

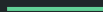
Implementing a portfolio-solving tactic for Why3

Andrew Healy, Dr. Rosemary Monahan, Dr. James F. Power
Dept. of Computer Science, Maynooth University
Postgraduate Workshop, 15th March 2016

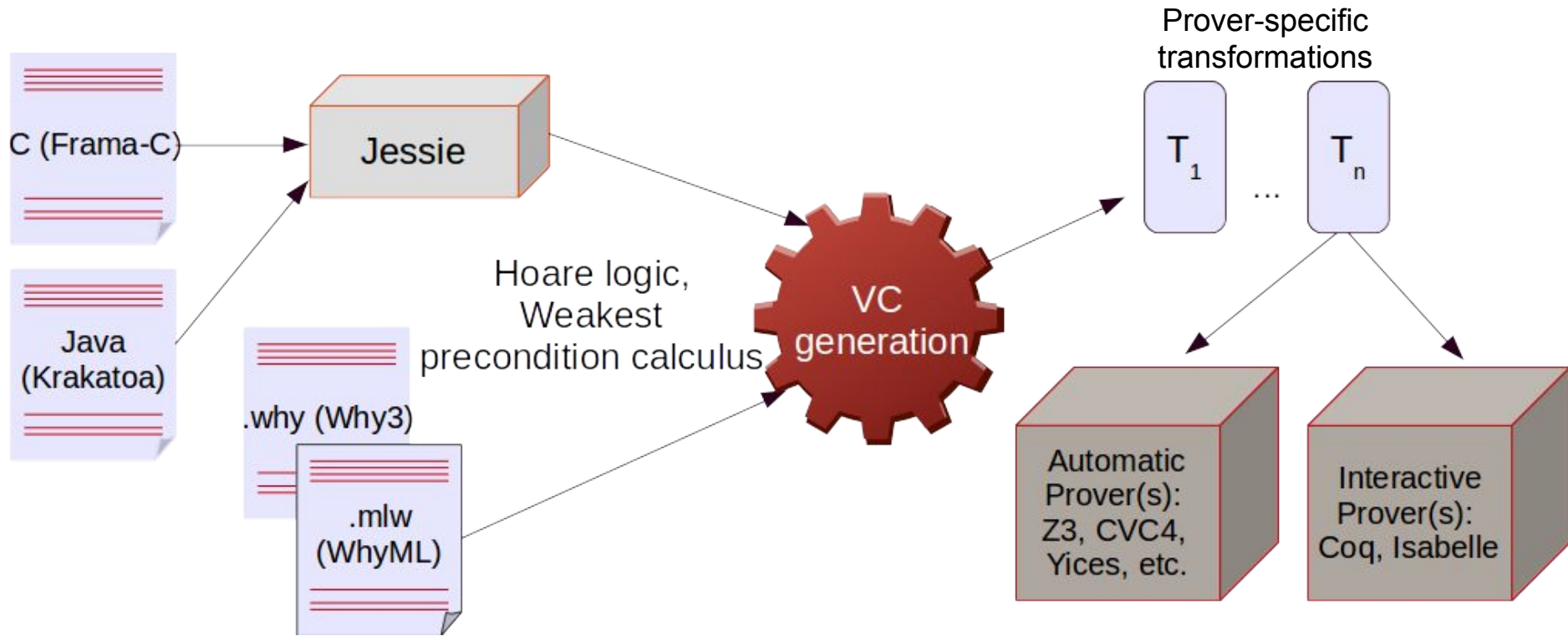
Can Why3 be trained
to delegate proof
tasks efficiently?

