Toward General Diagnosis of Static Errors

Danfeng Zhang and Andrew C. Myers

Cornell University

POPL 2014

Static Program Analysis

- Many flavors
  - Type system
  - Dataflow analysis
  - Information-flow analysis
- Useful properties
  - Type safety
  - Memory safety
  - Information-flow security
- But, (sometimes) confusing error messages make static analyses hard to use

Example 1: ML Type Inference

- OCaml

```
let foo(lst: int list): (float*float) list =
  …
  let rec loop lst x y dir acc =
    if lst = [] then
      acc
    else
      print_string "foo"
  in
    List.rev (loop lst 0.0 0.0 0.0 [(0.0,0.0)])
```

Mistake

OCaml: This expression has type 'a list but is here used with type unit

Locating the error cause is

- Time-consuming
- Difficult

Example 2: Information-Flow Analysis

- Jif: Java + Information-Flow control

Mistake

```
public final byte[{}] {this} encText;
…
public void m(FileOutputStream[{this}]{this} encFos)
  throws (IOException) {
  try {
    for (int i=0; i<encText.length; i++)
      encFos.write(encText[i]);
  } catch (IoException e) {}
}
```

Jif: This label is too restrictive

Better error report is needed

Toward Better Error Reports

- Limitations of previous work
  - Methods reporting full explanation – Verbose reports
  - Analysis-specific methods – Tailored heuristics
  - Methods diagnosing false alarms – No diagnosis of true errors
- Our approach
  - Applies to a large class of program analyses
  - Diagnoses the cause of both true errors and false alarms
  - Reports error causes more accurately than existing tools

Approach Overview

Language-Specific

Programs

Jif

Others

OCaml

```
let foo(lst: int list):(float*float) list =
 let rec loop lst x y dir acc =
   if lst = [] then
     acc
   else
     print_string "foo"
 in
  List.rev(loop lst 0.0 0.0 0.0 [(0.0,0.0)])
```

