Streamlining Policy Creation in Policy Frameworks

Mark Hills

21st International Workshop on Algebraic Development Techniques
June 8, 2012
Salamanca, Spain

http://www.rascal-mpl.org

Overview

- Policy Frameworks
- Challenges
- Adding Support for Extensibility

Overview

- Policy Frameworks
- Challenges
- Adding Support for Extensibility

Initial Motivation

- Units of measurement are important!

- Initial work: built units checkers for BC and for a small subset of C

1. Feng Chen, Grigore Rosu, and Ram Prasad Venkatesan. *Rule-Based Analysis of Dimensional Safety.* In *Proceedings of RTA'03.*

2. Grigore Rosu and Feng Chen. *Certifying Measurement Unit Safety Policy.* In *Proceedings of ASE'03.*

Why That Wasn’t Enough

- Early work was not modular
- Could not easily extend semantics (e.g., cover more of C)
- Could not add new analyses
- Could not share specification fragments between analyses
- Goal: build a semantics-based, modular analysis framework

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

CPF and SILF-PF

- CPF: C Policy Framework, analysis policies for units of measurement and pointer analysis
- Worked on real C code, found unit bugs seeded in NASA test code (C++ converted to C)
- SILF-PF: SILF Policy Framework, policies for units and types
- Units domain shared between C and SILF

3. Mark Hills, Feng Chen, and Grigore Rosu. *A Rewriting Logic Approach to Static Checking of Units of Measurement in C.* In *Proceedings of RULE'08.*

4. Mark Hills and Grigore Rosu. *A Rewriting Logic Semantics Approach To Modular Program Analysis.* In *Proceedings of RTA'10.*

Overview

- Policy Frameworks
- Challenges
- Adding Support for Extensibility

Modularity Works, so What’s Wrong?

- Transformed specification challenge into software engineering challenge!
- Need to define “boilerplate” functionality to interact with existing framework
- Need to know which hooks are available for extension
- Need to know what modules can be extended
- Need to write lots of redundant cases for error propagation
- Need to define custom annotation languages and parsers

Overview

- Policy Frameworks
- Challenges
- Adding Support for Extensibility

Define Functionality to Interact with Framework

- Analysis domains based on definition of Policy Values
- Multiple policies can be active at once, need to generate annotation filters
- Need to define pretty-printing for error message generation

Current Code: Defining Types in SILF

```
ops $int $bool : -> BaseType .
op $notype : -> PolicyVal .
op $array : BaseType -> PolicyVal .

eq pv2pv($('int)) = $int .
eq pv2pv($('bool)) = $bool .
eq pv2pv($('array) ( T ) ) = $array(pv2pv(T)) .

eq ta2pv($('int)) = $int .
eq ta2pv($('bool)) = $bool .
eq ta2pv($('array) ( T ) ) = $array(ta2pv(T)) .

eq pretty-print($int) = "$int" .
eq pretty-print($bool) = "$bool" .
eq pretty-print($notype) = "$notype" .
eq pretty-print($array(T)) = "$array(" + pretty-print(T) + ")" .
```

Proposed Code: Defining Policies in a Policy DSL

```
Policy TYPES

PolicyVal $int;
PolicyVal $bool;
PolicyVal $noType;
PolicyVal $array(PolicyVal as pv) display as "$array[<$pv>]";

End Policy
```

Policy Name Provides Filtering

Default Pretty Printing Rules

Annotation Filtering Rules Generated

Custom Pretty Printing Rule

Which Hooks Can Be Extended?

- Extension points, i.e. “hooks”, are operators with no defining equations
- New policies provide equations to add functionality
- How to find hooks? all ops in a module? all ops of a given sort or sorts?

Proposed Solution: Maude Reflection

Sample Hook Definitions

```
op defaultIntVal : -> Value [metadata "hook"] .
ops + - * / % : Exp Exp -> ComputationItem [metadata "hook"] .

Maude> red hookRelToRascal(computeHookRel('GENERIC-ARITH-SEMANTICS)) .
reduce in HOOK-OPS : hookRelToRascal(computeHookRel('GENERIC-ARITH-
SEMANTICS))
    .
rewrites: 201 in 0ms cpu (0ms real) (11823529 rewrites/second)
result String: "[hook(\"GENERIC-ARITH-SEMANTICS\",\"%\",[\"Exp\",\"Exp\"],
\"ComputationItem\"), hook(\"GENERIC-ARITH-SEMANTICS\",\"*\",[\"Exp\",\"Exp\"],
\"ComputationItem\"),
hook(\"GENERIC-ARITH-SEMANTICS\",\"+\",[\"Exp\",\"Exp\"],\"ComputationItem\"),
hook(\"GENERIC-ARITH-SEMANTICS\",\"-\",[\"Exp\",\"Exp\"],\"ComputationItem\"),
hook(\"GENERIC-ARITH-SEMANTICS\",\"/\",[\"Exp\",\"Exp\"],\"ComputationItem\"),
hook(\"GENERIC-ARITH-SEMANTICS\",\"u-\",[\"Exp\"],\"ComputationItem\")]"
```

Extraction from Maude

Proposed Solution: A Policy Rule Definition DSL

```
Policy SILF-TYPES

prule[GENERIC-ARITH-SEMANTICS, + : Exp Exp -> Exp]:
   k(val(V1,V2) -> +(E1,E2) -> K) = k(K)
   if notfail(V1) and notfail(V2) .

 prule[GENERIC-ARITH-SEMANTICS, + : Exp Exp -> Exp]:
   k(val(V1,V2) -> +(E1,E2) -> K) =
   k(mergefail(V1,V2) -> K) if fail(V1) or fail(V2) .

End Policy
```

Extraction generates default equations that do nothing

Need to add better notation for error propagation: still working on this (currently done by writing more equations)

Limitation: don't want to reparse Maude, so the body isn't checked...

