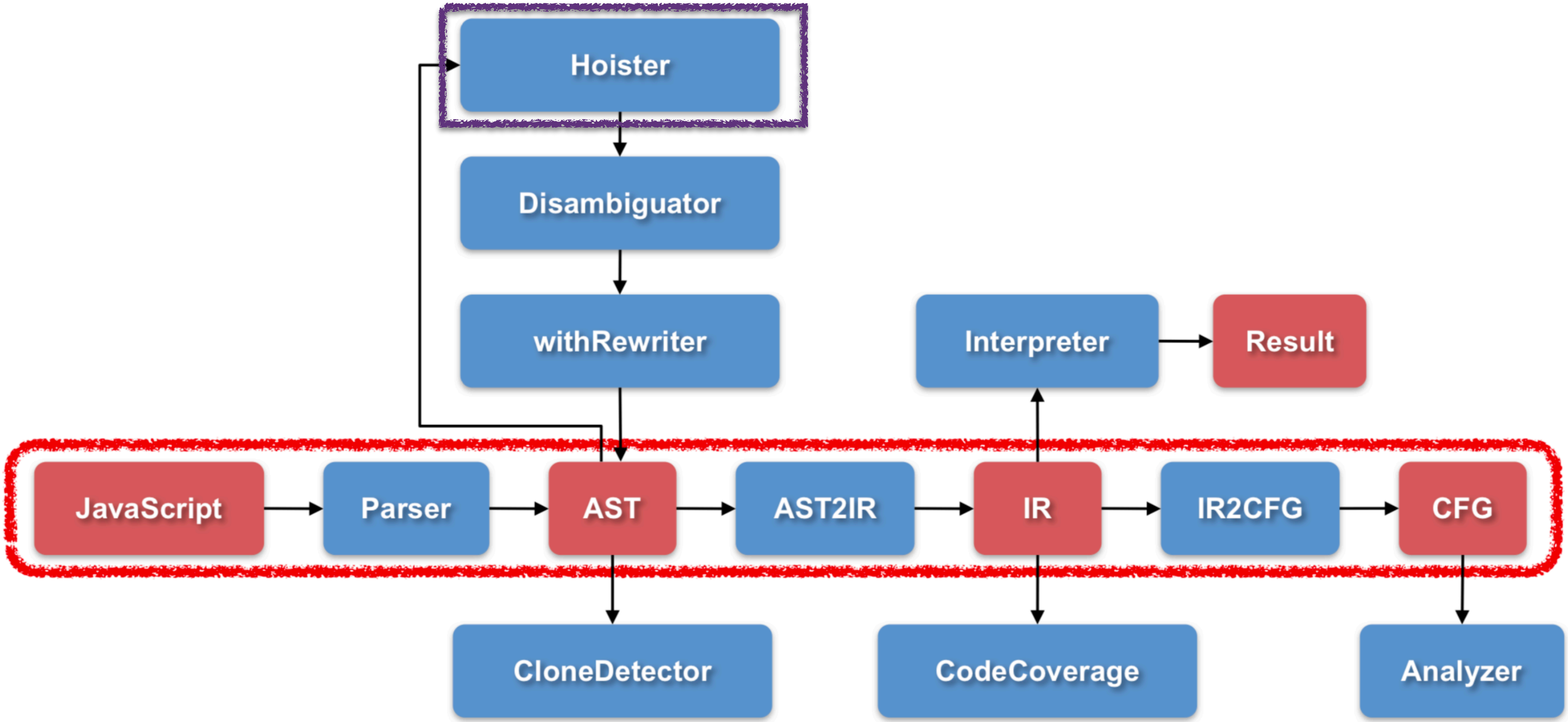
Seonghoon Kang and Sukyoung Ryu. **Formal Specification of a JavaScript Module System** (OOPSLA'12)

Changhee Park, Hongki Lee, and Sukyoung Ryu. **All about the "with" Statement in JavaScript: Removing "with" Statements in JavaScript Applications** (DLS'13)

WaiTing Cheung, Sukyoung Ryu, **Sunghun Kim**. **Development Nature Matters: An Empirical Study of Code Clones in JavaScript Applications** (EMSE'15)

Analyzing JS

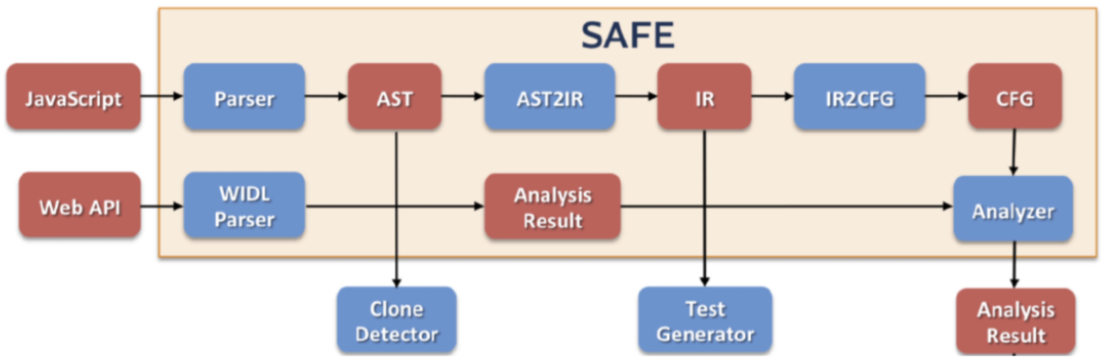
`module` $M\ \{s \cdots\}$	module definition	$\phi\ ::= x \cdots$	path
`module` $M = M \cdots .M$;	module alias	$\varphi_i\ ::= \phi.(x)$	internal qualified name
`import` $M \cdots .x$;	qualified import	$\varphi_e\ ::= \phi.x$	external qualified name

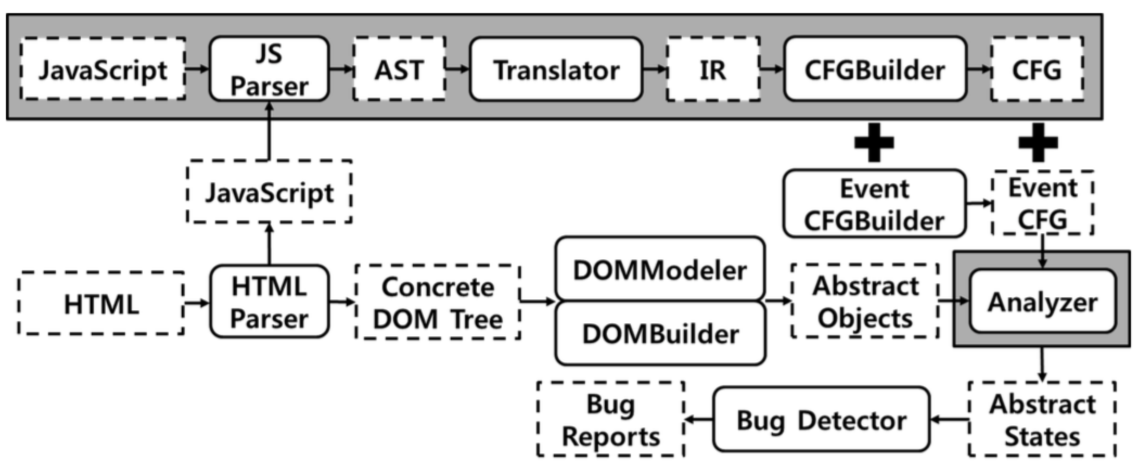


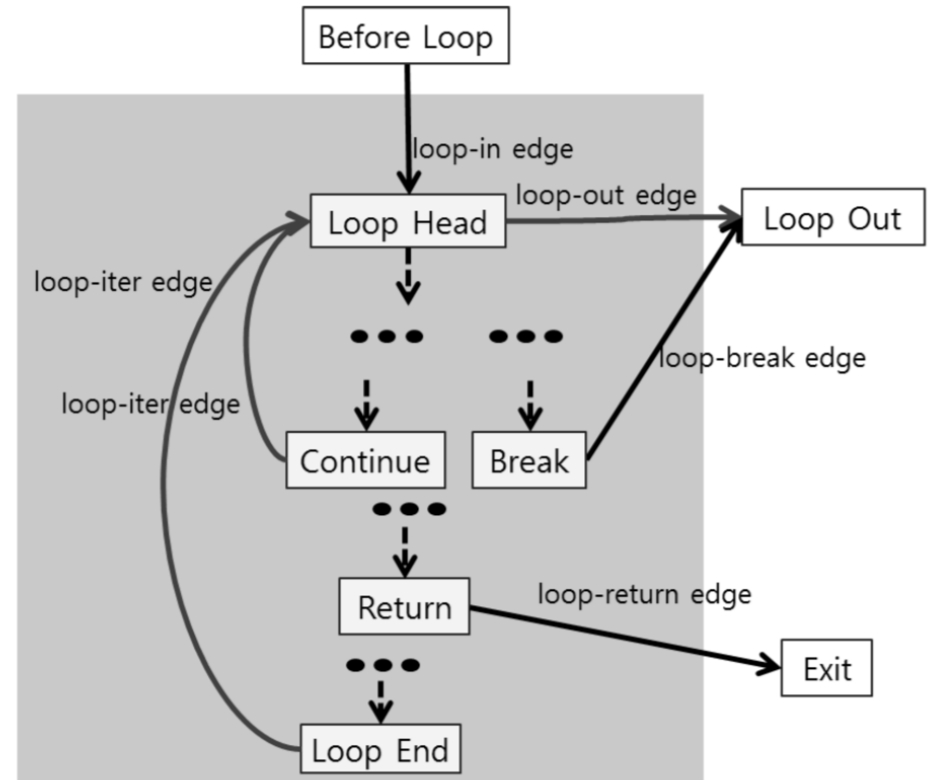
Changhee Park, Hongki Lee, and Sukyoung Ryu. **All about the "with" Statement in JavaScript: Removing "with" Statements in JavaScript Applications** (DLS'13)