Constraints

$\texttt{unit} = acc_5$

$acc_5 = acc_3$

$acc_3 = \texttt{(float*float) list}$

$\texttt{unit} = loopret$

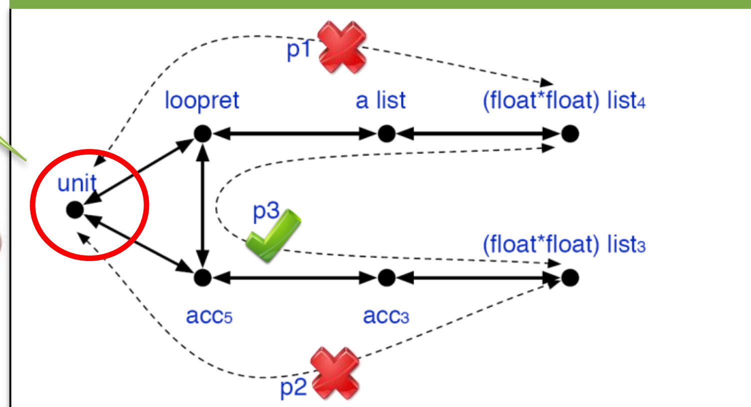
$loopret = \alpha\ \texttt{list}$

$\alpha\ \texttt{list} = \texttt{(float*float) list}$

$loopret = acc_5$

Language-Agnostic

Constraints Analysis via Graph



Cause

Based on Bayesian interpretation

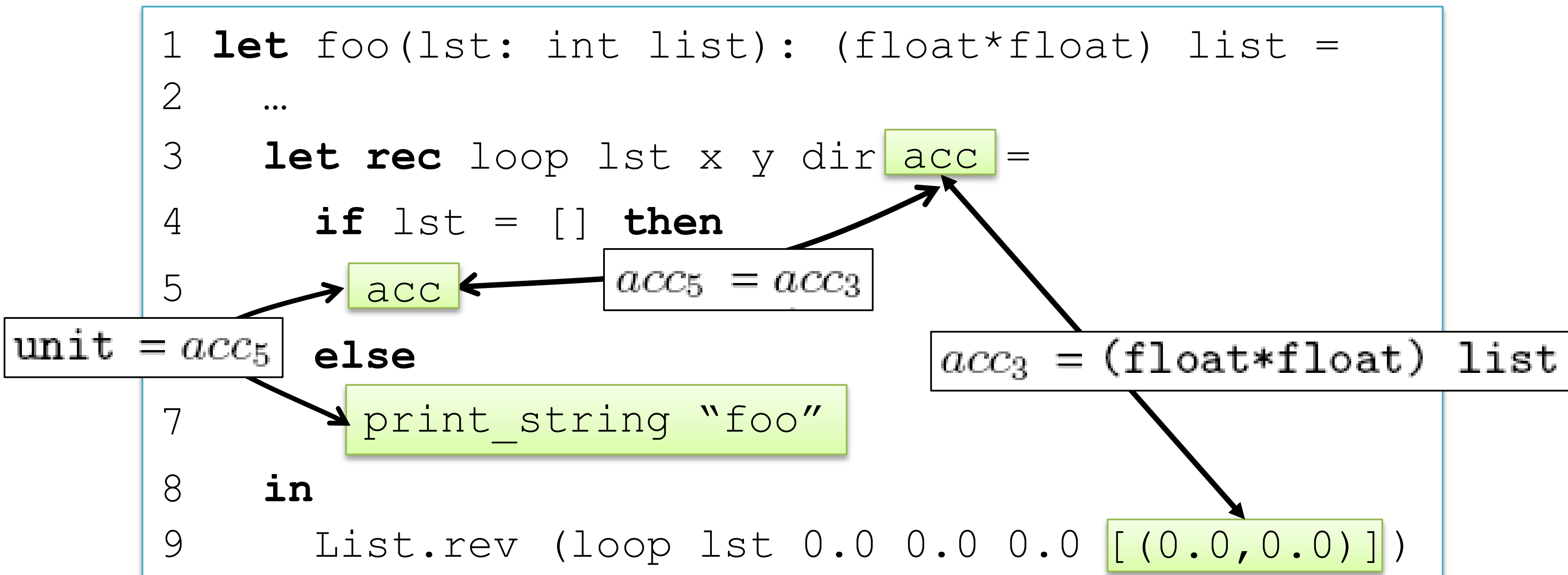
General Diagnosis Heuristics

The error cause is likely to be

- Simple
- Able to explain all errors
- Not used often on correct paths
- (false alarm) weak and simple

From Programs to Constraints

- ML type inference
  - Constraint elements: types
    - Constructors: unit, float, list, $*$
    - Variables: acc_3, acc_5
  - Constraints: type equalities



A General Constraint Language

Syntax of Constraints
$E ::= \alpha \mid c(E_1, \ldots, E_n) \mid \bar{c}^i(E) \mid E_1 \sqcup E_2 \mid E_1 \sqcap E_2 \mid \bot \mid \top$