Outline

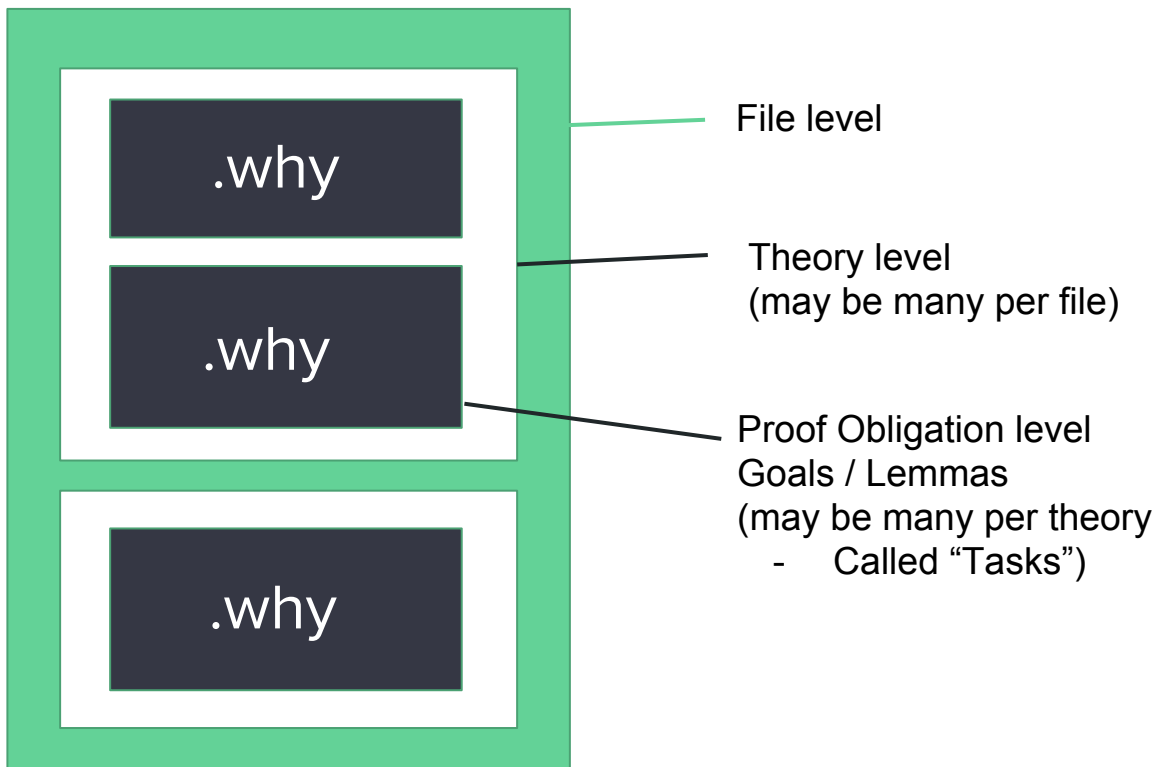
1. (Re)-introducing Why3
2. Describing the problem
3. Profiling Workload
4. Related Approaches + the Future



1. (Re)-Introducing Why3



Last Year: Why3 Architecture and Workflow



Anatomy of a Why3ML file

.why

Proof Obligation level
Goals / Lemmas
(may be many per theory
- Called "Tasks")

.smt2

CVC4, veriT,
Z3

.cvc

CVC3

.yics

Yices

.why

Alt-Ergo (old
syntax)

...

Others: .tptp,
.gappa, etc.

From one, many - *ex uno plures*

2. Describing the Problem

Why3 Interactive Proof Session

Context

☐ Unproved goals

☒ All goals

Strategies

Compute

Inline

Split

Provers

Alt-Ergo (0.95.2)

CVC3 (2.4.1)

CVC4 (1.4)

Eprover (1.8-001)

Gappa (1.2.0)

MathSAT5 (5.2.2)

MetiTarski (2.4)

Psyche (2.0)

Spass (3.7)

Yices (1.0.40)

Z3 (4.3.2)

veriT (201506)

Tools

Edit

Replay

Remove

Theories/Goals	Status	Time
arm.mlw	✓	4.05
M	✓	3.86
VC for insertion_sort	✓	3.86
Alt-Ergo (0.95.2)	✓	3.60 (steps: 74)
Z3 (4.3.2)	✗	0.26
CVC4 (1.4)	✗	5.05
CVC3 (2.4.1)	?	0.39
ARM	✓	0.00
InsertionSortExample	✓	0.19
VC for path_init_I2	✓	0.16
Alt-Ergo (0.95.2)	✓	0.16 (steps: 19)
Z3 (4.3.2)	?	0.01
CVC4 (1.4)	?	0.02
CVC3 (2.4.1)	?	1.84
VC for path_I2_exit	✓	0.02
Z3 (4.3.2)	✓	0.01
Alt-Ergo (0.95.2)	✓	0.00 (steps: 9)
CVC4 (1.4)	✓	0.01
CVC3 (2.4.1)	✓	0.00