WaiTing Cheung, Sukyoung Ryu, **Sunghun Kim**. **Development Nature Matters: An Empirical Study of Code Clones in JavaScript Applications** (EMSE'15)

Analyzing JS Web Apps



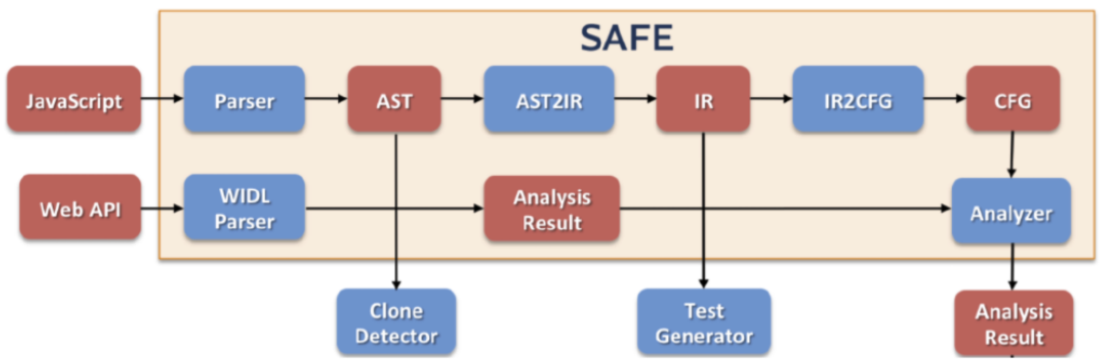


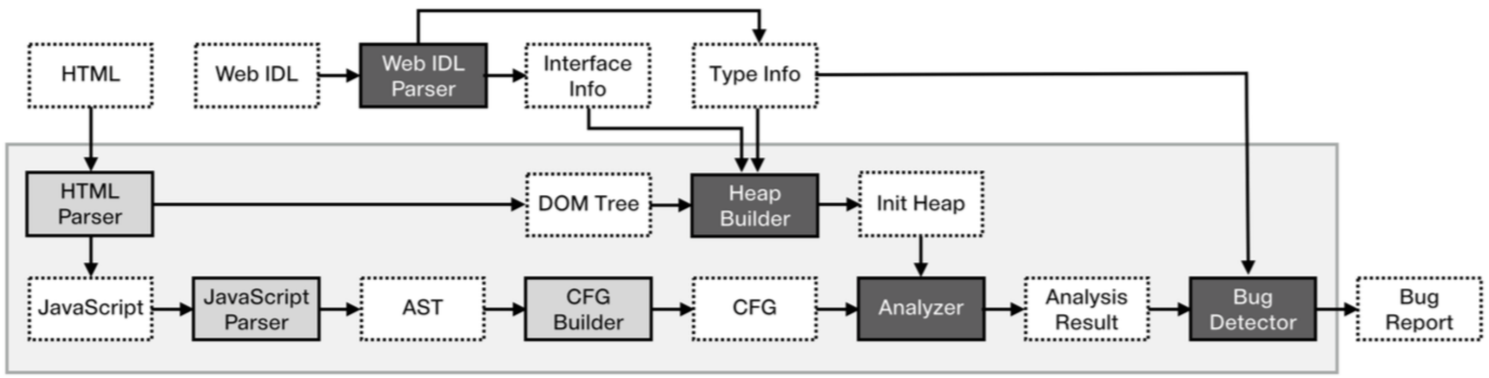


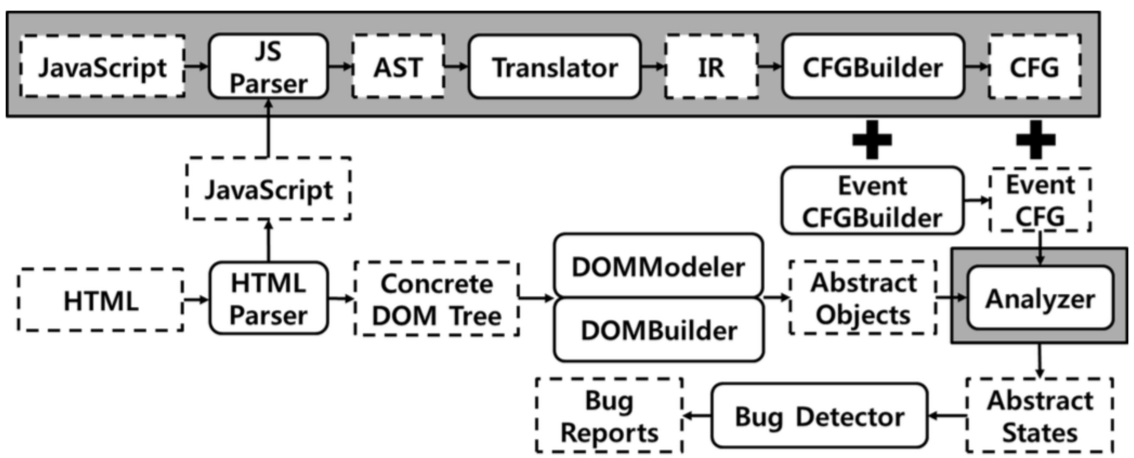
Changhee Park, Sooncheol Won, Joonho Jin, and Sukyoung Ryu. **Static Analysis of JavaScript Web Applications in the Wild via Practical DOM Modeling** (ASE'15)

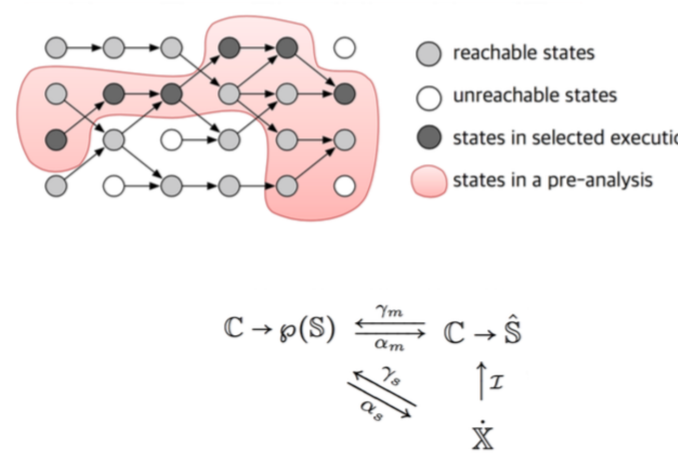
Changhee Park and Sukyoung Ryu. **Scalable and Precise Static Analysis of JavaScript Applications via Loop-Sensitivity** (ECOOP'15)