$I ::= E_1 \leq E_2 \qquad C ::= \bigwedge_i I_{1i} \vdash \bigwedge_j I_{2j}$

- Element (E): form a lattice, with an ordering $\leq$
- Inequality (I): a partial order on elements
  - E.g., "subtype of", "subset of", "less confidential than"
- Constraint (Hypothesis $\vdash$Conclusion)
  - Hypothesis captures programmer assumptions
  - Variable-free constraint is valid when all $\leq$ in conclusion can be derived from hypothesis

Properties of the Constraint Language

- Expressive
  - ML type inference with polymorphism
  - Information-flow analysis with complex security model
  - Dataflow analysis

  (See formal translations in paper)

- Practical to calculate satisfiable/unsatisfiable subsets of constraints

Approach Overview

Programs

- Jif
- Others
- OCaml

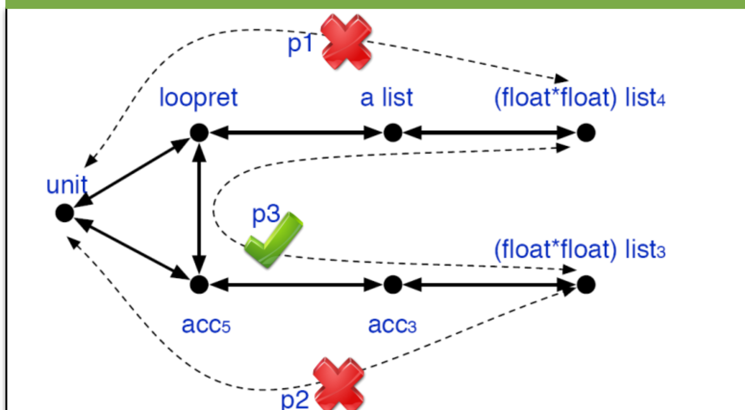
```
let foo(lst: int list):(float*float) list =
 let rec loop lst x y dir acc =
   if lst = [] then
     acc
   else
     print_string "foo"
 in
  List.rev(loop lst 0.0 0.0 0.0 [(0.0,0.0)])
```