Source code

Task

Edited proof

Prover Output

file: ../why3-0.86.2/examples/arm/./arm.mlw

```

1 (* experiments related to ARM program verification *)
2
3 module M
4
5   use import int.Int
6   use import ref.Refint
7   use import array.Array
8
9   val a : array int
10
11   predicate inv (a : array int) =
12     a[0] = 0 /\ length a = 11 /\ forall k:int. 1 <= k <= 10 ->
13
14   val ghost loop1 : ref int
15   val ghost loop2 : ref int
16
17   let insertion_sort ()
18     requires { inv a /\ !loop1 = 0 /\ !loop2 = 0 }
19     ensures { !loop1 = 9 /\ !loop2 <= 45 }
20     = let i = ref 2 in
21       while !i <= 10 do
22         invariant { 2 <= !i <= 11 /\ inv a /\
23                   !loop1 = !i - 2 /\ 2 * !loop2 <= (!i-2) * (
24                   variant { 10 - !i }
25                   ghost incr loop1;
26                   let j = ref !i in
27                   while a[!j] < a[!j - 1] do
28                     invariant { 1 <= !j <= !i /\ inv a /\
29                               2 * !loop2 <= (!i-2) * (!i-1) + 2*(!i - !
30                     variant { !j }
31                     ghost incr loop2;
32                     let temp = a[!j] in
33                     a[!j] <- a[!j - 1];
34                     a[!j - 1] <- temp;
35                     decr j
36                   done;
37                   incr i
38                 done

```

What makes one prover better than another?

The problem:

- Difficulty in comparing SMT provers due to the diverse range of input languages and logical specialisations [1]
- We need some way of categorising programs in order to reason about provers
- Knowledge of each prover's strengths and limitations is needed in order to verify programs

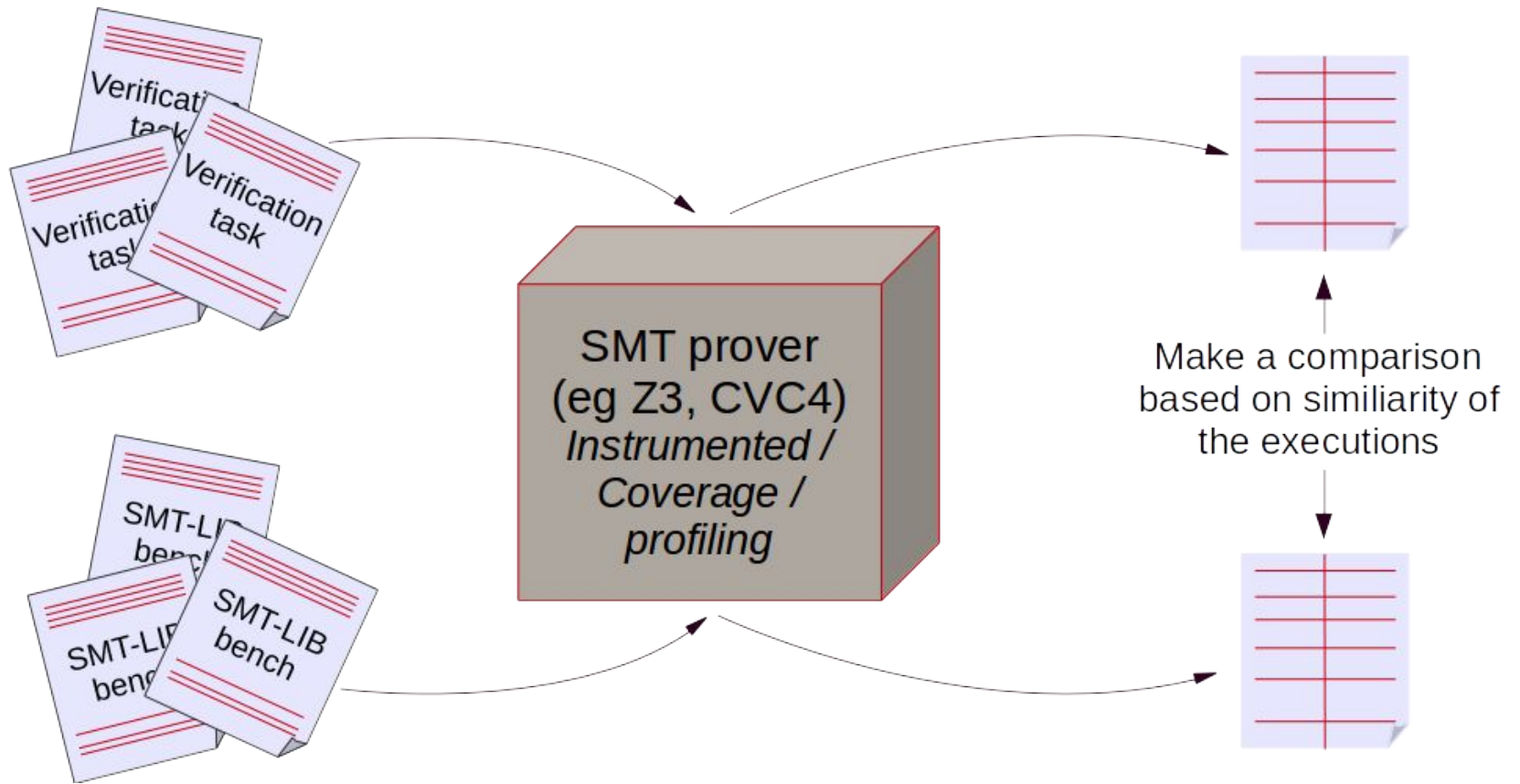
The problem:

