East Carolina UNIVERSITY

Capturing Programmer Intent with Extensible Annotation Systems

Mark Hills

OOPSLE 2014: Open and Original Problems in Software Language Engineering
February 3, 2014
Antwerp, Belgium

http://www.rascal-mpl.org

Overview

- Representing Implicit Information in Programs
- Policy Frameworks
- Questions and Open Problems

East Carolina UNIVERSITY

Overview

- Representing Implicit Information in Programs
- Policy Frameworks
- Questions and Open Problems

East Carolina UNIVERSITY

Initial Motivation: Units of Measurement

"NASA lost a $125 million Mars orbiter because one engineering team used metric units while another used English units for a key spacecraft operation ... For that reason, information failed to transfer between the Mars Climate Orbiter spacecraft team at Lockheed Martin in Colorado and the mission navigation team in California."

Picture and text from CNN.com, "NASA's metric confusion caused Mars orbiter loss", http://www.cnn.com/TECH/space/9909/30/mars.metric

The Root of the Problem: Implicit Information

- Each developer knew what units they were personally working with

- But, they had no way to write them down

- Also, they assumed they knew what units other code used

- But, they couldn’t check to see if they were right

East Carolina UNIVERSITY

Question: What units are used here?

```
typedef struct {
  double atomicWeight;
  double atomicNumber;
} Element;

double radiationLength(Element * material) {
  double A = material->atomicWeight;
  double Z = material->atomicNumber;
  double L = log( 184.15 / pow(Z, 1.0/3.0) );
  double Lp = log( 1194.0 / pow(Z, 2.0/3.0) );
  return ( 4.0 * alpha * re * re) * ( NA / A ) *
         ( Z * Z * L + Z * Lp );
}
```

East Carolina UNIVERSITY

First Rewriting Logic Semantics Approaches

- Unit checker for BC [Chen et al, RTA'03]
- Unit checker for small subset of C [Rosu and Chen, ASE'03]
- Approach in both: added annotations in comments for specifying unit properties
- Whole program analysis, abstract evaluation semantics
- Note: lots of related work outside of RLS, part of what makes this problem interesting — lots of different approaches!

East Carolina UNIVERSITY

What's Wrong? Early work was not scalable!

- Major rework needed to extend semantics
- New analysis, new language features == massive changes
- Could not share specification fragments between analyses
- Whole program analysis: not scalable for users
- New Goal: build a semantics-based, modular analysis framework

East Carolina UNIVERSITY

Overview

- Representing Implicit Information in Programs
- Policy Frameworks
- Questions and Open Problems

East Carolina UNIVERSITY

Solution: Policy Frameworks!

- Modular static analysis framework
- Built in Maude with K-style rewriting logic semantics
- Language generic: analysis domains
- Language-specific, analysis-generic: base semantics, annotation-aware parser
- Analysis-specific: analysis semantics, annotation language

East Carolina UNIVERSITY

A Prototype Implementation: CPF

- CPF: C Policy Framework, analysis policies for units of measurement and pointer analysis [Hills et. al, RULE'08]

- Worked on real C code, found unit bugs seeded in NASA test code (C++ converted to C)

```
//@ pre(UNITS): unit(material->atomicWeight) = kg
//@ pre(UNITS): unit(material->atomicNumber) = noUnit
//@ post(UNITS): unit(@result) = m ^ 2 kg ^ -1
double radiationLength(Element * material) {
  double A = material->atomicWeight;
  double Z = material->atomicNumber;
  double L = log( 184.15 / pow(Z, 1.0/3.0) );
  double Lp = log( 1194.0 / pow(Z, 2.0/3.0) );
  return ( 4.0 * alpha * re * re) * ( NA / A ) *
         ( Z * Z * L + Z * Lp );
}
```

SILF-PF

- SILF-PF: SILF Policy Framework, policies for units and types [Hills and Rosu, RTA'10]

- Annotations added as language constructs and types

- Units domain shared between C and SILF

```
function main(void)
begin
  var x; var y; var n;
  assume(UNITS): @unit(x) = $m;
  assume(UNITS): @unit(y) = $m;
  for n := 1 to 10
    invariant(UNITS): @unit(x) = @unit(y);
  do
    x := x * x;
    y := y * y;
  od
  write x + y;
end
```

East Carolina UNIVERSITY

Did this Work?

- Reuse of modules achieved in both CPF and SILF-PF
- Reuse of annotation-aware frontends for both policy frameworks (CIL for CPF, custom for SILF-PF)
- UNITS analysis domain shared between frameworks for SILF and C

East Carolina UNIVERSITY

So what's still wrong?

