Evaluating SMT solvers for software verification

Andrew Healy, Dr. Rosemary Monahan, Dr. James F. Power
Dept. of Computer Science, Maynooth University
BCTCS, Queen’s University Belfast, 22nd March 2016

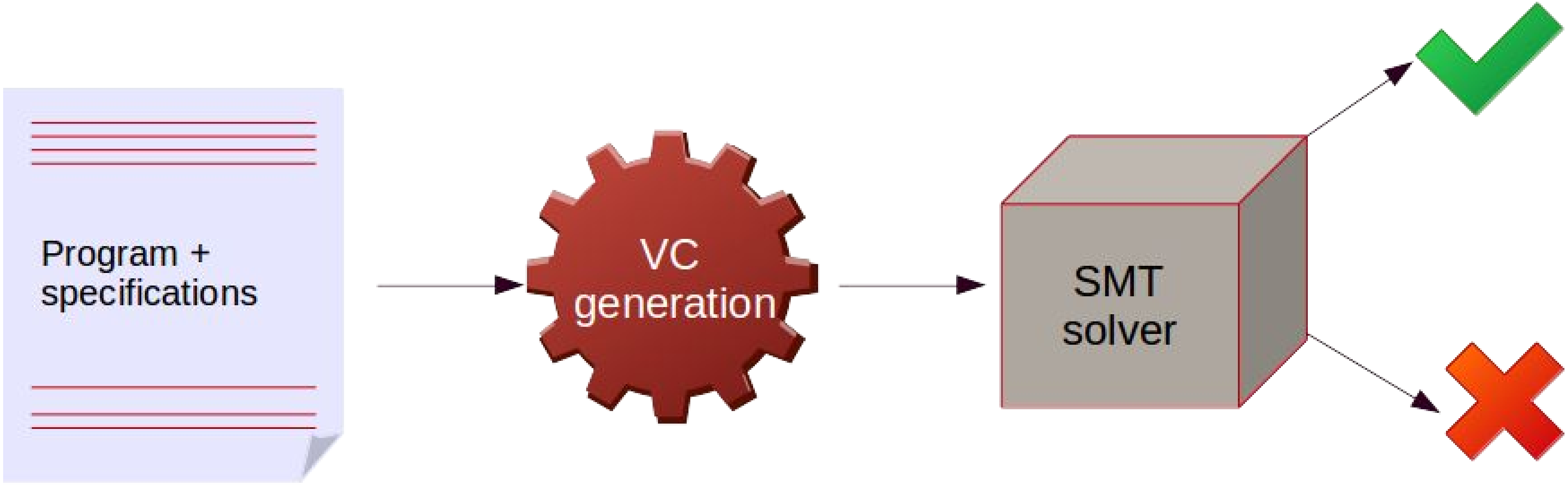
Can the workload of SMT solvers for software verification be characterised & predicted…?

…. If so, can Why3 be trained to delegate proof tasks efficiently?