Language-Agnostic

Constraints

$\texttt{unit} = acc_5$
$acc_5 = acc_3$
$acc_3 = \texttt{(float*float) list}$
$\texttt{unit} = loopret$
$loopret = \alpha\ \texttt{list}$
$\alpha\ \texttt{list} = \texttt{(float*float) list}$
$loopret = acc_5$

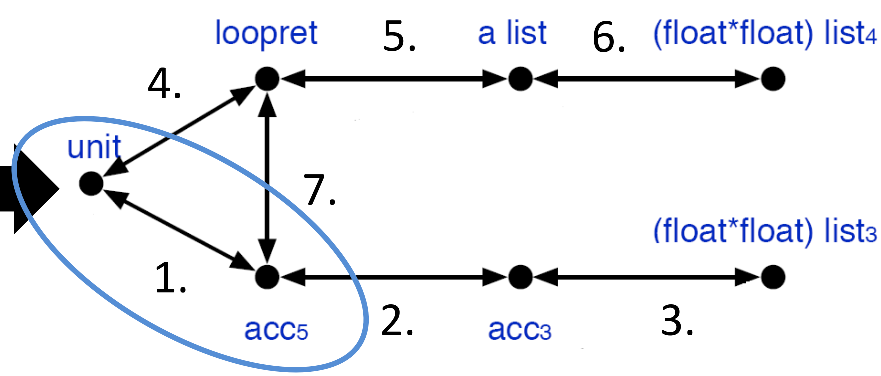
Constraints Analysis via Graph



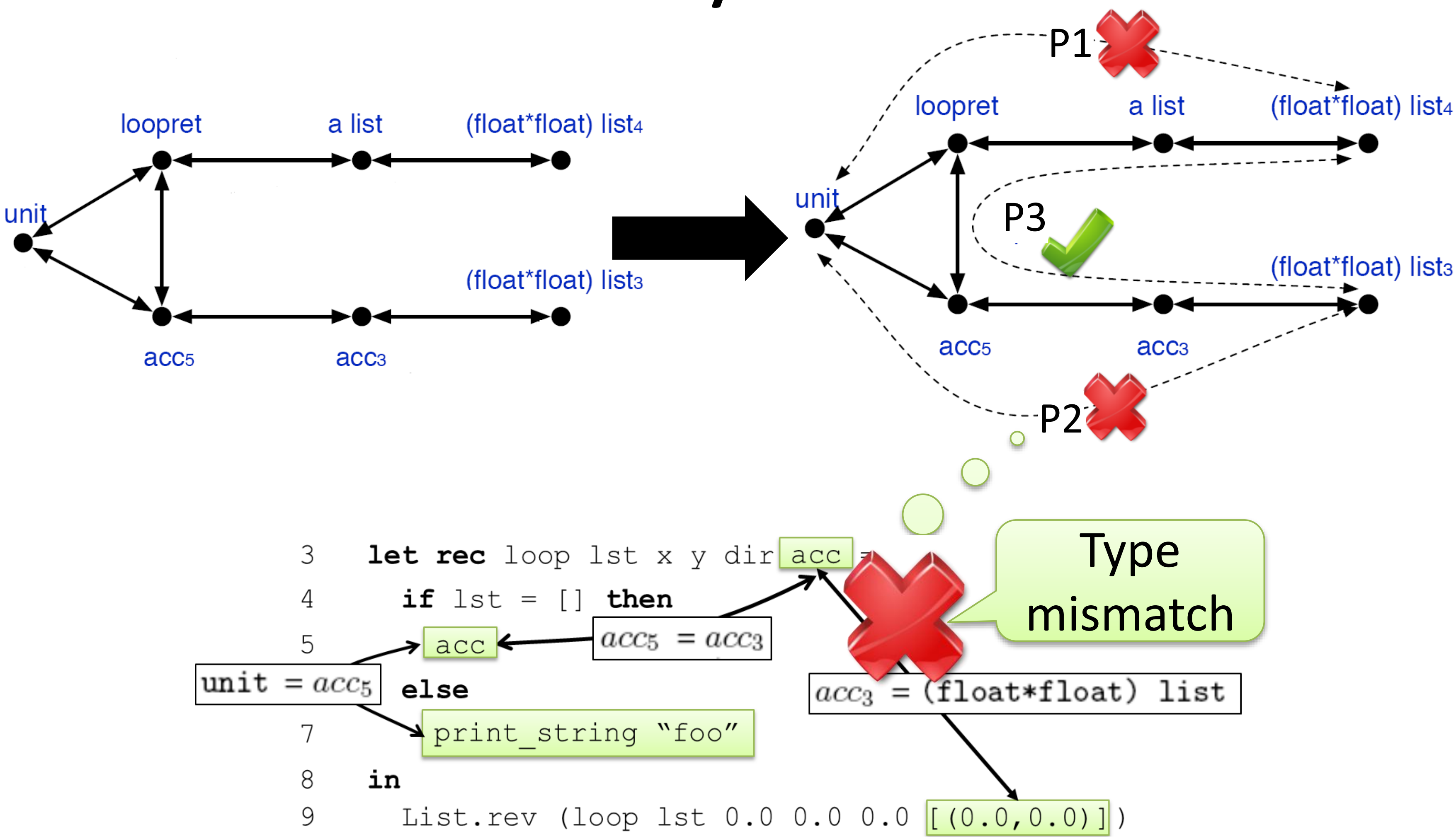
Constraint Graph in a Nutshell

- Graph construction (simple case)
  - Node: constraint element
  - Directed edge: partial ordering

1. unit $= acc_5$
2. $acc_5 = acc_3$
3. $acc_3 =$ (float*float) list
4. unit $= loopret$
5. $loopret = \alpha$ list
6. α list = (float*float) list
7. $loopret = acc_5$