- Need to define "boilerplate" functionality to interact with existing framework

- Need to know which extension points are available

- Need to write lots of redundant cases for error propagation

- Need to define custom annotation languages and parsers

East Carolina UNIVERSITY

What happened?

- We’ve transformed a specification challenge into a software engineering challenge

- More modular in many ways, but not always in practice for the users

East Carolina UNIVERSITY

Overview

- Representing Implicit Information in Programs
- Policy Frameworks
- Questions and Open Problems

East Carolina UNIVERSITY

Questions and Discussion Topics

- Q: How do we enable the easier construction of custom specification languages (including the semantics behind them)?
- Q: How do we do this in a way that supports non-experts? Is this even possible?
- Q: Are there language features that would make this process easier? Can we even impose them?
- Q: What domains outside units can we apply this to?

East Carolina UNIVERSITY

An idea: apply SLE to annotation languages!

- Provide targeted DSLs for different parts of definition
  - Raise level of abstraction
  - Provide reuse between language frameworks
  - Provide clean separation of concerns between different tool aspects
  - Generate complex parts of the specification

East Carolina UNIVERSITY

How do policies vary?

- Annotation languages
- Abstract value domains
- Memory layouts
- Rule definitions

```
//@ pre(UNITS): unit(material->atomicWeight) = kg
//@ pre(UNITS): unit(material->atomicNumber) = noUnit
//@ post(UNITS): unit(@result) = m ^ 2 kg ^ -1
double radiationLength(Element * material) {
  double A = material->atomicWeight;
  double Z = material->atomicNumber;
  double L = log( 184.15 / pow(Z, 1.0/3.0) );
  double Lp = log( 1194.0 / pow(Z, 2.0/3.0) );
  return ( 4.0 * alpha * re * re) * ( NA / A ) *
         ( Z * Z * L + Z * Lp );
}
```

East Carolina UNIVERSITY

Can we make specifications “self-aware”?

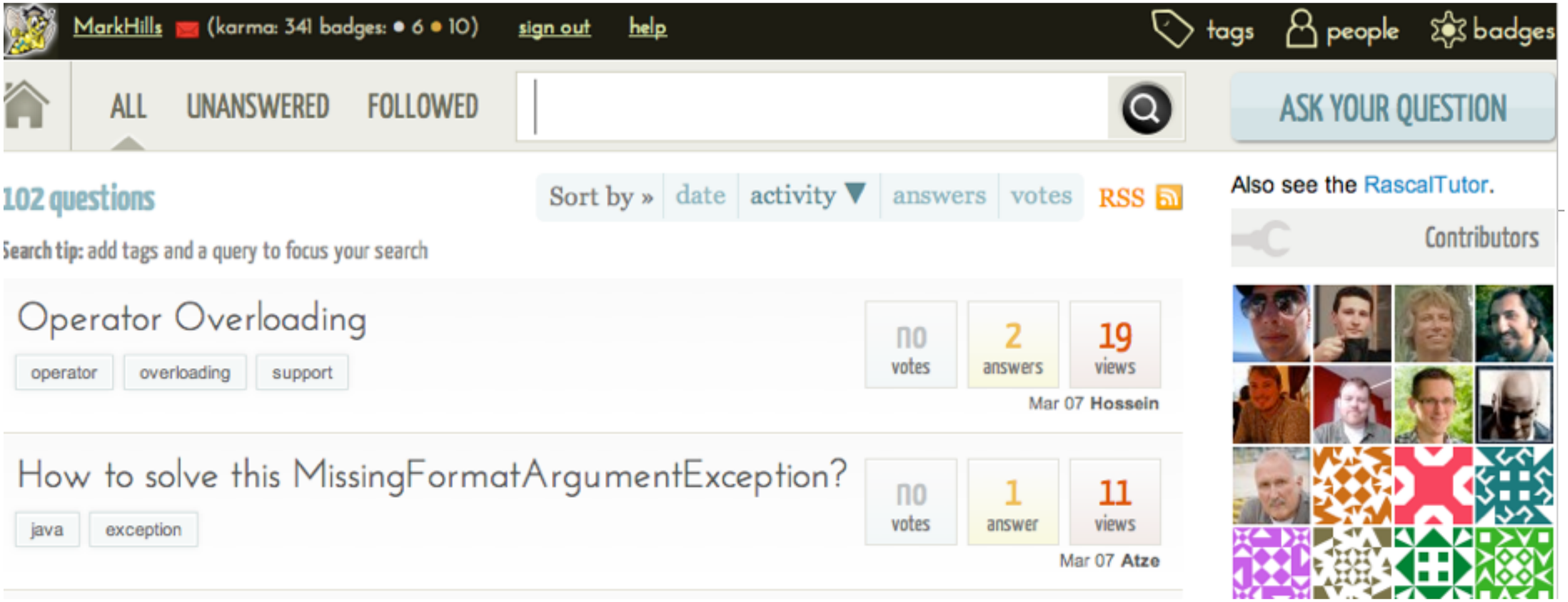
- New policies extend existing extension points to “plug in” to framework
- Can we allow formal specifications to reason about where they can be extended?
- Rewriting logic is reflective, should be achievable!