Analyzing JS Web Apps in the Wild







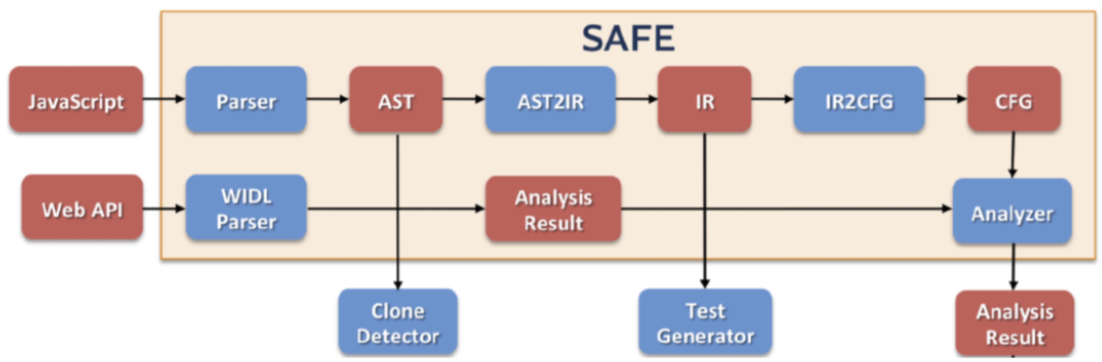


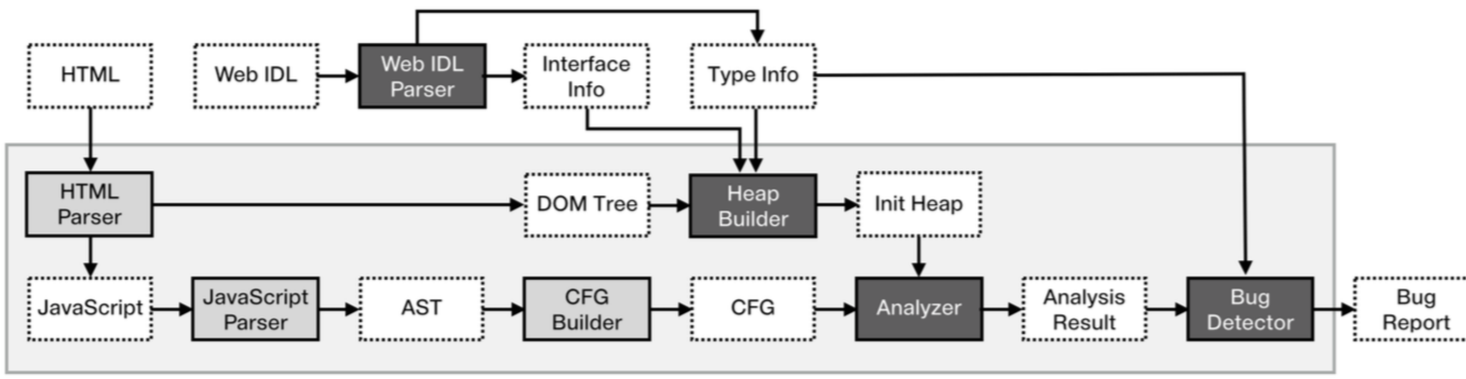
	Subject (LOC)	SAFE	SWP	SWF
Benchmark	RayTrace (679)	✗	576.3 (4.8/571.5)	600.3 (2.9/597.4)
	Richards (288)	2.1	3.7 (2.4/1.3)	3.9 (2.2/1.7)
	Splay (205)	9.2	8.6 (2.1/6.5)	15.8 (1.8/14.0)
	NavierStokes (331)	1.0	4.4 (3.1/1.3)	3.2 (2.0/1.2)
	Box2dWeb (10918)	45.0	58.5 (15.6/42.9)	48.2 (45.4/2.8)
Library	jQuery (6206)	✗	✗	28.6 (23.9/4.7)
	MooTools (4022)	✗	✗	27.6 (25.7/1.9)
	Prototype (5914)	✗	✗	26.2 (24.2/2.0)
	YUI (7181)	✗	8.6 (6.9/1.7)	18.7 (13.7/5.0)
	Underscore (1065)	✗	✗	5.2 (4.6/0.6)
Website	live.com (6325)	240.8	✗	128.0 (32.2/95.8)
	wikipedia.org (291)	0.1	5.8 (5.8/0.0)	5.8 (5.8/0.0)
	facebook.com (5584)	✗	✗	64.1 (62.1/2.0)
	youtube.com (5061)	✗	✗	382.6 (247.6/135.0)
	baidu.com (9739)	✗	✗	119.0 (100.2/18.8)

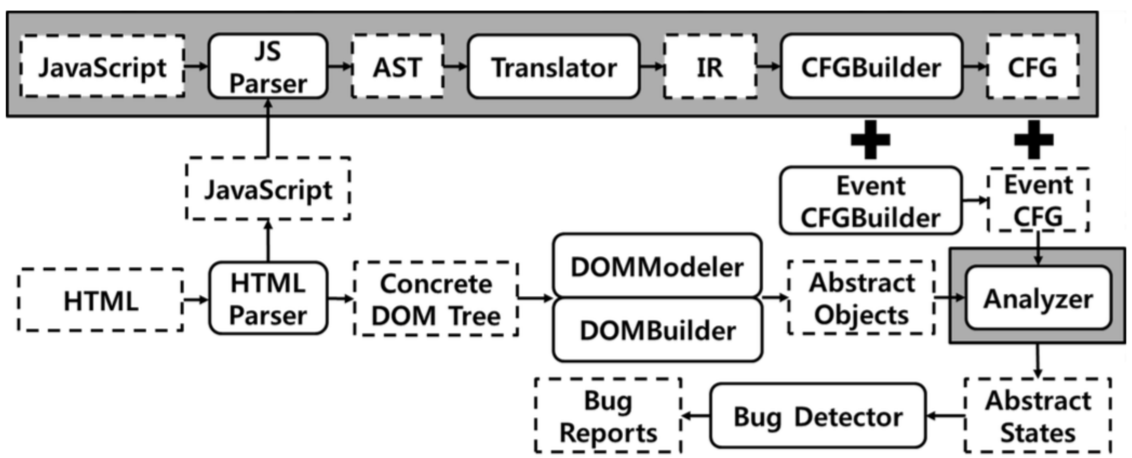
SungGyeong Bae, **Hyunghun Cho**, **Inho Lim**, and Sukyoung Ryu. **SAFEWAPI: Web API Misuse Detector for Web Applications** (FSE'14)

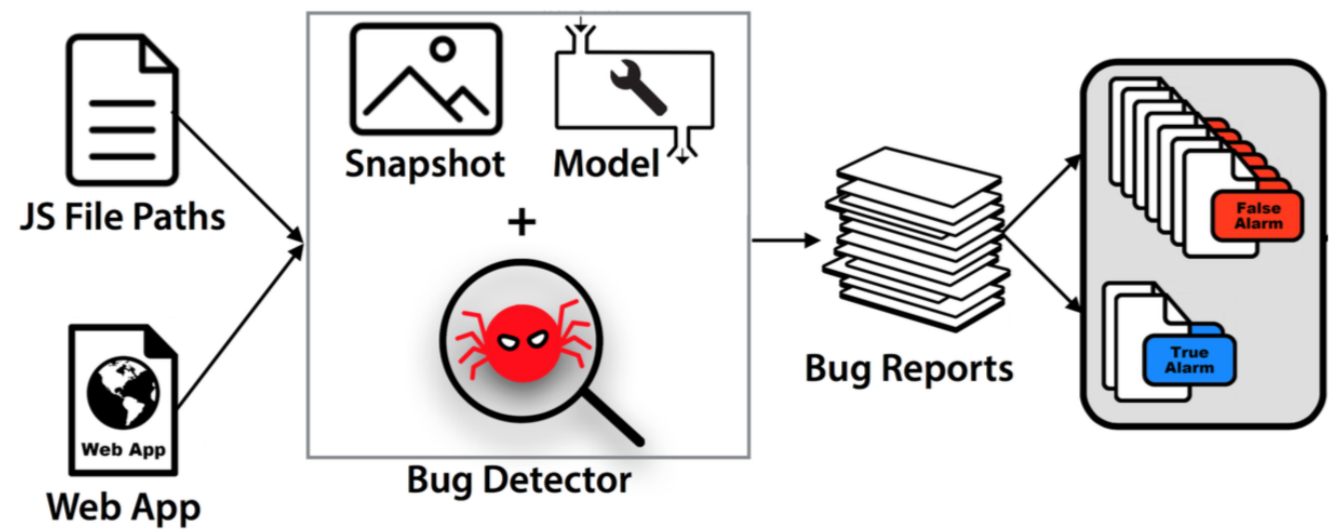
Yoonseok Ko, Hongki Lee, **Julian Dolby**, and Sukyoung Ryu. **Practically Tunable Static Analysis Framework for Large-Scale JavaScript Applications** (ASE'15)

Analyzing JS Web Apps in the Wild Partially









Joonyoung Park, **Inho Lim**, and Sukyoung Ryu. **Battles with False Positives in Static Analysis of JavaScript Web Applications in the Wild** (ICSE-SEIP'16)

Joonyoung Park, **Kwangwon Sun**, and Sukyoung Ryu. **EventHandler-based Analysis Framework for Web Apps using Dynamically Collected States** (FASE'18)

Analyzing JS Web Apps in the Wild Partially

JavaScript Bug Detection

```
3d-raytrace.js:118:19~118:23: [Warning] Trying to convert undefined to number. 'self[3]' can be undefined.
3d-raytrace.js:118:19~118:23: [Warning] Reading absent property '3' of object 'self'.
3d-raytrace.js:119:19~119:23: [Warning] Trying to convert undefined to number. 'self[7]' can be undefined.
3d-raytrace.js:119:19~119:23: [Warning] Reading absent property '7' of object 'self'.
3d-raytrace.js:120:19~120:24: [Warning] Trying to convert undefined to number. 'self[11]' can be undefined.
3d-raytrace.js:120:19~120:24:
```

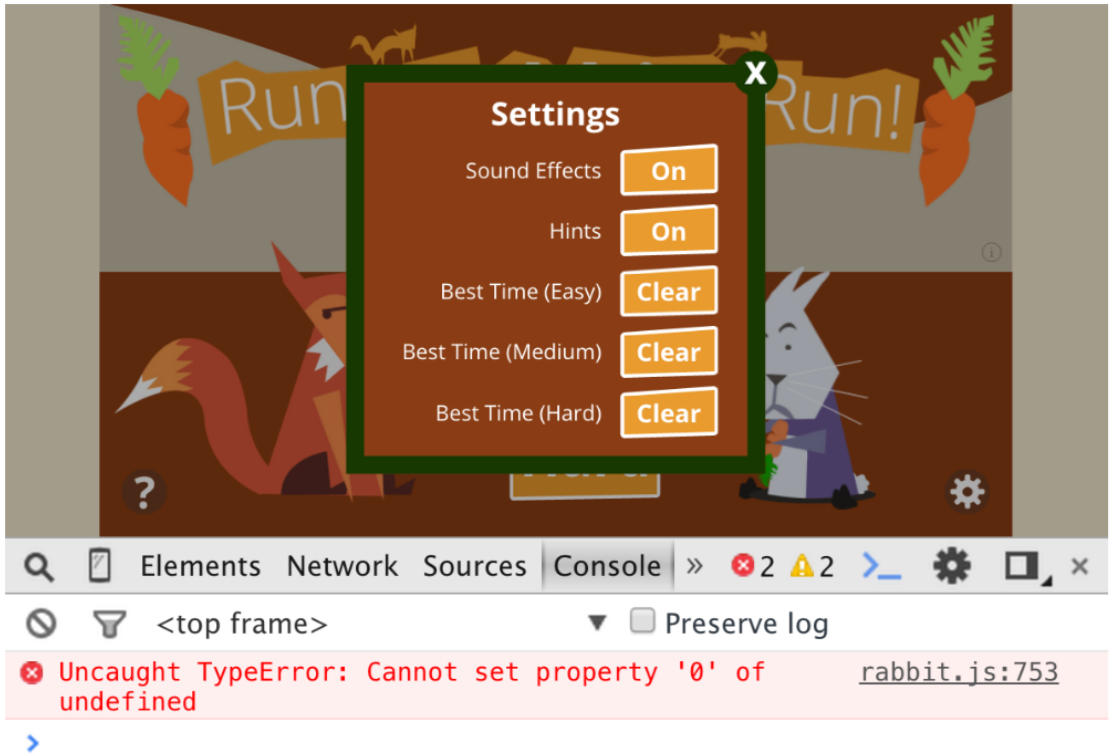
```
function invertMatrix(self
    var temp = new Array(1
    var tx = -self[3];
    var ty = -self[7];
    var tz = -self[11];
    for (h = 0; h < 3; h++
        for (v = 0; v < 3;
            temp[h + v * 4
    for (i = 0; i < 11; i+
        self[i] = temp[i];
    self[3] = tx * self[0]
    self[7] = tx * self[4]
    self[11] = tx * self[8
```

```
// this camera code is from notes i made ages ago, it is from
*somewhere* -- i cannot remember where
// that somewhere is
function Camera(origin, lookat, up) {
    var zaxis = normaliseVector(subVector(lookat, origin));
    var xaxis = normaliseVector(cross(up, zaxis));
    var yaxis = normaliseVector(cross(xaxis, subVector([0,0,0]
, zaxis)));
    var m = new Array(16);
    m[0] = xaxis[0]; m[1] = xaxis[1]; m[2] = xaxis[2];
    m[4] = yaxis[0]; m[5] = yaxis[1]; m[6] = yaxis[2];
    m[8] = zaxis[0]; m[9] = zaxis[1]; m[10] = zaxis[2];
    invertMatrix(m);
    m[3] = 0; m[7] = 0; m[11] = 0;
    this.origin = origin;
```

