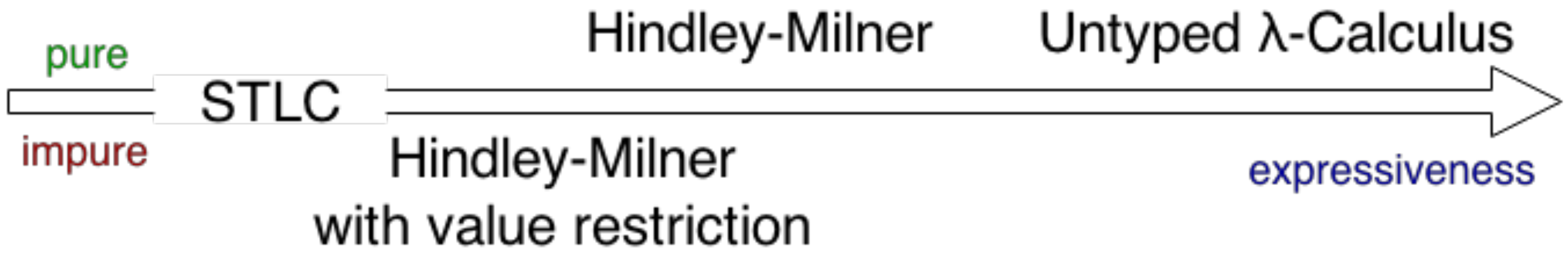
Locally Relax the Value Restriction by Control Flow Analysis

Team 17
b00902064 資訊四 宋昊恩
b00902107 資訊四 游書泓

Abstract

- This project is aiming at relaxing the value restriction locally via flow analysis, where the value striction has long been the standard yet too restrictive solution to integrating Hindler-Milner style polymorphism with imperative features.



Motivation

- The value restriction refuses to generalize all non-value terms, hence rejecting procedures that compute polymorphic functions.
  - Also rejects polymorphic data structures

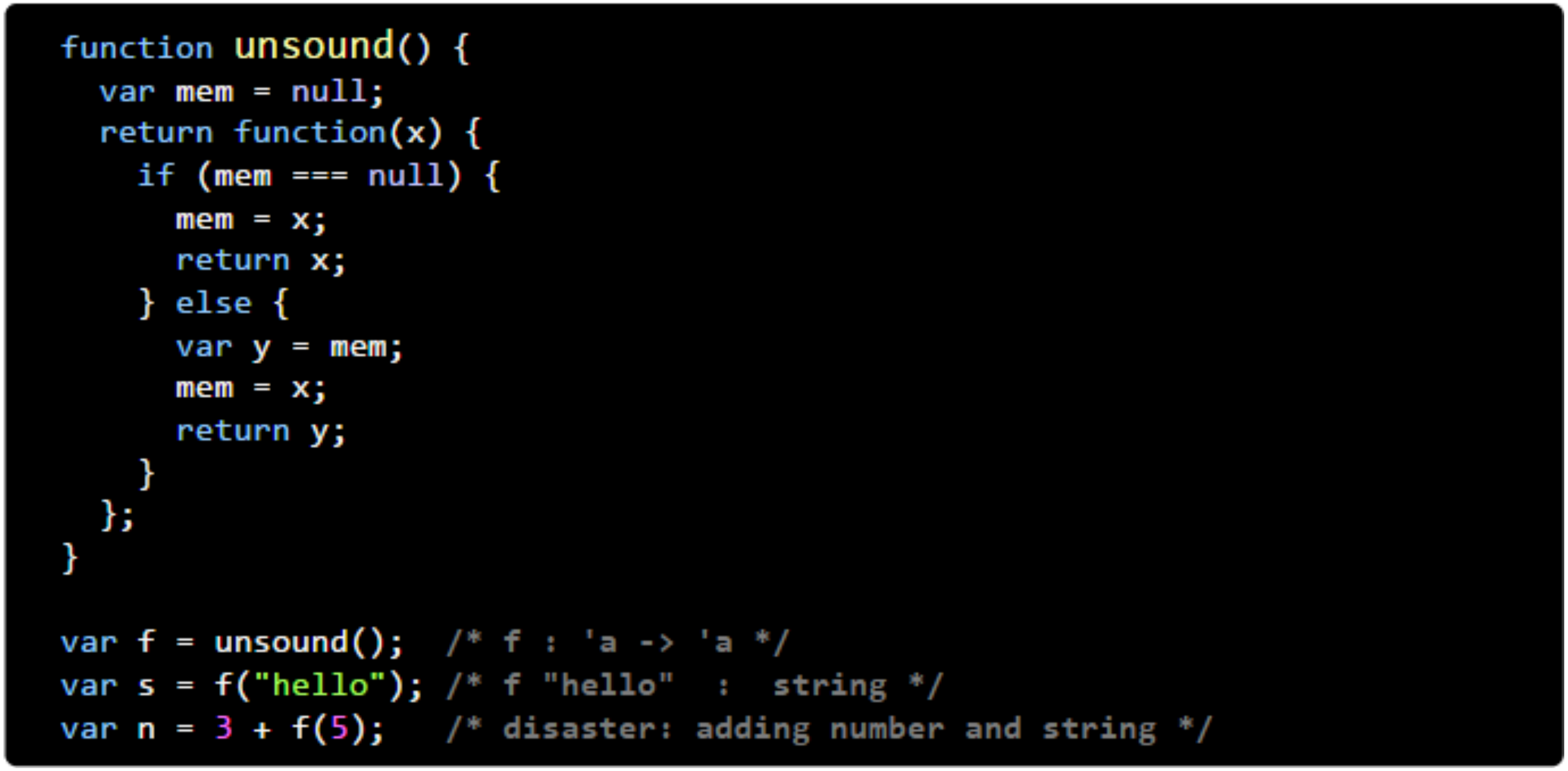
- The use of imperative features is rare; most of the computations are functionally pure.