- Difficulty in comparing SMT provers due to the diverse range of input languages and logical specialisations [1]
- We need some way of categorising programs in order to reason about provers
- Knowledge of each prover's strengths and limitations is needed in order to verify programs

A solution?

Use techniques from data analysis and machine learning to predict the most successful prover for a given problem

3. Profiling Workload



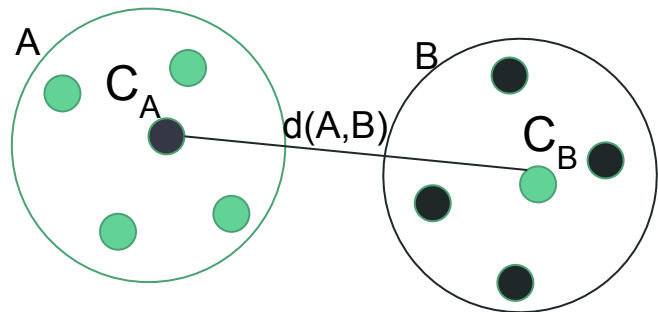
Last Year: A Proposed Workflow

Actual Workflow:

1. Profile programs from the Why3 examples and SMTLIB repo with “callgrind”
2. Filter system calls to maintain platform independence
3. Perform dimensionality reduction via PCA
4. Assuming the Why3 programs are a representative “verification cluster”, measure its distance:
 - a. To randomly-sampled SMTLIB programs
 - b. To sets of SMTLIB programs composed according to the inherent hierarchy
 - c. To a workload-based clustering of SMTLIB programs