Web Application Bug Detection (I)

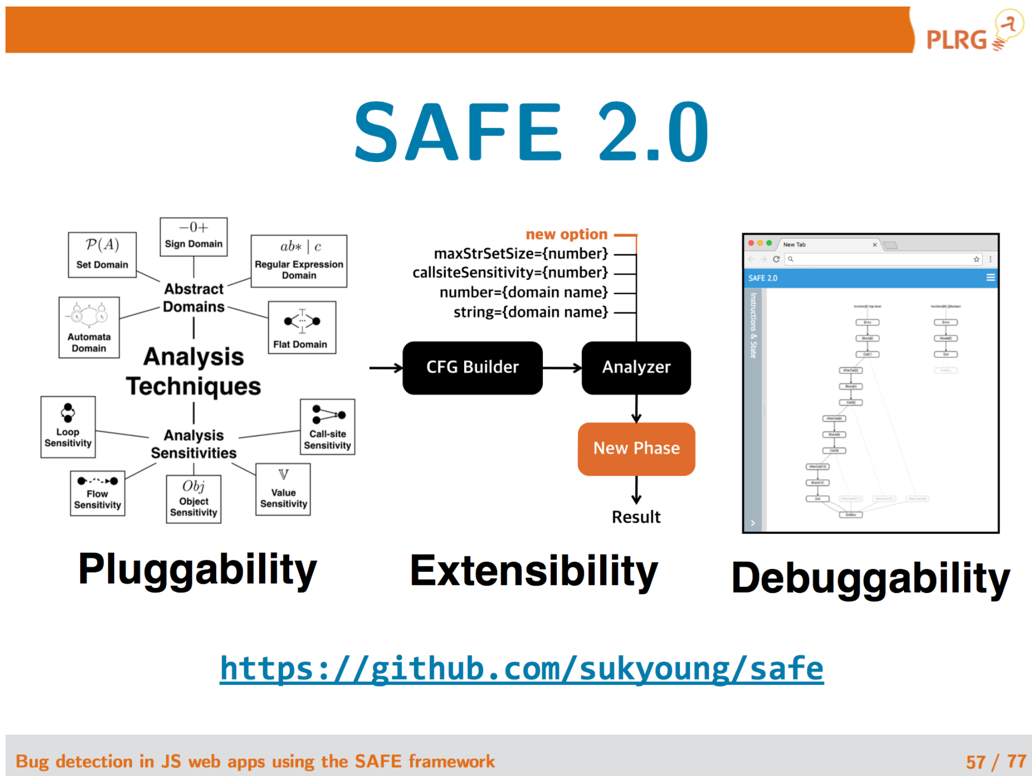
```
loading : function(i) {
        // alert("<img src='image/loading_"+i+".png>");
        var tmpdivB = document.getElementById("waitIcon");

        i = i == null ? 1 : i;

        // set timeout is 30s
        this.totalCnt++;
        if (this.totalCnt > 200) {
                this.endLoading();
                tmpdivB.className = "";
                this.showNoResultFound(tmpdivB);
                return;
        }
```

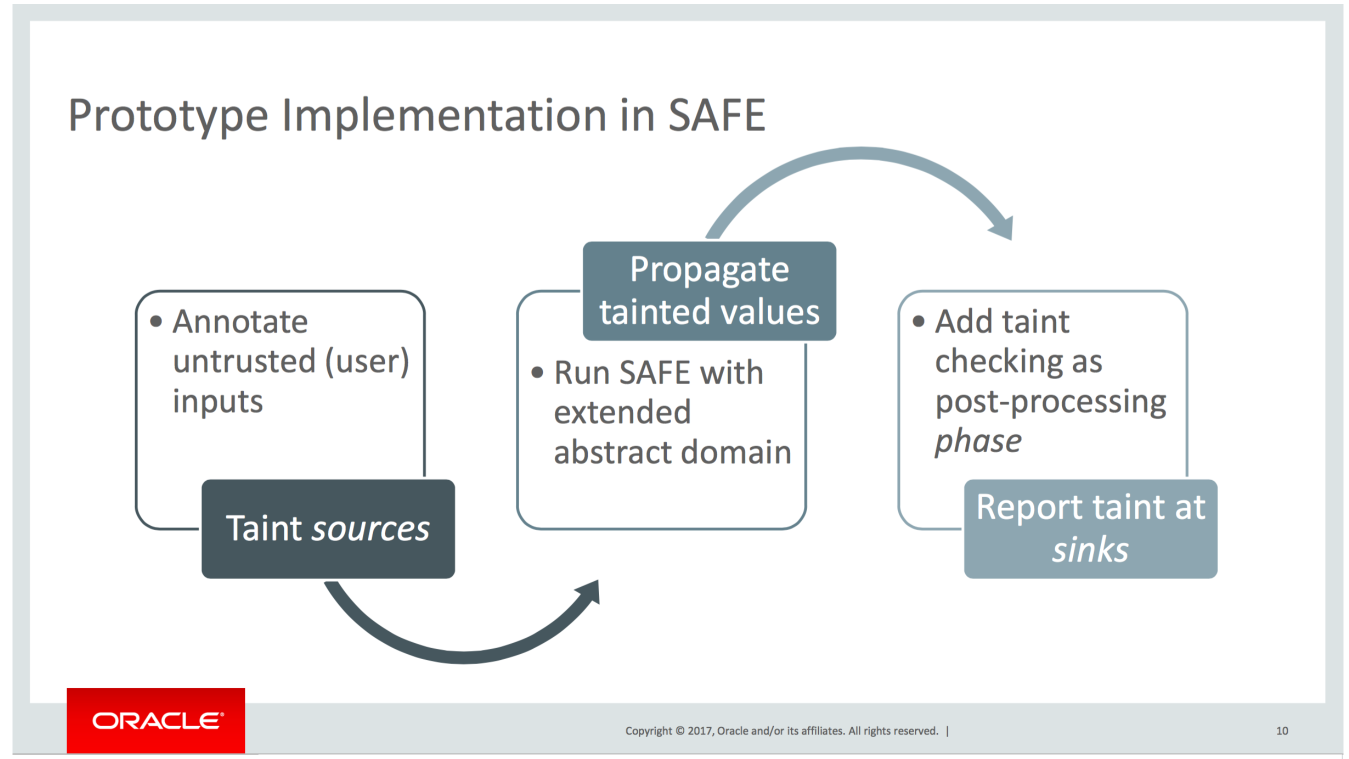
```
<div id="waitIconDiv" class="waitIconType" style="display:none;">
    <img src = "css/images/009.gif" width = "50px" height = "50px"/>
</div>
```