Challenges

- The value restriction is actually at a balance point that any extension could probably be unwillingly complex and break the module abstraction.

- Polymorphism has a bad interaction with imperative features such as mutable variables.

Challenges (cont.)



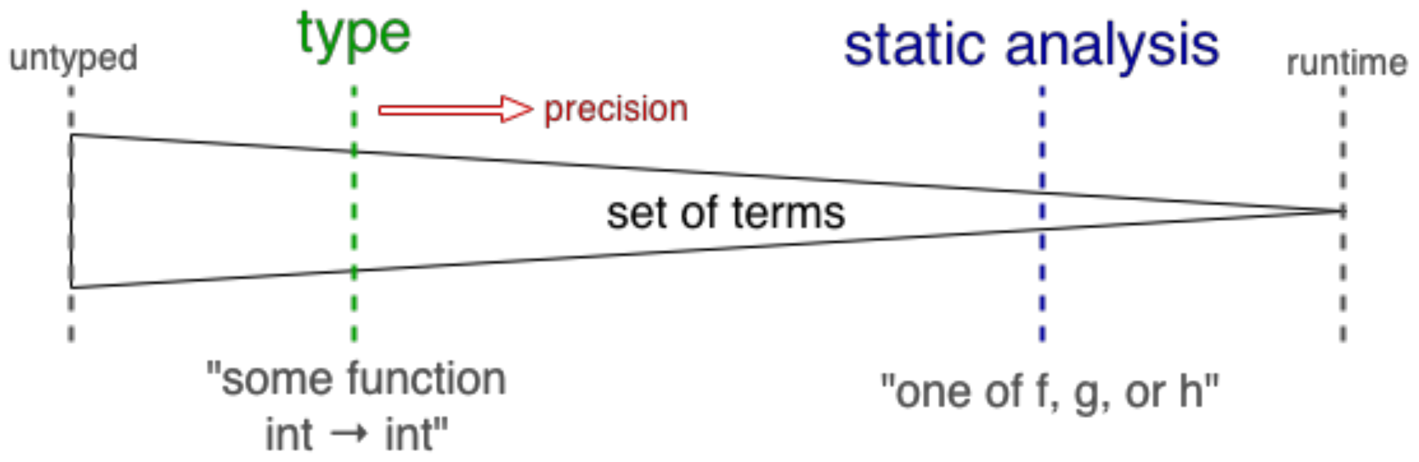
```
function unsound() {
  var mem = null;
  return function(x) {
    if (mem === null) {
      mem = x;
      return x;
    } else {
      var y = mem;
      mem = x;
      return y;
    }
  };
}

var f = unsound();  /* f : 'a -> 'a */
var s = f("hello"); /* f "hello"  :  string */
var n = 3 + f(5);   /* disaster: adding number and string */
```