Constraint Analysis in a Nutshell



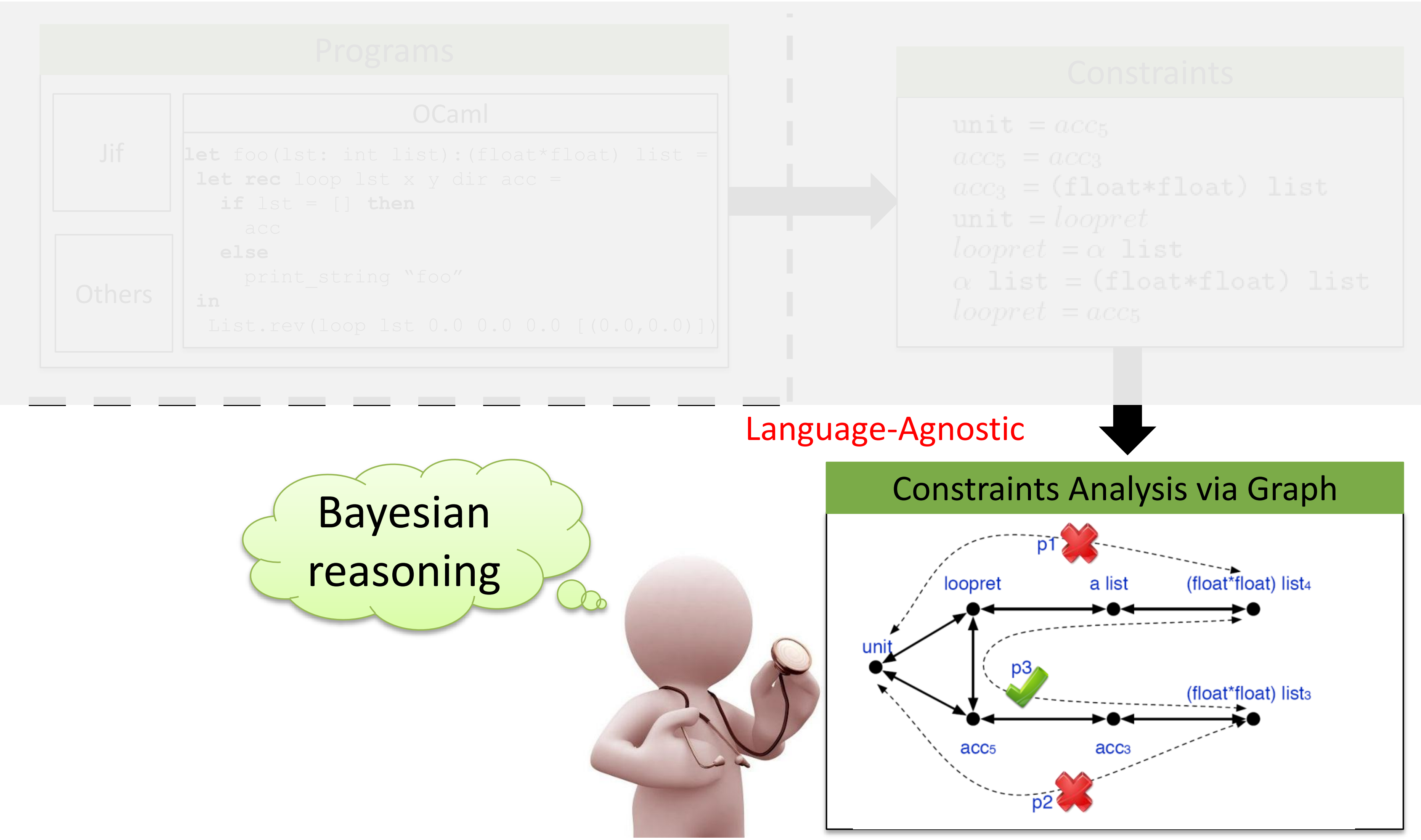
Constraint Analysis for the Full Constraint Language

- Handling constructors, hypotheses
  - CFG Reachability [Barrett et al. 2000, Melski&Reps 2000]
  - Also handles join/meet operations

  (See details in paper)

- Performance
  - Scalable: quadratic w.r.t. # graph nodes in practice

Error Diagnosis



Possible Explanations

- When an analysis reports an error, either
  - The program being analyzed is wrong (true alarm)
    - E.g., an expression is wrong in OCaml program
  - The program analysis reports an false alarm (false alarm)
    - E.g., an assumption is missing in Jif program

- Explanations to find
  - Wrong expressions
  - Missing hypotheses

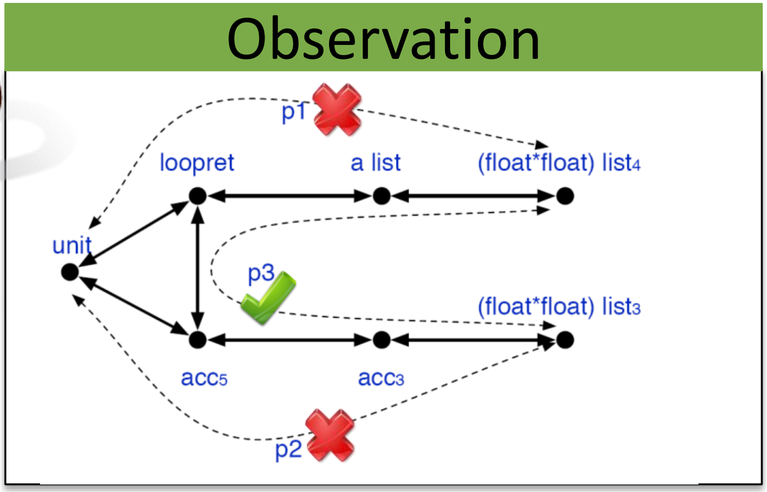
Key insight:
Bayesian reasoning

Inferring Most-Likely Error Cause

- The most likely explanation

$$\underset{(E,H)\in\mathcal{G}}{\operatorname{argmax}} P(E,H|o)$$

  - $\mathcal{G}$: explanation (pair of constraint elements and hypotheses)
  - o : observation (structure of a constraint graph)