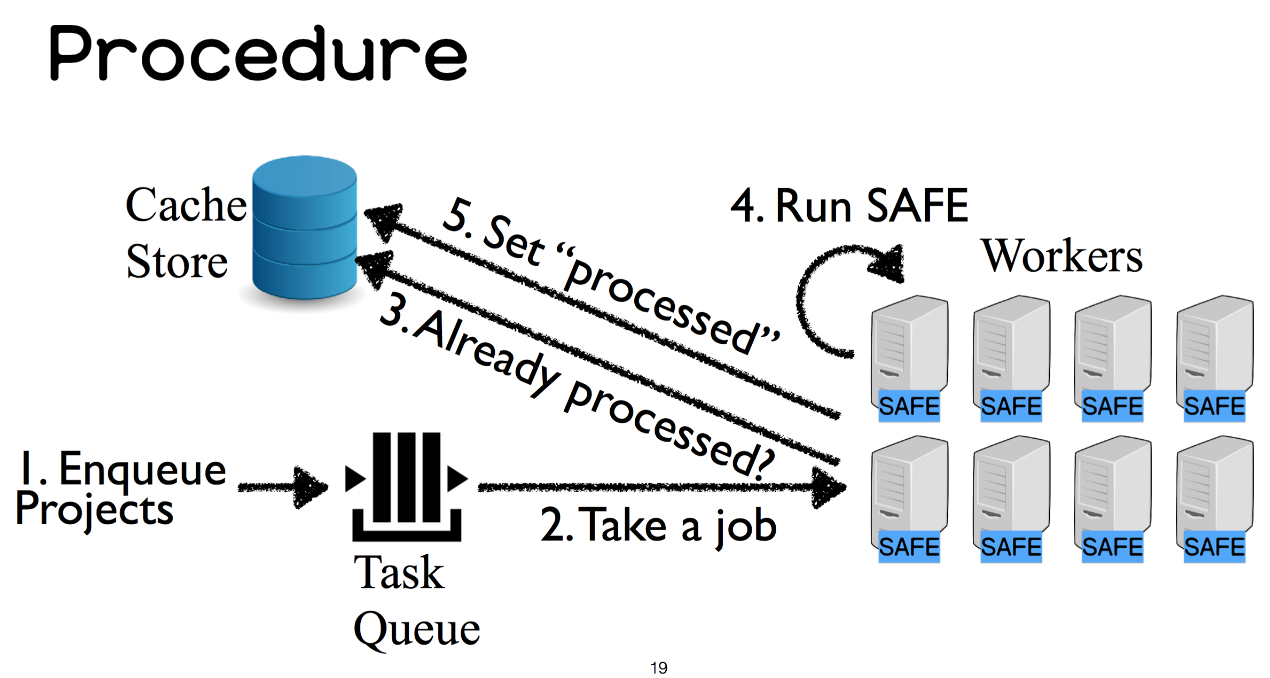


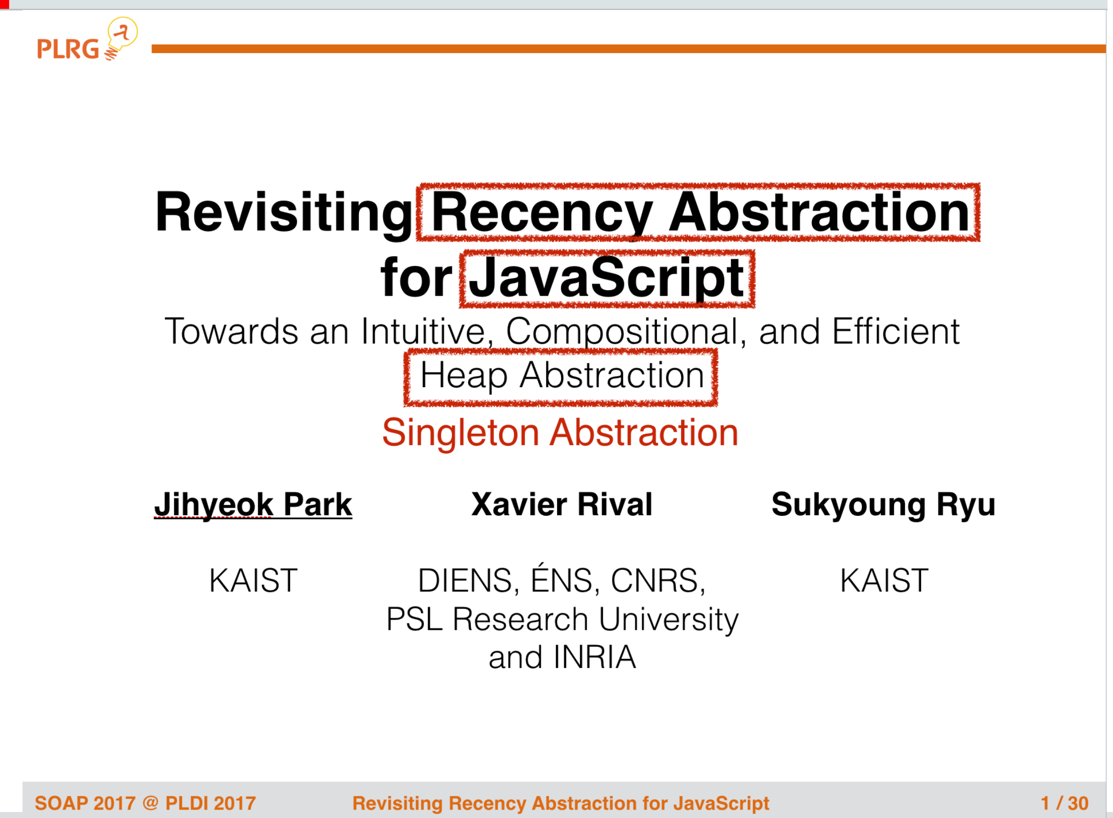
Web Page Bug Detection (I)

Bug Detection in JS Web Apps with SAFE









Now, Solidity!

coindesk

The DAO Attacked: Code Issue Leads to $60 Million Ether Theft

in 466

Michael del Castillo

Jun 17, 2016 at 14:00 UTC | Updated Jun 18, 2016 at 14:46 UTC

NEWS

The DAO, the distributed autonomous organization that had collected over $150m worth of the cryptocurrency ether, has reportedly been hacked, sparking a broad market sell-off.

A leaderless organization comprised of a series of smart contracts written on the ethereum codebase, The DAO has lost 3.6m ether, which is currently sitting in a separate wallet after being split off into a separate grouping dubbed a "child DAO"

MailOnline | TV&Showbiz | Femail | Health | Science | Money | Video

Discounts

£200 million worth of digital cryptocurrency is wiped out as bungling developer locks investors out while trying to stop hackers

- A developer was fixing a bug that let hackers steal funds from virtual wallets
- But the developer accidentally left a second flaw in its systems
- When the user tried to undo the damage by deleting the flaw in the code, this locked the funds in the wallets permanently
- The only way to reverse the issue is a 'hard-fork', but not everyone supports this

http://www.dailymail.co.uk/sciencetech/article-5062543/200-MILLION-virtual-currency-Ether-lost.html

https://www.coindesk.com/dao-attacked-code-issue-leads-60-million-ether-theft/

BBC

Code bug freezes $150m of Ethereum crypto-cash

9 November 2017

Share

http://www.bbc.com/news/technology-41928147

Hackers Have Walked Off With About 14% of Big Digital Currencies

By Olga Kharif

January 18, 2018, 7:19 PM GMT+5:30

→ Cybercriminals compromise Bitcoin, Ether supply, blockchains

→ Crypto-crazed users adopt technology without weighing risks

https://www.bloomberg.com/news/articles/2018-01-18/hackers-have-walked-off-with-about-14-of-big-digital-currencies

Smart Contract Vulnerabilities

Schneier on Security

Blog | Newsletter | Books | Essays | News | Talks | Academic | About Me

Blog >

Ethereum Hacks

The press is reporting a $32M theft of the cryptocurrency Ethereum. Like all such thefts, they're not a result of a cryptographic failure in the currencies, but instead a software vulnerability in the software surrounding the currency -- in this case, digital wallets.

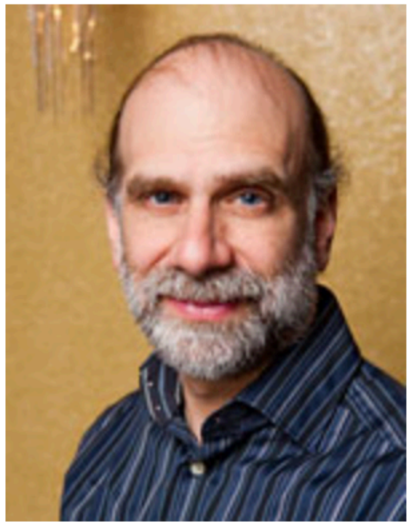
This is the second Ethereum hack this week. The first tricked people in sending their Ethereum to another address.

This is my concern about digital cash. The cryptography can be bulletproof, but the computer security will always be an issue.

Tags: cryptocurrency, cryptography, hacking, theft, vulnerabilities

Posted on July 20, 2017 at 9:12 AM • 46 Comments

About Bruce Schneier



I've been writing about security issues on my blog since 2004, and in my monthly newsletter since 1998. I write books, articles, and academic papers. Currently, I'm the Chief Technology Officer of IBM Resilient, a fellow at Harvard's Berkman Center, and a board member of EFF.

Smart Contract Vulnerabilities: Research

0s: Stop and Arithmetic Operations

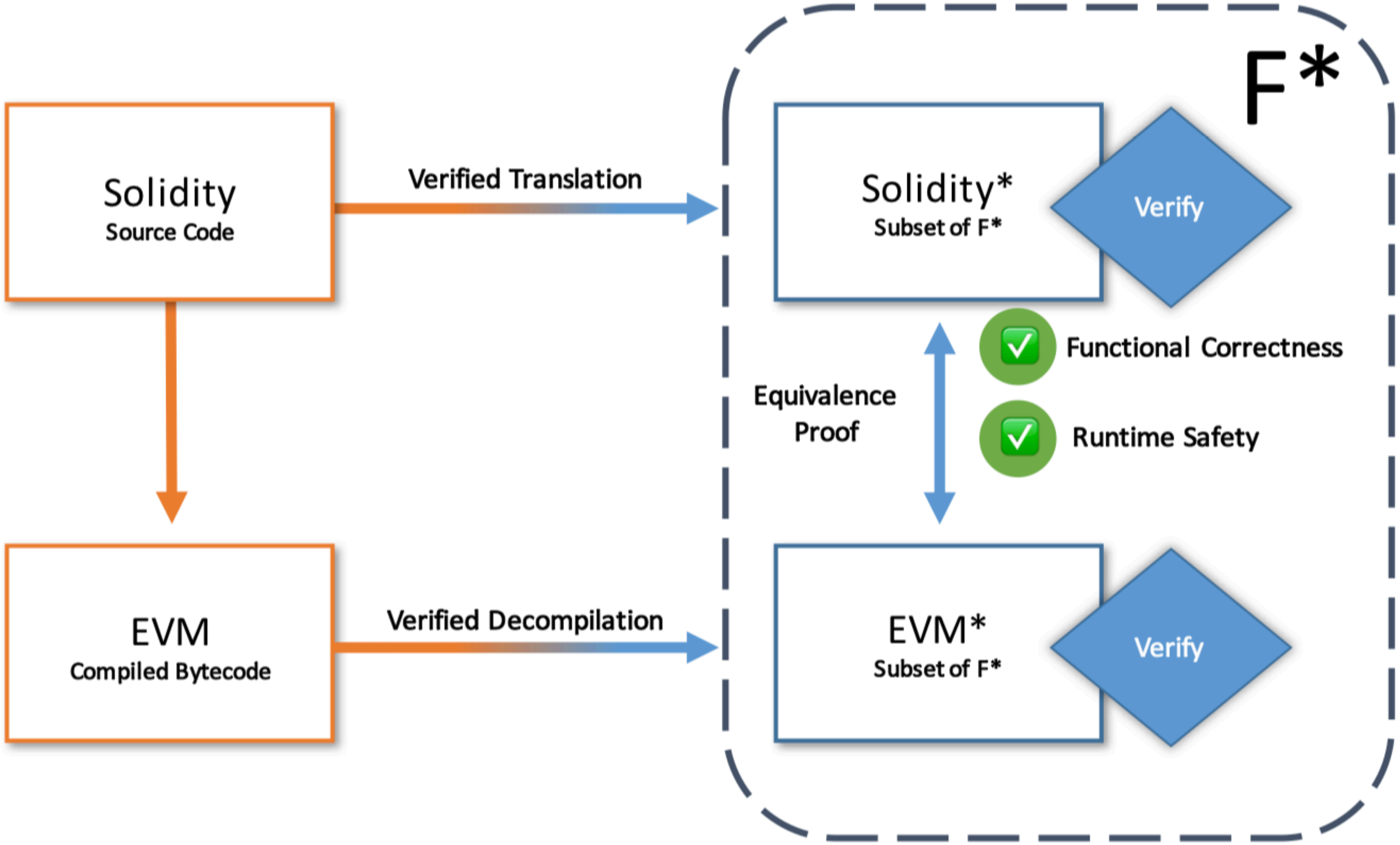
All arithmetic is modulo 2^{256} unless otherwise noted. The zero-th power of zero 0^0 is defined to be one.