Potential Solutions

- Apply techniques from static program analysis, tracking type information together with the use of imperative features.
  - The analysis shall basically be *syntax-directed*.
  - Any spotted safe type variables should be generalized.

Novelty

- Static analysis have been used to identify **potential** errors in the program, while type systems can **prove** the absence of certain runtime errors.



- This project is an attempt to bring the two together so that some safe but non-provable programs could be allowed without much modification to the original type system.

Evaluation

- Our concerns include
  - Accept more terms in practice, i.e. improve expressiveness
  - Impact on compilation time

Locally Relax the Value Restriction by Control Flow Analysis

- **Team 17, b00902064 宋昊恩, b00902107 游書泓**
- Reference
  - Simple Imperative Polymorphism
  - Andrew K. Wright
  - *Lisp and Symbolic Computation - Special issue on state in programming languages* (part I) archive
    Volume 8 Issue 4, Dec. 1995
    Pages 343 - 355

Problem statement

- If we want to do substitution $e\downarrow 1$ for x in $e\downarrow 2$:
  - let val x = e↓1 in e↓2 end
    - pretty well, system will automatically do the substitution

- However, if reference were of our concern:
  - let val x = (ref 1) in x := 2; !x end
    - we wish to create a variable with value 1, set it to 2 then output 2
    - but the system will automatically do substitution and output 1

- The properties of the type system were broken

Problem statement (cont.)

- Is the problem clearly defined?
  - The author provides 3 counterexamples to the original type system were there references in the program
- Is it a good research problem?
  - Yes, as the interaction between side-effects and higher-order features is non-trivial
- Is the problem relevant to your research or interests?
  - Yes, it provides an innovative direction for our research.

Proposed solution

- Limit polymorphism to values
  - the former sample will not type check
  - the type system will be sound
- However, it causes many false alarm
  - compiler alerts non-existent error
  - treat type variables too strictly than it should be
- Three new scenarios that need to be fixed

Proposed solution (cont.)

- Are they stated clearly? Can you describe them in a short paragraph?
  - This solution is simple and understandable.
  - It can be briefly described.
- Are the solutions sound and reasonable?
  - Yes; though there are some exceptions, the author provides explicitly solutions and claims that they rarely occur.
- Can you use/modify their solutions to solve your problem?
  - The provided solution can be strengthen.

Evaluation

- The paper tests dozens of ML programs with the modified compiler
  - code sizes range from 100 to 83100 lines
  - only half of them need to be fixed
  - the most complicated modification involves
    - 6 η-expansions
    - one declaration moved
    - one conditional statement changed

- In practice, exceptions rarely occur

Evaluation (cont.)

- Are their solutions carefully evaluated?.
  - Evaluation method and statistics are compact and convincing.