Likelihood Estimation

$$\underset{(E,H)\in\mathcal{G}}{\operatorname{argmax}} P_{\Omega}(E)P(o|E,H)P_{\Psi}(H)$$

MAP estimation

Likelihood Estimation

sat paths use elements in E

$$\underset{(E,H)\in\mathcal{G}}{\operatorname{argmax}}\ P_1^{|E|}\left(\frac{P_2}{1-P_2}\right)^{k_E} P_\Psi(H)$$

- Simplifying assumptions:
  - All expressions are equally likely to be wrong (with P_1)
  - Errors are unlikely (with $P_2 < 0.5$) to appear on satisfiable paths
- Intuitively,

General Diagnosis Heuristics

The error cause is likely to be
- Simple
- Able to explain all errors
- Not used often on correct paths
- (missing hypotheses) weak and simple

Explain later

Inferring Likely Wrong Expressions

$$\underset{E}{\text{argmax}}\, P_1^{|E|} \left(\frac{P_2}{1 - P_2}\right)^{k_E}$$

- Search space
  - all **subsets** of expressions (nodes in constraint graph)
- A* search
  - Optimal: all most likely wrong expressions are returned
  - Efficient: 10 seconds when the search space is over 2^{1000}

Evaluation suggests the accuracy is not sensitive to the value of P_1 and P_2

Inferring Likely Missing Hypotheses

$$\underset{H}{\operatorname{argmax}}\, P_{\Psi}(H)$$

- Simplicity is not the only metric
  - $\top \leq \bot$ "explains" all errors
- Likely missing hypotheses are both ***weak*** and ***simple***
  - Minimal weakest hypothesis

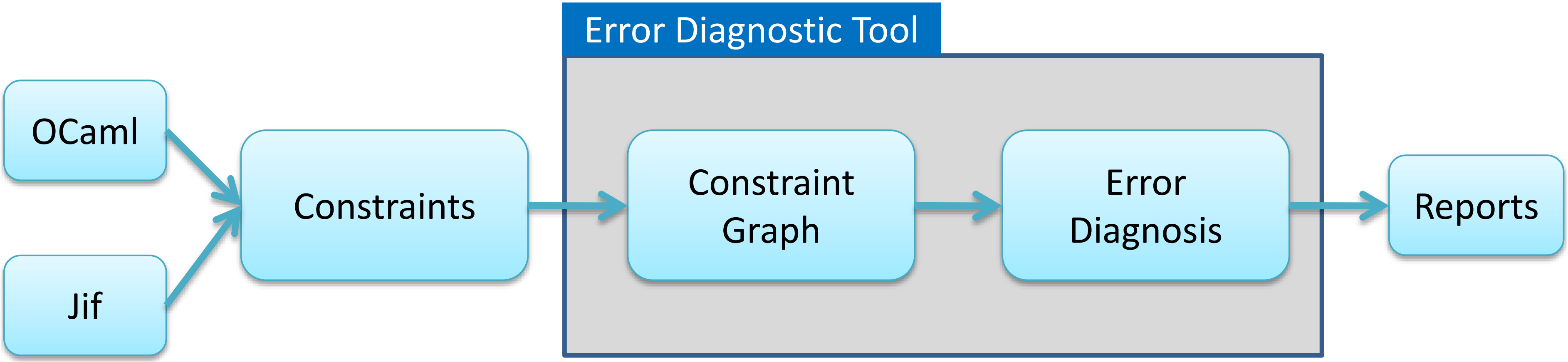
$$\text{Bob} \leq \text{Carol} \vdash \text{Alice} \leq \text{Bob}$$
$$\text{Bob} \leq \text{Carol} \vdash \text{Alice} \leq \text{Carol}$$
$$\text{Bob} \leq \text{Carol} \vdash \text{Alice} \leq \text{Carol} \sqcup \bot$$