East Carolina UNIVERSITY

Other domains with implicit information?

- Note: focus is on domains that do not appear in the code at all (null values do, meters don’t)
- Scientific applications: coordinate systems, angular measurements (not officially units!)
- Auditing applications (e.g., Next Generation Auditing project with CWI and several partners): currency, sales/stocking units
- Empirical software engineering: units related to empirical properties of programs, like lines of code, statement counts, number of commits

East Carolina UNIVERSITY



- Me: http://www.cs.ecu.edu/hillsma

- Rascal: http://www.rascal-mpl.org

East Carolina University

Why Pick Units of Measurement?

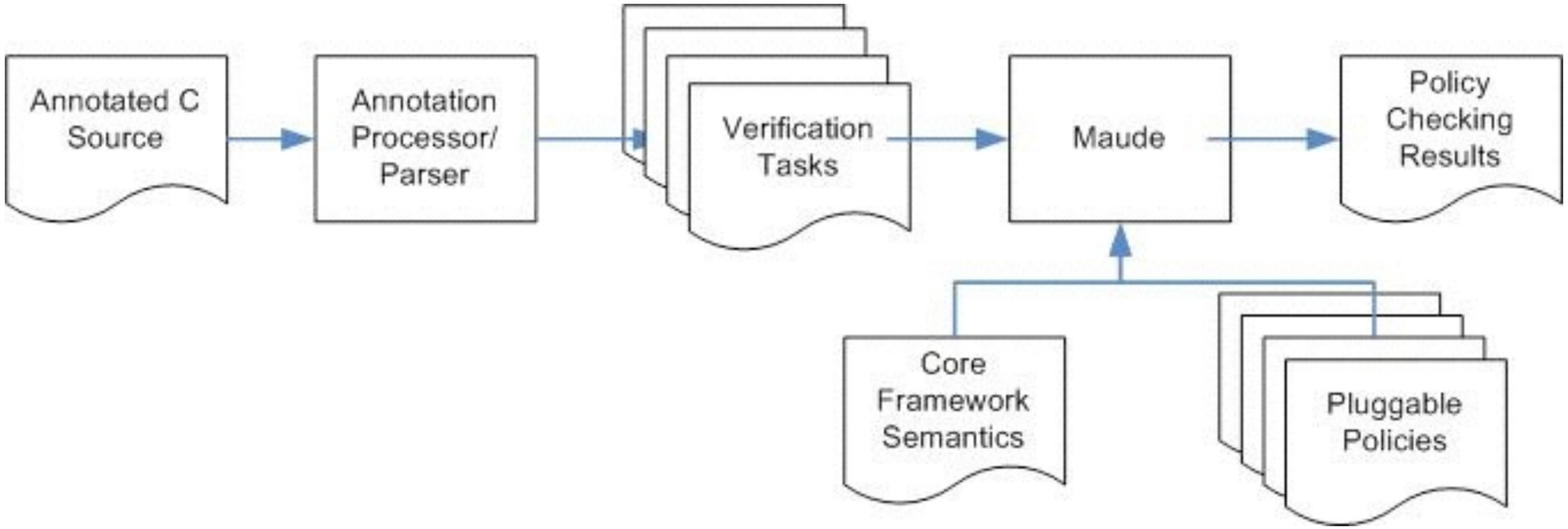
- Tangible: unit safety violations have caused some well-known malfunctions; units used in many applications

- Interesting: has been the focus of much research, many different possible approaches

- Challenging: units have equational properties (not standard types); software in scientific domains can be hard to analyze (C, C++, Fortran, etc...)

East Carolina UNIVERSITY

CPF: Architectural View



East Carolina UNIVERSITY

Why is this a problem?

- CPF Core: 69 modules, 548 ops, 586 equations, 2016 lines
- CPF Units: 22 modules, 56 ops, 291 equations, 805 lines
- More than 100 “hooks” for policy-specific semantics

East Carolina UNIVERSITY

Some opportunities for DSLs...

- Annotation languages
- Abstract value domains
- Memory layouts
- Rule definitions/skeletons
- Control Flow Graph construction
- Intermediate code generation (for program analysis)

East Carolina UNIVERSITY