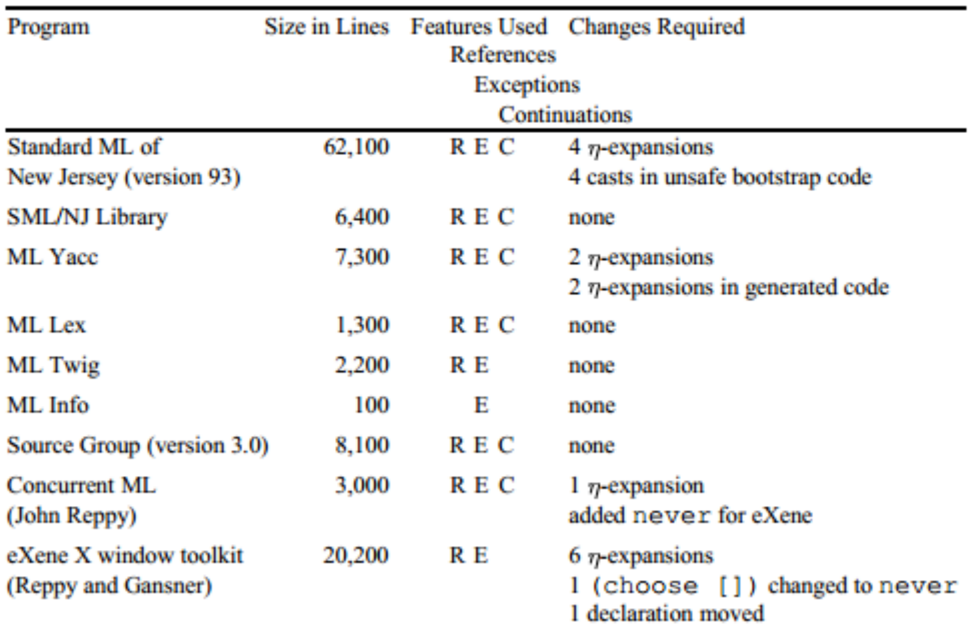


Table 1. Practical impact of limiting polymorphism to values

Program	Size in Lines	Features Used	Changes Required
		References	
		Exceptions	
		Continuations	
Standard ML of New Jersey (version 93)	62,100	R E C	4 η-expansions 4 casts in unsafe bootstrap code
SML/NJ Library	6,400	R E C	none
ML Yacc	7,300	R E C	2 η-expansions 2 η-expansions in generated code
ML Lex	1,300	R E C	none
ML Twig	2,200	R E	none
ML Info	100	E	none
Source Group (version 3.0)	8,100	R E C	none
Concurrent ML (John Reppy)	3,000	R E C	1 η-expansion added `never` for eXene
eXene X window toolkit (Reppy and Gansner)	20,200	R E	6 η-expansions 1 `(choose [])` changed to `never` 1 declaration moved

Applications and Future work

- Type system plays a key role in functional programming language.

- Concept of functional programming has become prevailing in C++, Java, Ruby, Python, etc.

- A more compatible compiler will shorten the gaps between static and runtime behavior.

Applications and Future work (cont.)

- Why people care about such applications? Where does its value come from?
  - The usage of type system becomes ubiquitous.

- How big is the gap there between the existing theory and its final application? Is there any research value hiding behind such gap?
  - The theory derived from the author can be directly implemented in practice.
  - The solution can be strengthened more advancedly.

Locally Relax the Value Restriction by Control Flow Analysis

- **Team 17, b00902064 宋昊恩, b00902107 游書泓**
- Reference
  - A Practical and Flexible Flow Analysis for Higher-Order Languages
  - J. Michael Ashley and R. Kent Dybvig.
  - *ACM Transactions on Programming Languages and Systems 20*, 4, 845-868, July 1988.

The Problem 1/2

- Control Flow Analysis
  - Find the function call at a given program point (together with the relevant bindings)
  - Inseparable from data flow analysis, in contrast to non-higher-order languages

The Problem 2/2

- Control flow analysis
  - It's **impossible** to determine the exact control flow
  - The approximation ought to be safe
  - Precision versus long run time
- Type system and control flow analysis are both one sort of **abstract interpretation**

Methodology 1/2

- A parameterized collecting machine, given operationally.
  - For $M \in CS$, $eval(M) = C$ if
  - $\langle C, \emptyset, \emptyset, C \downarrow 0 \ , \emptyset \rangle \to \uparrow + \langle halt, E, S, C, \emptyset \rangle$
- A variable allocation function, new, controlling precision
- A projection function, Θ, controlling the speed
- Instantiating with different functions gives different analysis, e.g. 0-CFA, 1-CFA