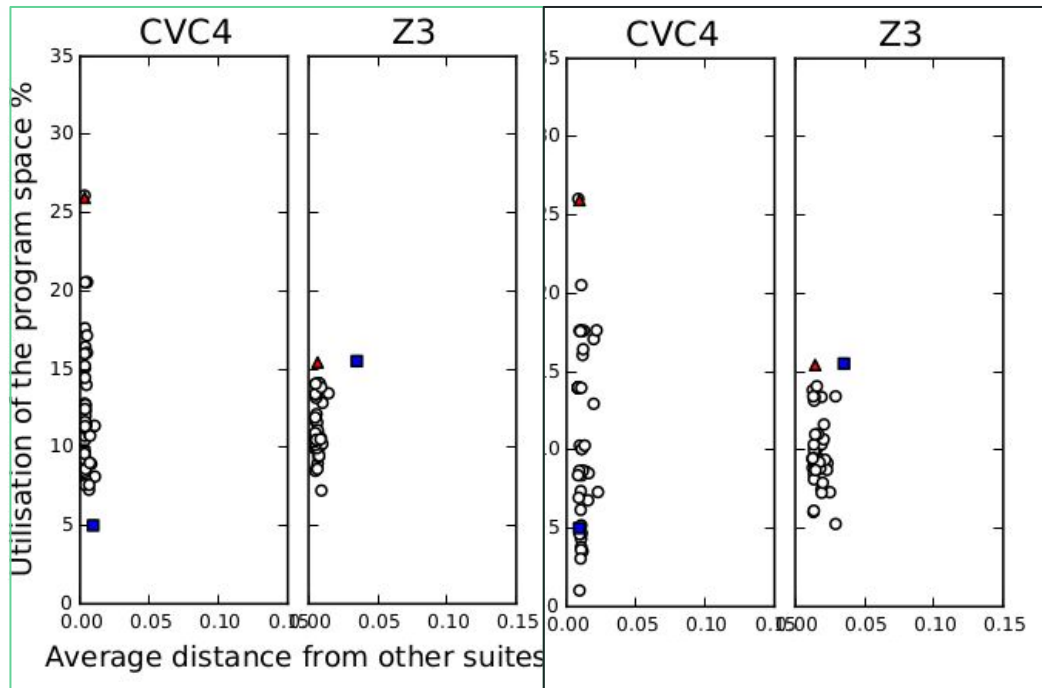
```
338,423,234 * /build/builddd/eglibc-2.19/malloc/malloc.c:_int_malloc [/lib/x86_64-linux-gnu/libc-2.19.so]
4,736 < /build/builddd/eglibc-2.19/malloc/malloc.c:_int_malloc (50x) [/lib/x86_64-linux-gnu/libc-2.19.so]
316,224 < /build/builddd/eglibc-2.19/malloc/malloc.c:_int_realloc (4142x) [/lib/x86_64-linux-gnu/libc-2.19.so]
193,924,776 < /build/builddd/eglibc-2.19/malloc/malloc.c:_free (2540202x) [/lib/x86_64-linux-gnu/libc-2.19.so]
190,082,586 * /build/builddd/eglibc-2.19/malloc/malloc.c:_int_free [/lib/x86_64-linux-gnu/libc-2.19.so]
6,837,856 < /build/builddd/eglibc-2.19/malloc/malloc.c:_realloc (50912x) [/lib/x86_64-linux-gnu/libc-2.19.so]
1,092 < /home/andrew/Documents/cvc4-1.4/builds/x86_64-unknown-linux-gnu/production/../../../../src/parser/bounded_token_factory.cpp:Bour
/lib/libcvc4parser.so.3.0.0]
157,796 < /home/andrew/Documents/cvc4-1.4/builds/x86_64-unknown-linux-gnu/production/../../../../src/context
/context_mm.cpp:CVC4::context::ContextMemoryManager::newData(unsigned long) (318x) [/usr/local/lib/libcvc4.so.3.0.0]
194 < /home/andrew/Documents/cvc4-1.4/builds/x86_64-unknown-linux-gnu/production/../../../../src/expr/node_manager.h:CVC4::TypeNode
CVC4::NodeManager::mkConstInternal<CVC4::TypeNode, CVC4::BitVectorSize>(CVC4::BitVectorSize const&) (2x) [/usr/local/lib/libcvc4.so.3.0.0]
10,567,460 < /home/andrew/Documents/cvc4-1.4/builds/x86_64-unknown-linux-gnu/production/../../../../src/expr/node_builder.h:CVC4::NodeBuilder
[/usr/local/lib/libcvc4.so.3.0.0]
324 < ???::antlr3VectorResize (2x) [/usr/local/lib/libcvc4parser.so.3.0.0]
368 < ???::antlr3LexerNew (2x) [/usr/local/lib/libcvc4parser.so.3.0.0]
822 < ???::newRaw0 (4x) [/usr/local/lib/libcvc4parser.so.3.0.0]
386 < /home/andrew/Documents/cvc4-1.4/builds/x86_64-unknown-linux-gnu/production/../../../../src/expr/expr/node_manager.h:CVC4::Not
CVC4::NodeManager::mkConstInternal<CVC4::NodeTemplate<true>, CVC4::String>(CVC4::String const&) (2x) [/usr/local/lib/libcvc4.so.3.0.0]
459,658 < /home/andrew/Documents/cvc4-1.4/builds/x86_64-unknown-linux-gnu/production/../../../../src/expr/expr/node_builder.h:CVC4::Not
[/usr/local/lib/libcvc4.so.3.0.0]
386 < ???::antlr38BitMark (2x) [/usr/local/lib/libcvc4parser.so.3.0.0]
736 < ???::antlr3BaseRecognizerNew (4x) [/usr/local/lib/libcvc4parser.so.3.0.0]
530 < /home/andrew/Documents/cvc4-1.4/builds/x86_64-unknown-linux-gnu/production/../../../../src/options/options_template.cpp:CVC4::Options::parse
/lib/libcvc4.so.3.0.0]
772 < /home/andrew/Documents/cvc4-1.4/builds/x86_64-unknown-linux-gnu/production/../../../../src/expr/node_manager.h:CVC4::NodeTemplat
CVC4::NodeManager::mkConstInternal<CVC4::NodeTemplate<true>, CVC4::Rational>(CVC4::Rational const&) (4x) [/usr/local/lib/libcvc4.so.3.0.0]
386 < ???::antlr3ParserNew (2x) [/usr/local/lib/libcvc4parser.so.3.0.0]
772 < /home/andrew/Documents/cvc4-1.4/builds/x86_64-unknown-linux-gnu/production/../../../../src/parser/bounded_token_buffer.cpp:Bounc
/lib/libcvc4parser.so.3.0.0]
368 < ???::antlr3CommonTokenStreamNew (2x) [/usr/local/lib/libcvc4parser.so.3.0.0]
```