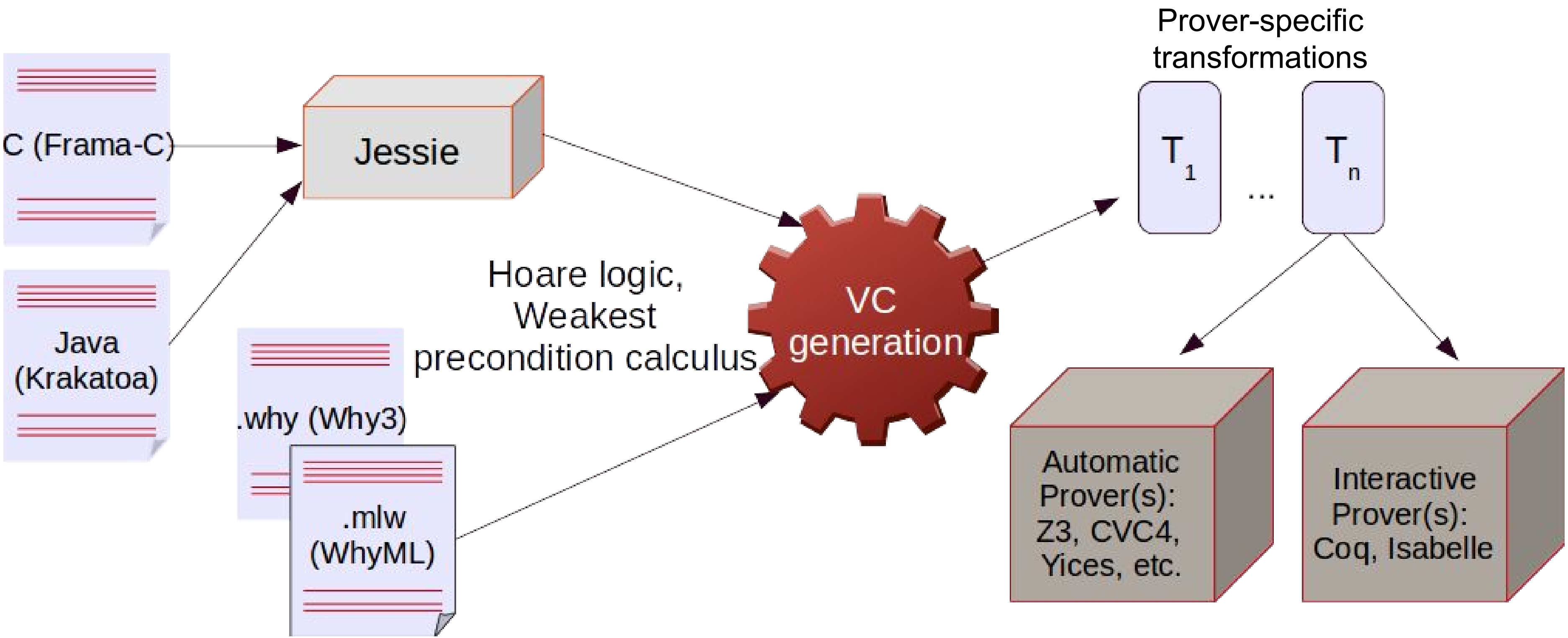
Outline

1. Why3’s approach to SV
2. The Need for Benchmarks
3. Previous Experiments
4. Why3 demonstration
5. Related Approaches + the Future

1. Why3's approach to Software Verification



Usually, there is a close integration between specification language, VC generator and SMT solver.



Why3 Architecture and Workflow

The problem:

- Difficulty in comparing formalisms due to the diverse range of input languages and logical specialisations [1]
- Difficulty in comparing SMT provers for the same reasons
- 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