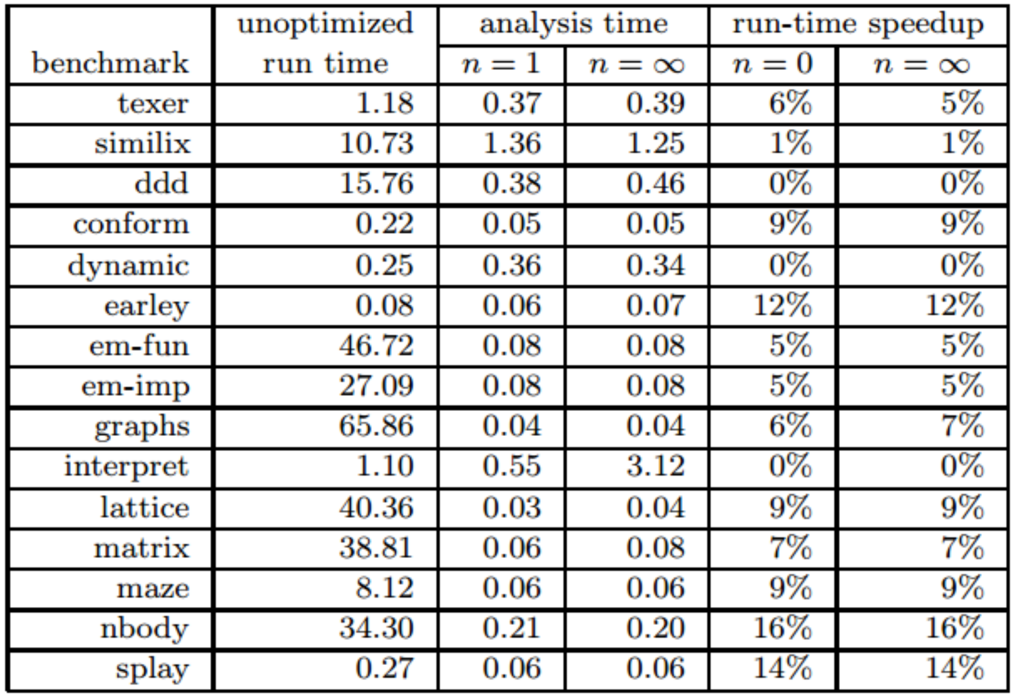
Methodology 2/2

- The collecting machine is sound, and is a unified framework.
- We may probably infer the *pureness* of the expression from the control flow information.