Value	Mnemonic	δ	α	Description
0x00	STOP	0	0	Halts execution.
0x01	ADD	2	1	Addition operation. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[0] + \boldsymbol{\mu}_s[1]$
0x02	MUL	2	1	Multiplication operation. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[0] \times \boldsymbol{\mu}_s[1]$
0x03	SUB	2	1	Subtraction operation. $\boldsymbol{\mu}'_s[0] \equiv \boldsymbol{\mu}_s[0] - \boldsymbol{\mu}_s[1]$
0x04	DIV	2	1	Integer division operation. $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 0 & \text{if} \quad \boldsymbol{\mu}_s[1] = 0 \\ \lfloor \boldsymbol{\mu}_s[0] \div \boldsymbol{\mu}_s[1] \rfloor & \text{otherwise} \end{cases}$
0x05	SDIV	2	1	Signed integer division operation (truncated). $\boldsymbol{\mu}'_s[0] \equiv \begin{cases} 0 & \text{if } \boldsymbol{\mu}_s[1] = 0 \\ -2^{255} & \text{if } \boldsymbol{\mu}_s[0] = -2^{255} \wedge \boldsymbol{\mu}_s[1] = -1 \\ \mathbf{sgn}(\boldsymbol{\mu}_s[0] \div \boldsymbol{\mu}_s[1]) \lfloor \lvert \boldsymbol{\mu}_s[0] \div \boldsymbol{\mu}_s[1] \rvert \rfloor & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers. Note the overflow semantic when -2^{255} is negated.

https://ethereum.github.io/yellowpaper/paper.pdf

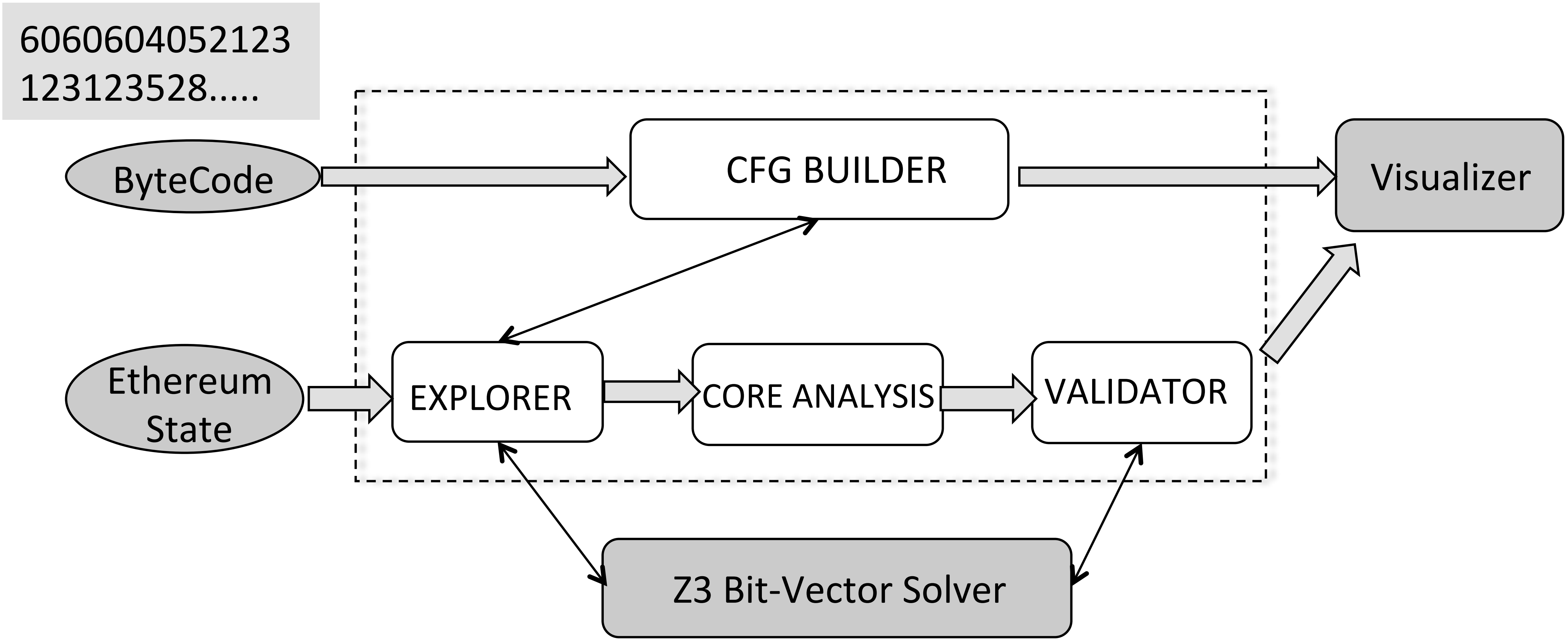
Smart Contract Vulnerabilities: Research

- Formal Verification of Smart Contracts, PLAS 2016
  - A small subset of the Solidity programming language
  - A tiny language and no automatic verification



Smart Contract Vulnerabilities: Research

- Making Smart Contracts Smarter, CCS 2016
  - Oyente: Symbolic execution of EVM bytecode
  - Not sound nor complete



Smart Contract Vulnerabilities: Research

- A Survey of Attacks on Ethereum Smart Contracts, POST 2017

Level	Cause of vulnerability	Attacks
Solidity	Call to the unknown	4.1
	Gasless send	4.2
	Exception disorders	4.2, 4.5
	Type casts	—
	Reentrancy	4.1
	Keeping secrets	4.3
EVM	Immutable bugs	4.4, 4.5
	Ether lost in trasfer	—
	Stack size limit	4.5
Blockchain	Unpredictable state	4.5, 4.6
	Generating randomness	—
	Time constraints	4.5

Table 1. Taxonomy of vulnerabilities in Ethereum smart contracts.

Smart Contract Vulnerabilities: Research

- Zeus: Analyzing Safety of Smart Contracts, NDSS 2018
  - Verification using an LLVM model checker after compiling Solidity code to LLVM bitcode

Smart Contract Vulnerabilities: Research

- A Semantic Framework for the Security Analysis of Ethereum Smart Contracts, POST 2018

Formal guarantees

- **We prove our abstraction to be sound:**
  - **"Every concrete execution can be mimicked by derivations in the abstract semantics"**

artificial id of contract c (in accordance with C_{call})

$$\Gamma \vDash s_c :: S \rightarrow^* S' + + S$$

set of known contracts

$$\Rightarrow \beta_s(s, id, 0) \cup \beta_C(\mathcal{C}_{call}, id) \vdash \beta_S(S', \mathcal{C}_{call}, id, 0)$$

encoding of execution state as predicate instances

encoding of program logics of known contracts as horn clauses

encoding of call stack as predicate instances

Smart Contract Vulnerabilities: Research

- USENIX Security 2018

Track 1

Smart Contracts

Hide details

Session Chair: Suman Jana, *Columbia University*

teEther: Gnawing at Ethereum to Automatically Exploit Smart Contracts
Johannes Krupp and Christian Rossow, *CISPA, Saarland University, Saarland Informatics Campus*

AVAILABLE MEDIA

Show details

Enter the Hydra: Towards Principled Bug Bounties and Exploit-Resistant Smart Contracts
Lorenz Breidenbach, *Cornell Tech, IC3, ETH Zurich;* Philip Daian, *Cornell Tech, IC3;* Florian Tramer, *Stanford;* Ari Juels, *Cornell Tech, IC3, Jacobs Institute*

AVAILABLE MEDIA

Show details

Arbitrum: Scalable, private smart contracts
Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten, *Princeton University*

AVAILABLE MEDIA

Show details

Erays: Reverse Engineering Ethereum's Opaque Smart Contracts
Yi Zhou, Deepak Kumar, Surya Bakshi, Joshua Mason, Andrew Miller, and Michael Bailey, *University of Illinois, Urbana-Champaign*