Distance Measurement & Results



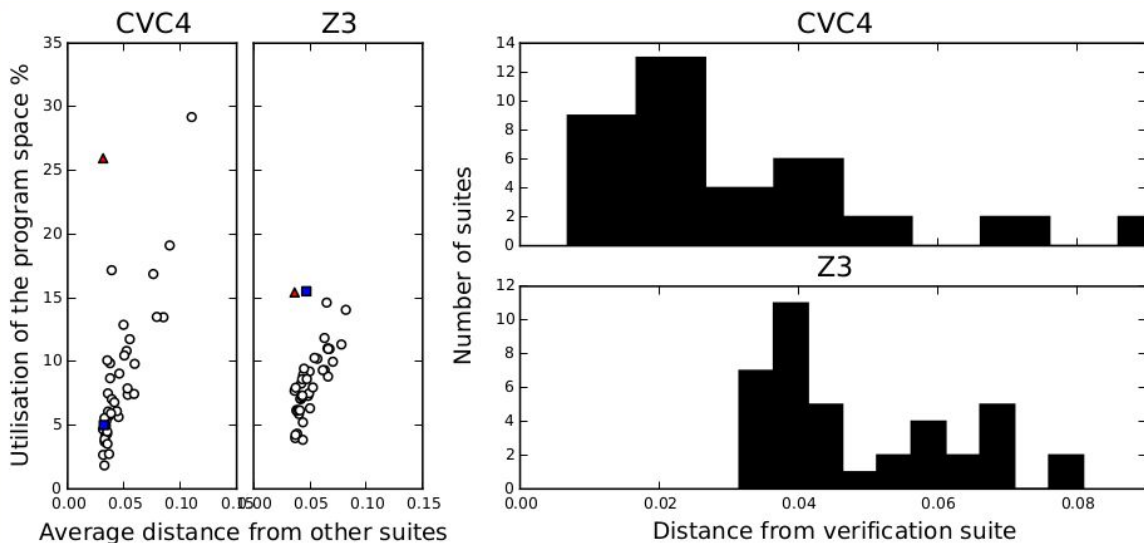
We used techniques and metrics described by Jozo Dujmovic in [2] to accurately characterise and describe machine-independent workloads.

$$diff(A, B) = \frac{1}{2} \sum_{i=0}^N \left| p_i^{(A)} - p_i^{(B)} \right|, sim(A, B) = 1 - diff(A, B)$$



What we learned

- Z3 uses more of its features during software verification tasks than CVC4 (experiment 1)
- The SMTLIB repository's structure is more closely aligned to how CVC4 functions (experiment 2)
- A workload-based clustering is more successful at identifying programs that are different from verification programs in the SMTLIB repository than those that are very similar



More details in [3]. Online data: <http://www.cs.nuim.ie/~ahealy/SAC2016>

Taking into account the result of the prover ...

... and the time the prover took to return a result...

Why3 presents an opportunity to combine all this relevant information to produce a useful tool.

4. Related Approaches

Some Portfolio Solvers and ML tools already exist

- **Sledgehammer** for the Isabelle/HOL interactive theorem prover - a Naive Bayes algorithm filters (previously-proved) facts for relevance, guiding interactive proving [3]
- **ML4PG** (Machine Learning for Proof General) Interfaces Coq proofs developed in Proof General to MATLAB/Weka clustering algorithms, again guiding interactive proofs [4]
- **Verifolio** extracts static metrics from C programs to choose the most likely ATP to use (via Support Vector Machines) [5]