2. The need for Benchmarks

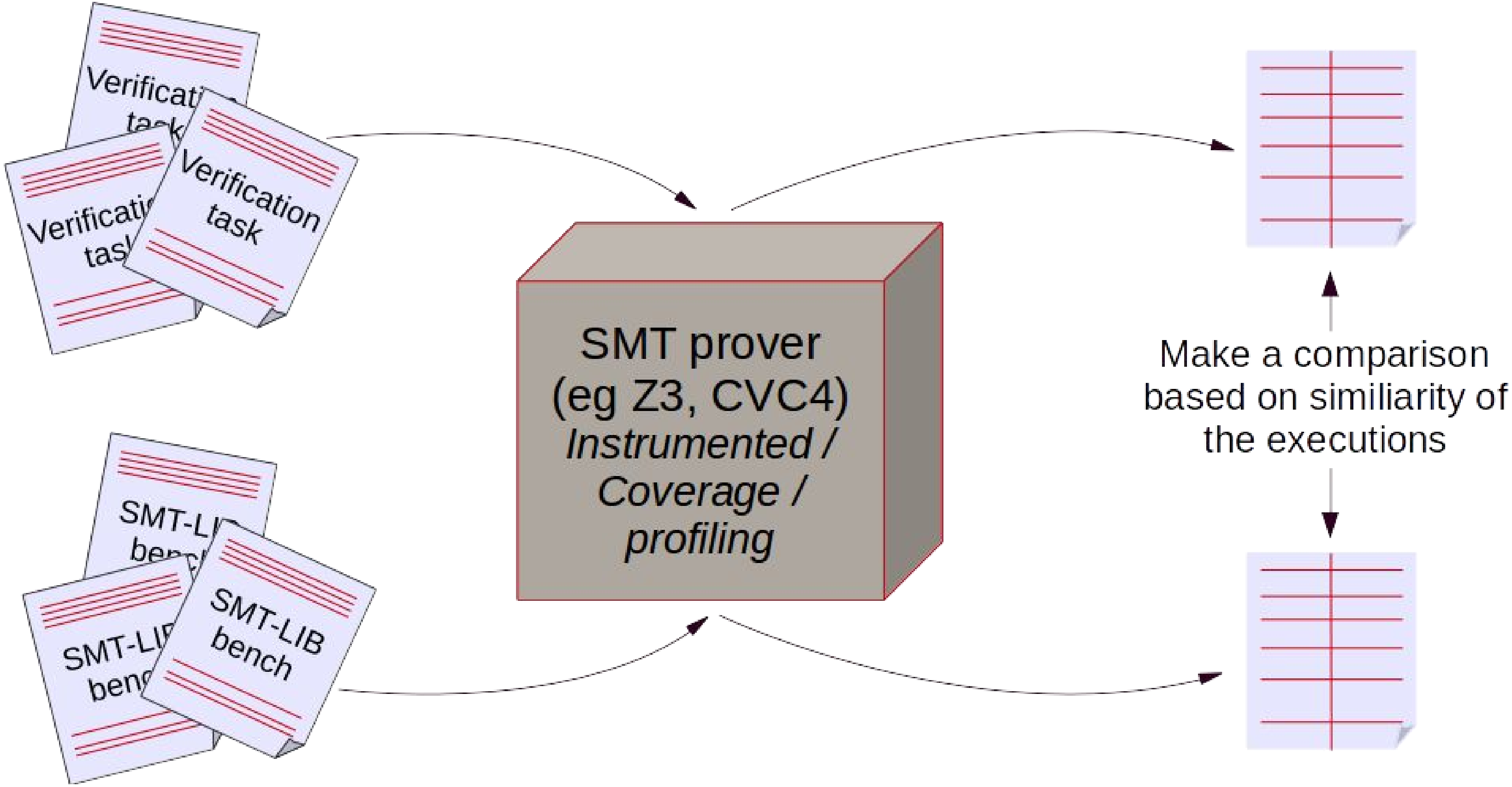
SMT and SMT-LIB

Satisfiability Modulo Theories - extend SAT solvers through the use of decision procedures;

eg. those for formulas containing Arrays, (Non)-Linear (Integer/Real) Arithmetic, Uninterpreted Functions, etc. and their combination.

Have been used to solve problems in scheduling, hardware verification, static analysis, ..., in addition to software verification.

The SMT-LIB project has developed a standardised input language, maintained a repository of over 100,000 benchmarks and organised an annual competition to evaluate SMT tools.



Can the comparative suitability of SMT solvers for software verification be derived from the use of the SMT-LIB benchmark repository?

3. Previous Experiments

Workflow:

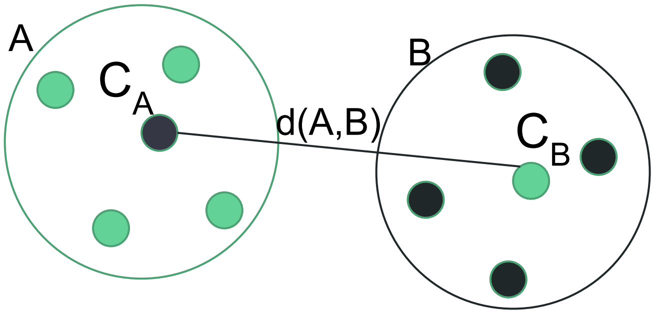
1. Profile programs from the Why3 examples and SMT-LIB repo with “callgrind”
2. Filter system calls to maintain platform independence
3. Perform dimensionality reduction via PCA
4. Assuming the Why3 programs are a representative “verification cluster”, measure its distance:
    a. To randomly-sampled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