AVAILABLE MEDIA

Show details

Smart Contract Vulnerabilities: Research

- ACM CCS 2018

Smart Contracts (203 AB)

Session Chair: Yan Chen

SECURIFY: Practical Security Analysis of Smart Contracts
Petar Tsankov (ETH Zurich), Andrei Marian Dan (ETH Zurich), Dana Drachsler Cohen (ETH Zurich), Arthur Gervais (Imperial College London), Florian Buenzli (ETH Zurich), Martin Vechev (ETH Zurich)

BitML: a calculus for Bitcoin smart contracts
Massimo Bartoletti (University of Cagliari), Roberto Zunino (University of Trento)

Vulnerable Semantics of Solidity: Scope

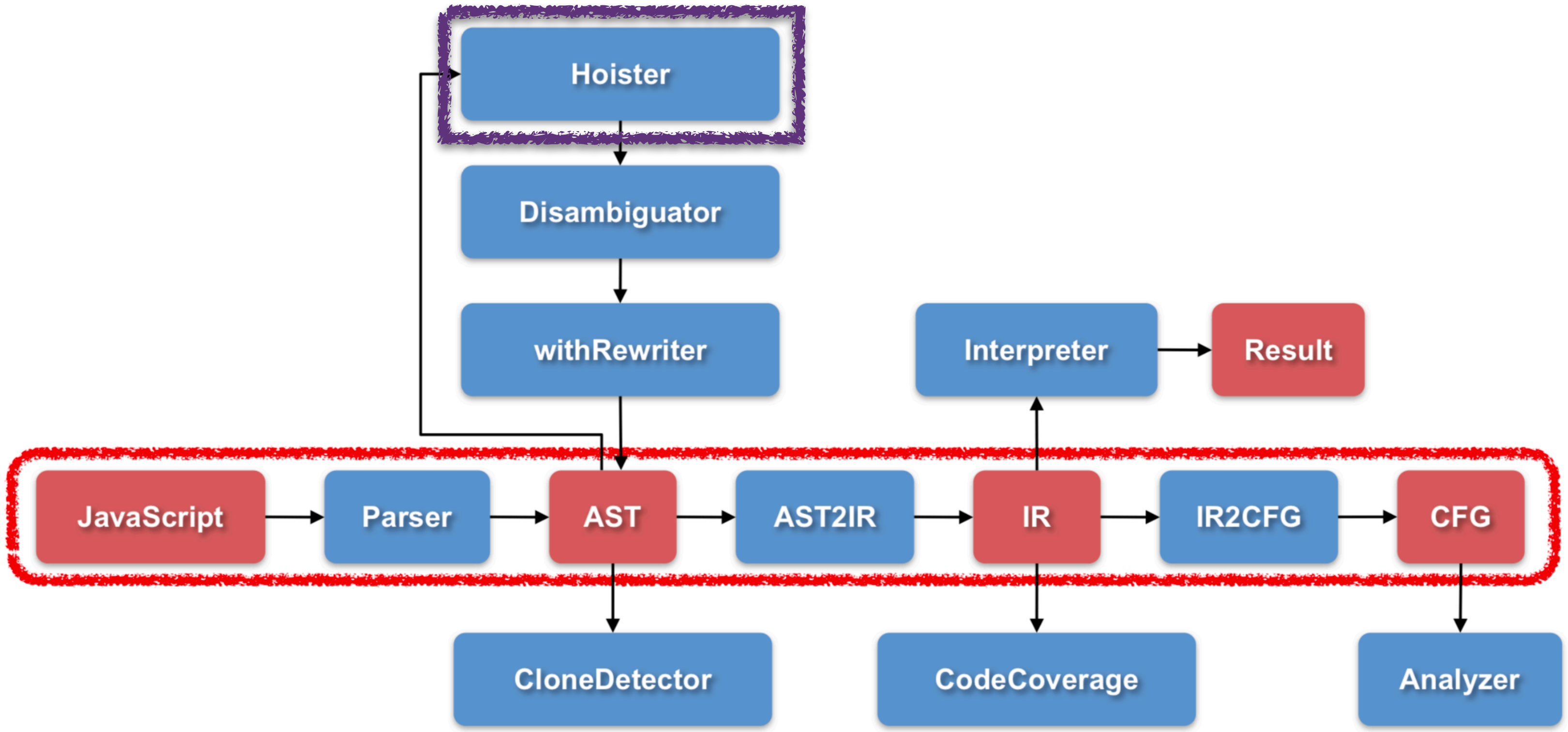
Scoping and Declarations

A variable declared anywhere within a function will be in scope for the *entire function*, regardless of where it is declared (this will change soon, see below). This happens because Solidity inherits its scoping rules from JavaScript. This is in contrast to many languages where variables are only scoped where they are declared until the end of the semantic block. As a result, the following code is illegal and cause the compiler to throw an error, `Identifier already declared`:

Vulnerable Semantics of Solidity: Scope

Scoping and Declarations

A variable declared anywhere within a function will be in scope for the *entire function*, regardless of where it is declared (this will change soon, see below). This happens because Solidity inherits its scoping rules from JavaScript. This is in contrast to many languages where variables are only scoped where they are declared until the end of the semantic block. As a result, the following code is illegal and cause the compiler to throw an error, `Identifier already declared`:



Vulnerable Semantics of Solidity: Scope

Scoping and Declarations

A variable declared anywhere within a function will be in scope for the *entire function*, regardless of where it is declared (this will change soon, see below). This happens because Solidity inherits its scoping rules from JavaScript. This is in contrast to many languages where variables are only scoped where they are declared until the end of the semantic block. As a result, the following code is illegal and cause the compiler to throw an error, `Identifier already declared`:

Scoping starting from Version 0.5.0

Starting from version 0.5.0, Solidity will change to the more widespread scoping rules of C99 (and many other languages): Variables are visible from the point right after their declaration until the end of a `{ }` -block. As an exception to this rule, variables declared in the initialization part of a for-loop are only visible until the end of the for-loop.

Vulnerable Semantics of Solidity: MM

Overload resolution and Argument matching

Overloaded functions are selected by matching the function declarations in the current scope to the arguments supplied in the function call. Functions are selected as overload candidates if all arguments can be implicitly converted to the expected types. If there is not exactly one candidate, resolution fails.

Note

Return parameters are not taken into account for overload resolution.

```
pragma solidity ^0.4.16;

contract A {
    function f(uint8 _in) public pure returns (uint8 out) {
        out = _in;
    }

    function f(uint256 _in) public pure returns (uint256 out) {
        out = _in;
    }
}
```

Vulnerable Semantics of Solidity: MM

Overload resolution and Argument matching

Overloaded functions are selected by matching the function declarations in the current scope to the arguments supplied in the function call. Functions are selected as overload candidates if all arguments can be implicitly converted to the expected types. If there is not exactly one candidate, resolution fails.

Note

Return parameters are not taken into account for overload resolution.

Return Type Rule: $\boxed{\Delta \vdash d \text{ return type wrt } d \text{ ok}}$

$$\frac{name(d) \neq name(d')}{\Delta \vdash d \text{ return type wrt } d' \text{ ok}} \; [\text{Return-Triv}] \qquad \frac{\neg(\Delta \vdash dom(d) \sqsubseteq dom(d'))}{\Delta \vdash d \text{ return type wrt } d' \text{ ok}} \; [\text{Return-Not-Less}]$$

$$\frac{\begin{array}{c} arrow(d) = \forall [\![\overline{\kappa}]\!](\alpha \to \rho) \quad \overline{\kappa = \{\overline{\chi}\} <: P <: \{\overline{\eta}\}} \quad arrow(d') = \forall [\![\overline{\kappa'}]\!](\alpha' \to \rho') \\ \overline{\kappa' = \{\overline{\chi'}\} <: Q <: \{\overline{\eta'}\}} \quad distinct(\overline{P}, \overline{Q}) \quad \Delta \vdash dom(d) \sqsubseteq dom(d') \\ \Delta \vdash \forall [\![\overline{\kappa}]\!](\alpha \to \rho) \sqsubseteq \forall [\![\overline{\kappa}, \overline{\kappa'}]\!]((\alpha \sqcap \alpha') \to \rho') \end{array}}{\Delta \vdash d \text{ return type wrt } d' \text{ ok}} \; [\text{Return-Test}]$$

Vulnerable Semantics of Solidity: MM

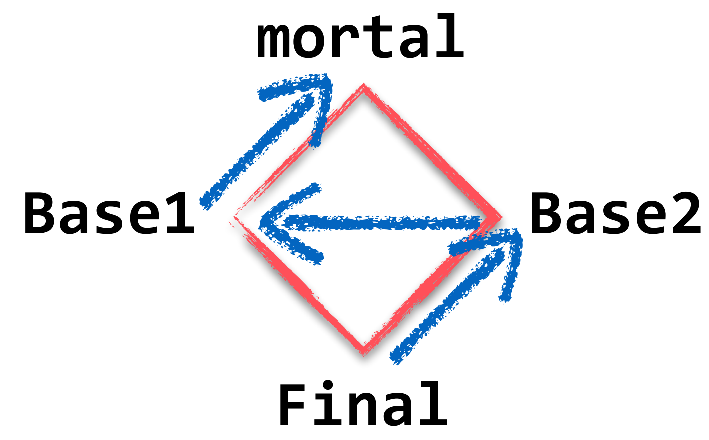
Multiple Inheritance and Linearization

Languages that allow multiple inheritance have to deal with several problems. One is the Diamond Problem. Solidity is similar to Python in that it uses "C3 Linearization" to force a specific order in the DAG of base classes. This results in the desirable property of monotonicity but disallows some inheritance graphs. Especially, the order in which the base classes are given in the `is` directive is important: You have to list the direct base contracts in the order from "most base-like" to "most derived". Note that this order is different from the one used in Python. In the following code, Solidity will give the error "Linearization of inheritance graph impossible".

```
// This will not compile

pragma solidity ^0.4.0;

contract X {}
contract A is X {}
contract C is A, X {}
```

The reason for this is that `C` requests `X` to override `A` (by specifying `A, X` in this order), but `A` itself requests to override `X`, which is a contradiction that cannot be resolved.