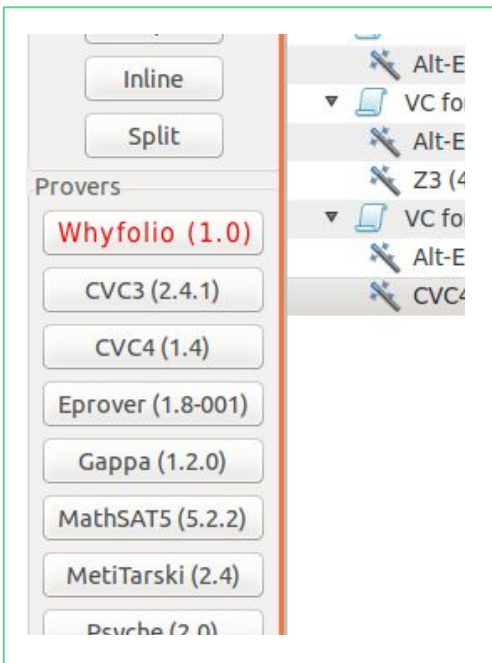
A Portfolio Solver for Why3

Two common input languages, three different ‘levels’ of viewing a program and an extensible driver-based architecture present an opportunity to develop a portfolio solver for SMT-based deductive program verification.


Current work: Implementing a version of process described in [5] through three experiments:

1. Using metrics derived from WhyML programs (contracts, invariants, quantifiers, shape analysis)
2. Using metrics derived from individual proof obligations (why3 logic syntax: predicate logic operators)
3. Comparing predictions from 1+2 to those obtained from instrumentation [3]

As a Why3 plugin



On the Cloud

why3 online**Maynooth University**
National University of Ireland Maynooth


Is this program valid?

```
1 @provers={Z3, CVC4}
2 @transform=[split]
3 (* The previous directives MUST be given at the start of the file
4   in order to be applied globally (ie: unless overridden locally - see G3 below) *)
5
6 (* List of accepted provers:
7   Alt-Ergo
8   CVC3
9   CVC4
10  Gappa
11  Simplify
12  veriT
13  Z3 *)
14
15 theory HelloProof
16   goal G1 : true
17   goal G2 : (true -> false) /\ (true /\ false)
18   use import int.Int
19   (* Z3 works better than CVC4 for quantifiers so let's use it
20    exclusively for G3. The following directive overrides the global provers *)
21   @provers={Z3}
```

DISCLAIMER: Why3 Online is a 3rd party tool offered by Maynooth University. By clicking '▶', you instruct Online to be analyzed. Please refer to the [terms of use](#) and [privacy policy](#) of Why3 Online. Contact [support](#)

[home](#) [permalink](#)

▶ shortcut: Alt+B



samples
Hello Proof
Timeout

about Why3 Online - The Why3 Platform online
Call multiple provers on verification goals written using polymorphic types

In a container



Contexts for our tool...

Thanks!

Any questions...

This material is based on
works carried out with
funding provided by the
Science Foundation
Ireland under grant
number 11/RFP.1/CMS/3068



ahealy@cs.nuim.ie

References

1. D. Beyer, M. Huisman, V. Klebanov, & R. Monahan. *Evaluating Software Verification Systems: Benchmarks and Competitions* (Dagstuhl Reports 14171). **Dagstuhl Reports**, 4(4):1–19, 2014
2. J. Dujmovic. *Universal benchmark suites - a quantitative approach to benchmark design*. In R. Eigenmann, ed., **Performance Evaluation and Benchmarking with Realistic Applications**. MIT Press, 2001.
3. A. Healy, R. Monahan, J. Power. *Evaluating the Use of a General-Purpose Benchmark Suite for Domain-Specific SMT-solving*, Proceeding of **Symposium on Applied Computing**, 2016.
4. D. Kuhlwein, J. C. Blanchette, C. Kaliszyk & J. Urban. *MaSh: Machine Learning for Sledgehammer*. In: 4th Conference on Interactive Theorem Proving (ITP'13), **Lecture Notes in Computer Science**.
5. E. Komendantskaya, J. Heras & G. Grov. *Machine Learning for Proof General: interfacing interfaces*. EPTCS Post-proceedings of **User Interfaces for Theorem Provers** (UITP 2012) see also ai4fm.org
6. Y. Demyanova, T. Pani, H. Veith, and F. Zuleger. *Empirical software metrics for benchmarking of verification tools*. In Computer Aided Verification, vol. 9206 of **Lecture Notes in Computer Science**.