Minimal weakest hypothesis
$\text{Alice} \leq \text{Bob}$

Formal definition & search algorithm in paper

Evaluation

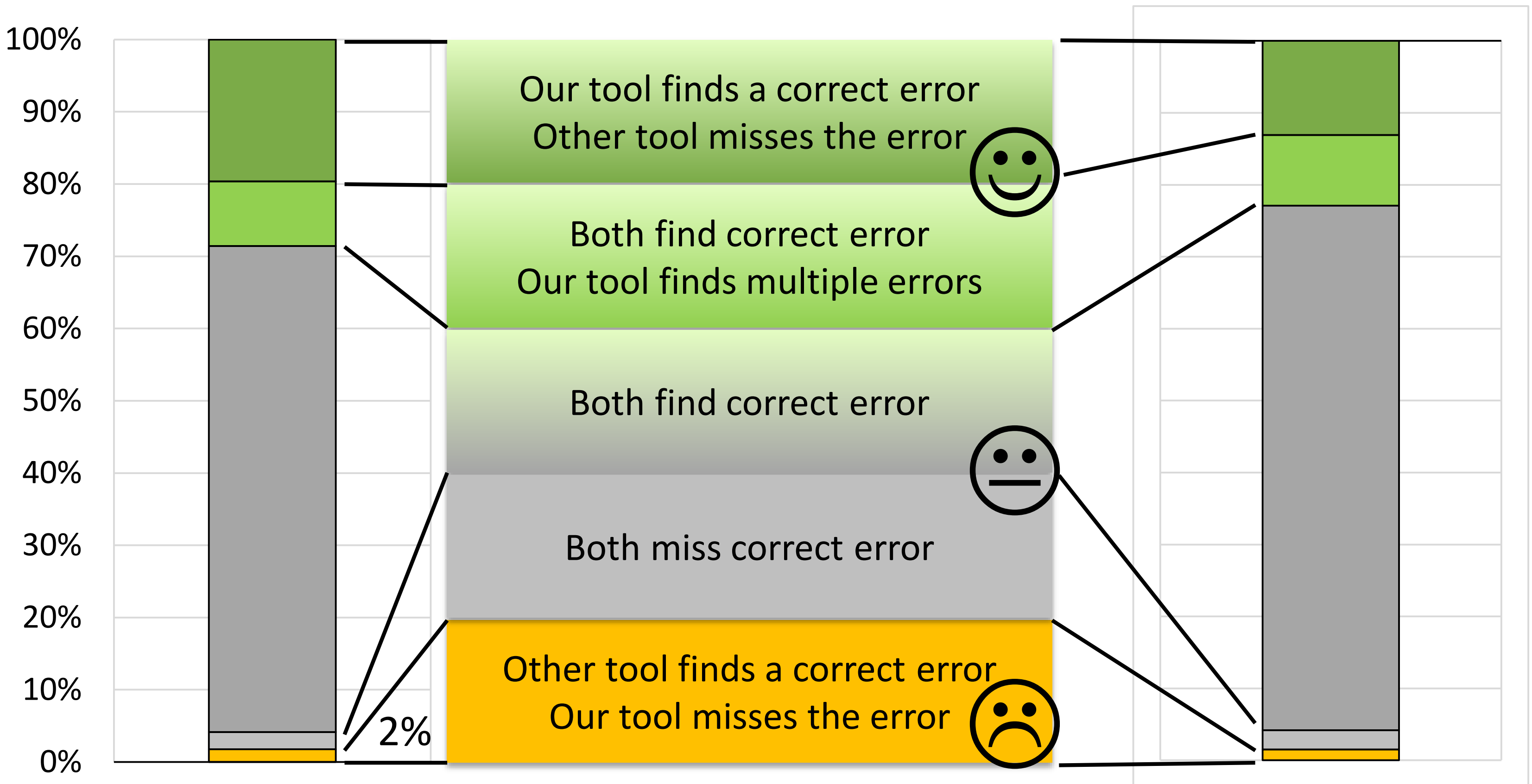
- Implementation
  - Translation from analyses to constraints
    - OCaml: modified EasyOCaml (500 on top of 9,000LoC)
    - Jif: modified Jif (300 on top of 45,000LoC)
  - General error diagnostic tool
    - ~5,500 LoC in Java



Accuracy of Error Reports: OCaml

- Data
  - A corpus of previously collected programs [Lerner et al.'07]
  - Analyzed 336 programs with type mismatch errors
- Metric of report quality
  - Location of programmer mistake: user's fix with larger timestamp
  - Correctness: only when the programmer mistake is returned