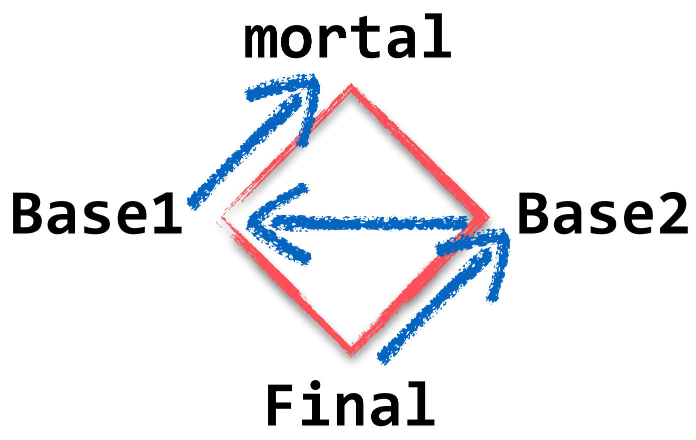
Vulnerable Semantics of Solidity: MM



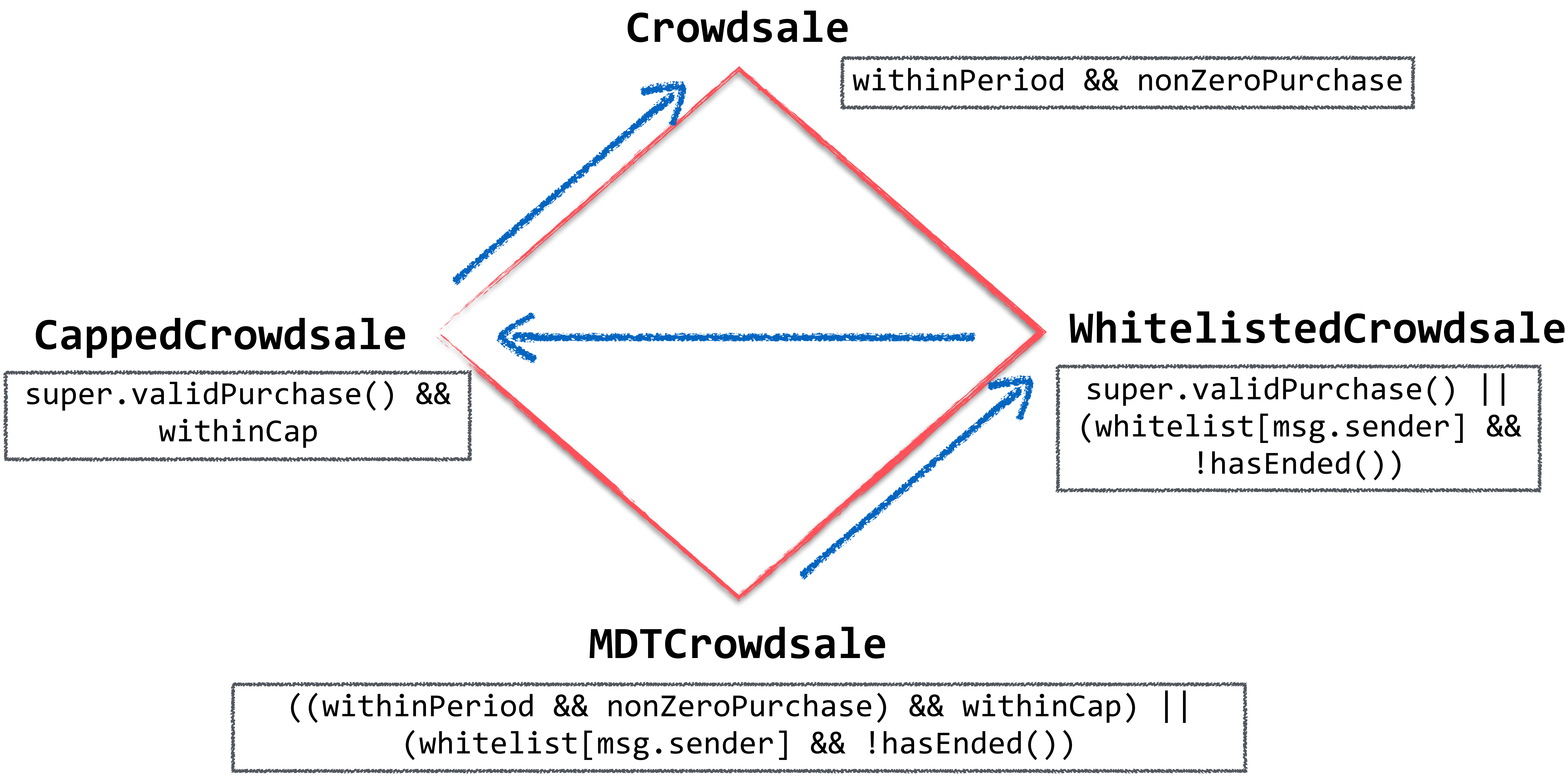

```
pragma solidity ^0.4.22;

contract owned { … }

contract mortal is owned {
    function kill() public {
        if (msg.sender == owner) selfdestruct(owner);
    }
}

contract Base1 is mortal {
    function kill() public { /* cleanup 1 */ mortal.kill(); }
}

contract Base2 is mortal {
    function kill() public { /* cleanup 2 */ mortal.kill(); }
}

contract Final is Base1, Base2 { … }
```

Vulnerable Semantics of Solidity: MM

```
pragma solidity ^0.4.22;

contract owned { … }

contract mortal is owned {
    function kill() public {
        if (msg.sender == owner) selfdestruct(owner);
    }
}

contract Base1 is mortal {
    function kill() public { /* cleanup 1 */ super.kill(); }
}

contract Base2 is mortal {
    function kill() public { /* cleanup 2 */ super.kill(); }
}

contract Final is Base1, Base2 { … }
```



Vulnerable Semantics of Solidity: MM



https://pdaian.com/blog/solidity-anti-patterns-fun-with-inheritance-dag-abuse/

Vulnerable Semantics of Solidity: MM

Multiple Inheritance and Linearization

Languages that allow multiple inheritance have to deal with several problems. One is the Diamond Problem. Solidity is similar to Python in that it uses "C3 Linearization" to force a specific order in the DAG of base classes. This results in the desirable property of monotonicity but disallows some

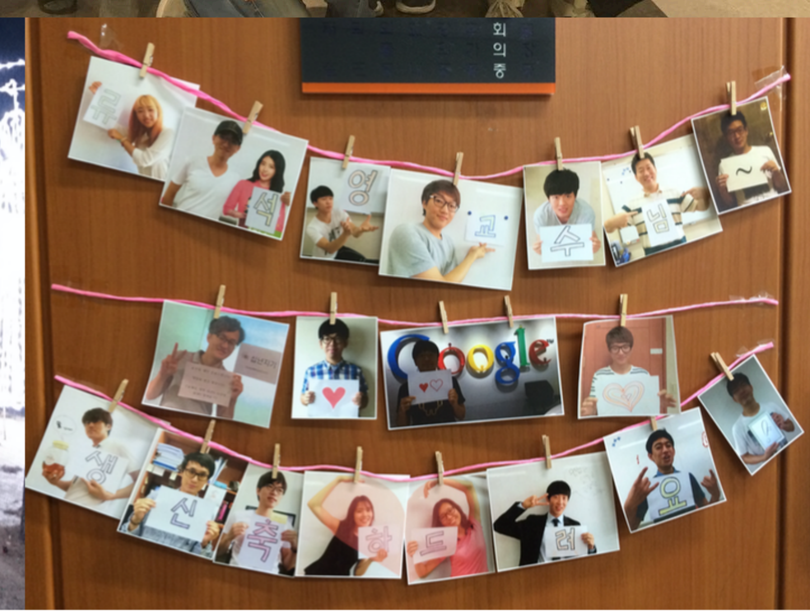
(The name "C3" is not an initialism.) It was first published at the 1996 OOPSLA conference, in a paper entitled "A Monotonic Superclass Linearization for Dylan".[1] It was adapted to the Open Dylan implementation in January 2012[2] following an enhancement proposal.[3] It has been chosen as the default algorithm for method resolution in Python 2.3 (and newer),[4][5] Perl 6,[6] Parrot,[7], Solidity, and PGF/TikZ's Object-Oriented Programming module[8]. It is also available as an alternative, non-default MRO in the core of Perl 5 starting with version 5.10.0.[9] An extension implementation for earlier versions of Perl 5 named `Class::C3` exists on CPAN.[10]

Smart Contract Vulnerabilities: Research

- Which platform?
  - Ethereum, Michelson/Liquidity, Zilliqa/Scilla, ...
- Which language?
  - Source-level: Solidity, LLL, Vyper, ...
  - Bytecode
- What problems?
  - Bugs: type-related, resource-related, ...
  - Vulnerabilities: security, privacy, ...

PLRG@KAIST for the Wild with Theory

- PL to save the world from *bugs* in real-world applications
- PL with *proofs*
- PL for Software Engineering
- PL for Security
- 열심히, 즐겁게, 자발적으로



PLRG@KAIST for the Wild with Theory

Come visit PLRG@KAIST!