Which Modules Can Be Extended?

- For now, just relying on modularity features of Maude, plus documentation
- Generally one feature or feature "group" (e.g., arithmetic expressions) per module
- So, leaving this as is (but, still a future challenge -- how can we make module reuse easier?)

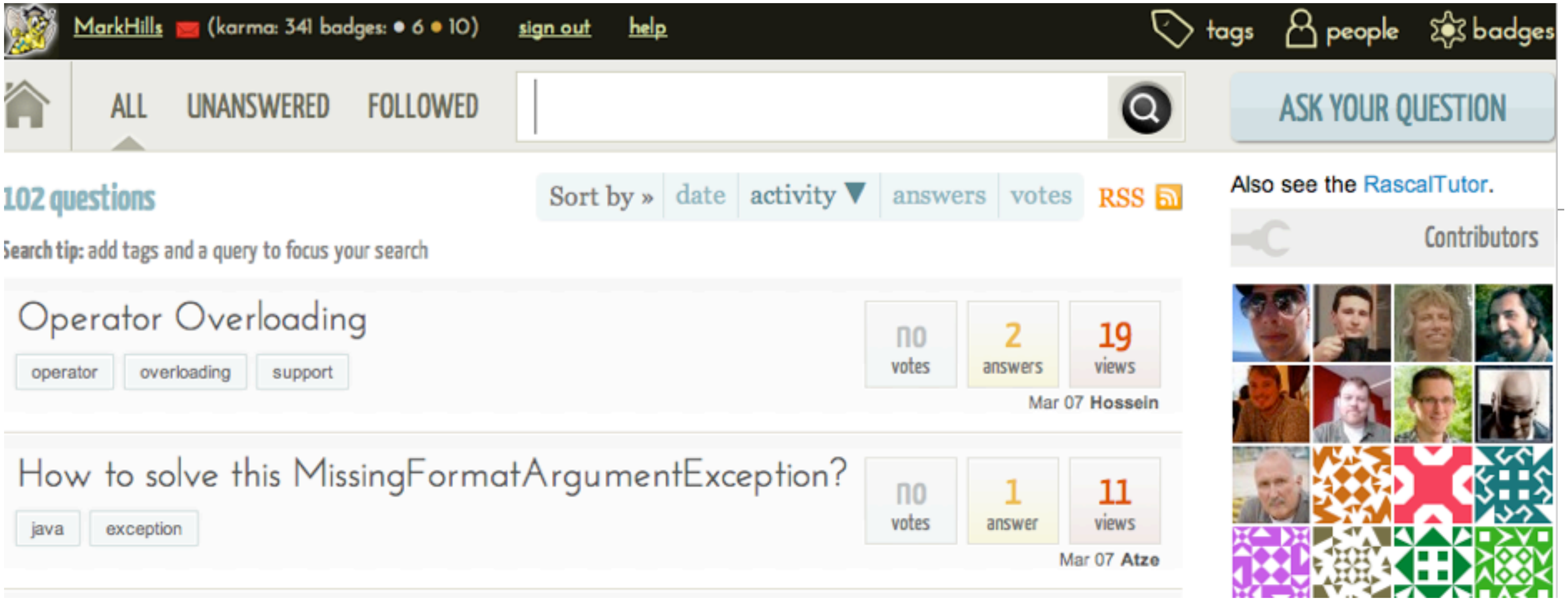
Open For Debate!

One More: Annotation Languages

- Language parser must be annotation language generic
- Current solution: pass annotation language fragments as strings to a parser for the policy
- In progress: convert parsing to using Rascal, GLL can combine grammars, provide for filtering rules
- Currently works for SILF, not yet in C
- In progress: link to Maude annotation language definitions (including shared definitions)
- Ideal: generate parser and Maude definition from same code

Wrap-Up: Further Challenges

- Should extraction support be extended to other operators?
- Declarations need more support, especially in languages like C
- Don’t want to rebuild Maude parser in Rascal! But how to best support analysis builders?


MarkHills (karma: 341 badges: ● 6 ● 10) sign out help

tags people badges

ALL UNANSWERED FOLLOWED

ASK YOUR QUESTION

102 questions

Sort by » date activity ▼ answers votes RSS

Search tip: add tags and a query to focus your search

Operator Overloading — operator, overloading, support — no votes, 2 answers, 19 views — Mar 07 Hossein

How to solve this MissingFormatArgumentException? — java, exception — no votes, 1 answer, 11 views — Mar 07 Atze

Also see the RascalTutor.

Contributors

- Rascal: http://www.rascal-mpl.org
- SEN1: http://www.cwi.nl/sen1
- Me: http://www.cwi.nl/~hills