Comparison with OCaml and Seminal



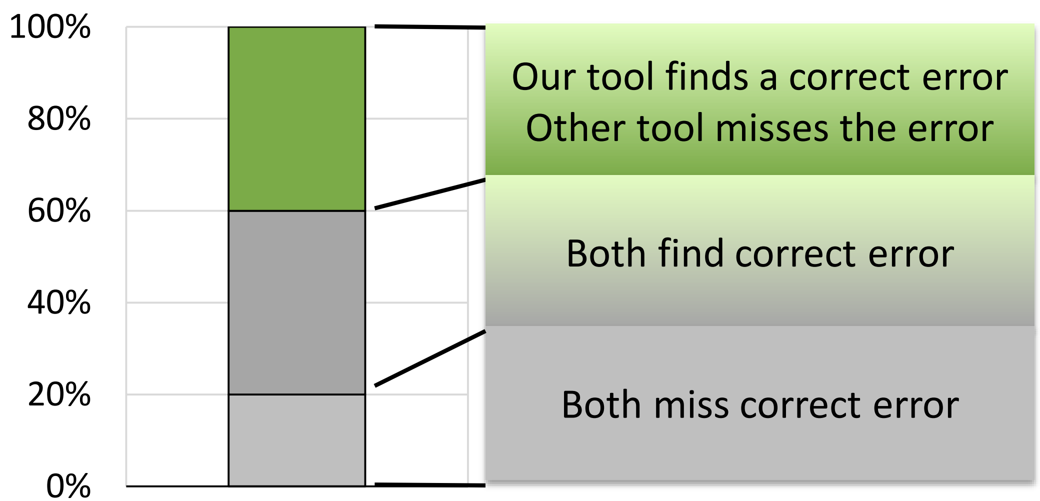
Comparison with the OCaml compiler

Comparison with the Seminal tool

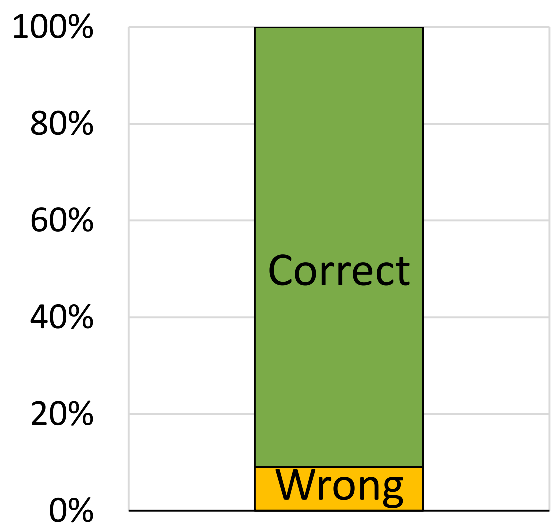
[Lerner et al.'07]

Comparison with Jif

- 16 previously collected buggy programs
  - An application with real-world security concern [Arden et al.'12]
  - Errors clearly marked by the application developer
  - Contains both error types



Comparison with the Jif compiler
(Wrong expression)



Accuracy on missing hypothesis

Related Work

- Program analyses as constraint solving [e.g., Aiken'99, Foster et al.'06]
  - No support for hypothesis; error report is verbose
- Diagnosing ML/Jif errors [e.g., McAdam'98, Heeren'05, Lerner'07, King'08, Chen&Erwig'14]
  - Tailored to specific program analysis
- Probabilistic inference [e.g., Ball et al.'03, Kremenek et al.'06, Livshits et al.'09]
  - Different contexts; errors are considered in isolation
- Diagnosing false alarms [e.g., Dillig et al.'12, Blackshear and Lahiri'13]
  - Does not diagnose true errors in program

Future Work

- More expressive language
  - Add arithmetic to the language
- Refine the simplifying assumptions
  - Remove assumptions on error independence
  - Incorporate domain specific knowledge

Conclusion

Program Analyses

- ML Type Inference
- Information-flow analysis
- Dataflow analysis

General diagnosis of static errors

- Applies to a large class of program analyses
- Diagnoses the cause of both true errors and false alarms
- Bayesian reasoning => more accurate reports than with existing tools

A demo is available at: http://apl.cs.cornell.edu/~zhangdf/diagnostic