$$
\begin{aligned}
\langle(\mathtt{let}\ ((\mathtt{v}\ N))\ A), \hat{E}, \hat{S}, \hat{C}, \hat{P}\rangle &\rightarrow \langle A, \hat{E}[\mathtt{v} := l], \hat{S}[l := \hat{\gamma}(N, \hat{E}, \hat{S})], \hat{C}, \hat{P}\rangle \\
&\quad \text{where } \langle l\rangle = \widehat{new}(\langle \mathtt{v}\rangle, \hat{E}, \hat{S}) \\
\langle(\mathtt{letrec}\ ((\mathtt{v}\ N))\ A), \hat{E}, \hat{S}, \hat{C}, \hat{P}\rangle &\rightarrow \langle A, \hat{E}', \hat{S}[l := \hat{\gamma}(N, \hat{E}', \hat{S})], \hat{C}, \hat{P}\rangle \\
&\quad \text{where } \langle l\rangle = \widehat{new}(\langle \mathtt{v}\rangle, \hat{E}, \hat{S}) \\
&\quad \qquad\quad \hat{E}' = \hat{E}[\mathtt{v} := l] \\
\langle(\mathtt{set!}\ \mathtt{v}\ N\ \ A), \hat{E}, \hat{S}, \hat{C}, \hat{P}\rangle &\rightarrow \langle A, \hat{E}, \hat{S}[\hat{E}(\mathtt{v}) := \hat{\gamma}(N, \hat{E}, \hat{S})], \hat{C}, \hat{P}\rangle \\
\langle \mathtt{halt}, \hat{E}, \hat{S}, \hat{C}, \hat{P}\rangle &\rightarrow \langle A, \hat{E}', \hat{S}', \hat{C}, \hat{P} - \{\langle A, \hat{E}', \hat{S}'\rangle\}\rangle \\
&\quad \text{where } \langle A, \hat{E}', \hat{S}'\rangle \in \hat{P} \\
\langle(\mathtt{pair?}\ N\ \ A_1\ \ A_2), \hat{E}, \hat{S}, \hat{C}, \hat{P}\rangle &\rightarrow \langle \mathtt{halt}, \hat{E}, \hat{S}, \hat{C}, \hat{P}'\rangle \\
&\quad \text{where } \hat{P}' = (\mathit{pair?}(\hat{\gamma}(N, \hat{E}, \hat{S})) \rightarrow \{\langle A_1, \hat{E}, \hat{S}\rangle\}, \emptyset) \cup \\
&\quad \qquad\qquad\ (\mathit{nonpair?}(\hat{\gamma}(N, \hat{E}, \hat{S})) \rightarrow \{\langle A_2, \hat{E}, \hat{S}\rangle\}, \emptyset) \\
\langle(N_0\ N_1\ \ldots\ N_n), \hat{E}, \hat{S}, \hat{C}, \hat{P}\rangle &\rightarrow \langle \mathtt{halt}, \hat{E}, \hat{S}, \hat{C}', \hat{P}'\rangle \\
&\quad \text{where } \{x_0, \ldots, x_m\} = \hat{\gamma}(N_0, \hat{E}, \hat{S}) \\
&\quad \qquad\qquad\qquad f = \mathit{apply}(\hat{E}, \hat{S}, N_1 \ldots N_n) \\
&\quad \qquad\qquad \langle \hat{C}', \hat{P}'\rangle = (f(x_0) \circ \ldots \circ f(x_m))\langle \hat{C}, \hat{P}\rangle
\end{aligned}
$$

Evaluation 1/2

- Use the control flow information to enable more optimizations



benchmark	unoptimized run time	analysis time		run-time speedup	
		$n = 1$	$n = \infty$	$n = 0$	$n = \infty$
texer	1.18	0.37	0.39	6%	5%
similix	10.73	1.36	1.25	1%	1%
ddd	15.76	0.38	0.46	0%	0%
conform	0.22	0.05	0.05	9%	9%
dynamic	0.25	0.36	0.34	0%	0%
earley	0.08	0.06	0.07	12%	12%
em-fun	46.72	0.08	0.08	5%	5%
em-imp	27.09	0.08	0.08	5%	5%
graphs	65.86	0.04	0.04	6%	7%
interpret	1.10	0.55	3.12	0%	0%
lattice	40.36	0.03	0.04	9%	9%
matrix	38.81	0.06	0.08	7%	7%
maze	8.12	0.06	0.06	9%	9%
nbody	34.30	0.21	0.20	16%	16%
splay	0.27	0.06	0.06	14%	14%

Evaluation 2/2

- The benchmarks indeed included
  - Impact on compile-time and run-time speedup
  - Optimized procedures calls -- which is enabled by the proposed CFA
  - Eliminated closure construction, also requires control flow informations

Applications and Future Work 1/2

- Being parameterized, the proposed framework is applicable to a wide range of analysis. The compiler may freely choose between precision and speed.
- Control flow information is essential to a plenty number of optimizations
  - Also possible to analyse OO languages
  - Higher-order languages are common nowaday: Scheme, Haskell, Standard ML, javascript, Python, Ruby, ...

Applications and Future Work 2/2

- Can also *statically* check the safety of **scripting languages**
- Optimizations for both scripting language and compiled languages are desired
- Between theory and application: Compile-time consumption and precision; many analysis requires tuning