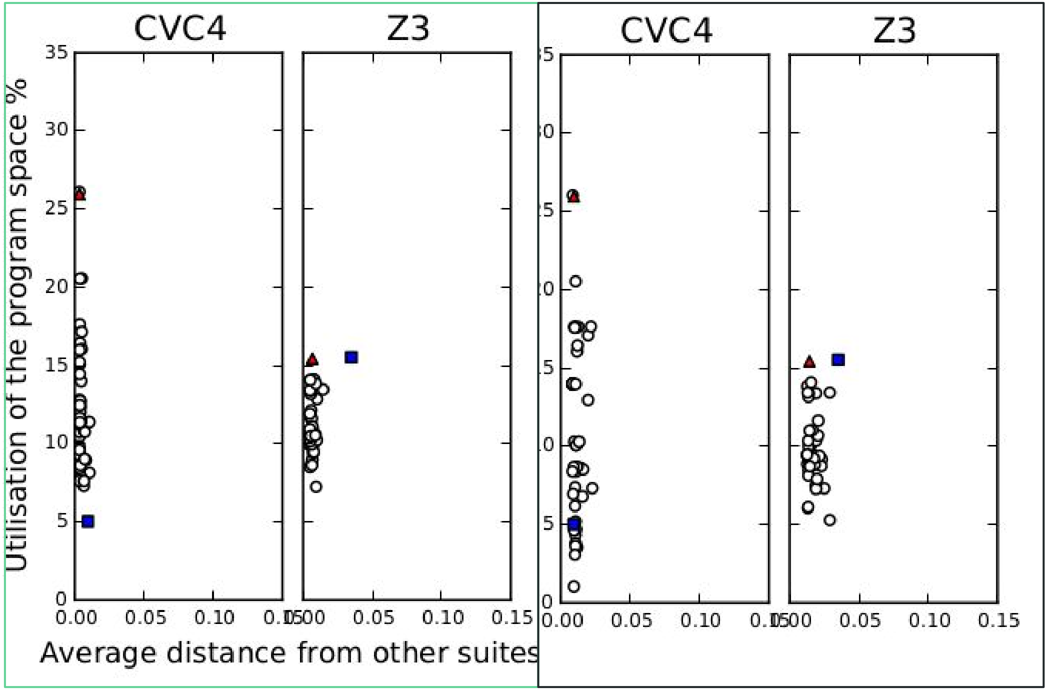
```
338,423,234  *  /build/buildd/eglibc-2.19/malloc/malloc.c:_int_malloc [/lib/x86_64-linux-gnu/libc-2.19.so]

      4,736  < /build/buildd/eglibc-2.19/malloc/malloc.c:_int_malloc (50x) [/lib/x86_64-linux-gnu/libc-2.19.so]
    316,224  < /build/buildd/eglibc-2.19/malloc/malloc.c:_int_realloc (4142x) [/lib/x86_64-linux-gnu/libc-2.19.so]
193,924,776  < /build/buildd/eglibc-2.19/malloc/malloc.c:free (2540202x) [/lib/x86_64-linux-gnu/libc-2.19.so]
190,082,586  *  /build/buildd/eglibc-2.19/malloc/malloc.c:_int_free [/lib/x86_64-linux-gnu/libc-2.19.so]

  6,837,856  < /build/buildd/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  < ???:newRaw8 (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::Noc
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::Noc
[/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/mdeters/cvc4/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



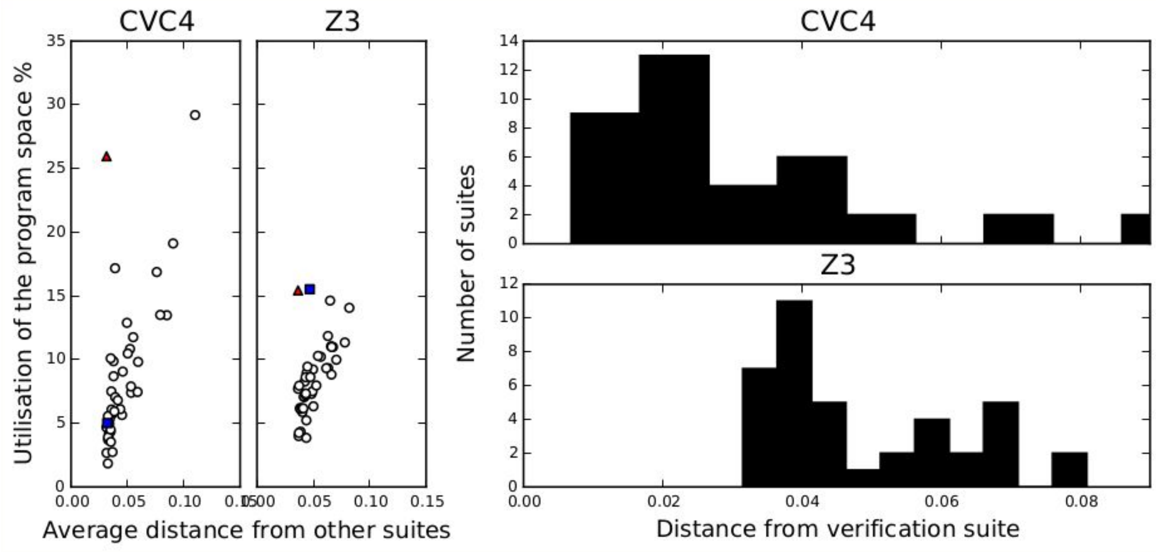


We used techniques and metrics described by Jozo Dujmovic in [2] to accurately characterise and describe machine-independent workloads.

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

What we learned

- Z3 uses more of its features during software verification tasks than CVC4 (experiment 1)
- The SMT-LIB 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 SMT-LIB repository than those that are very similar



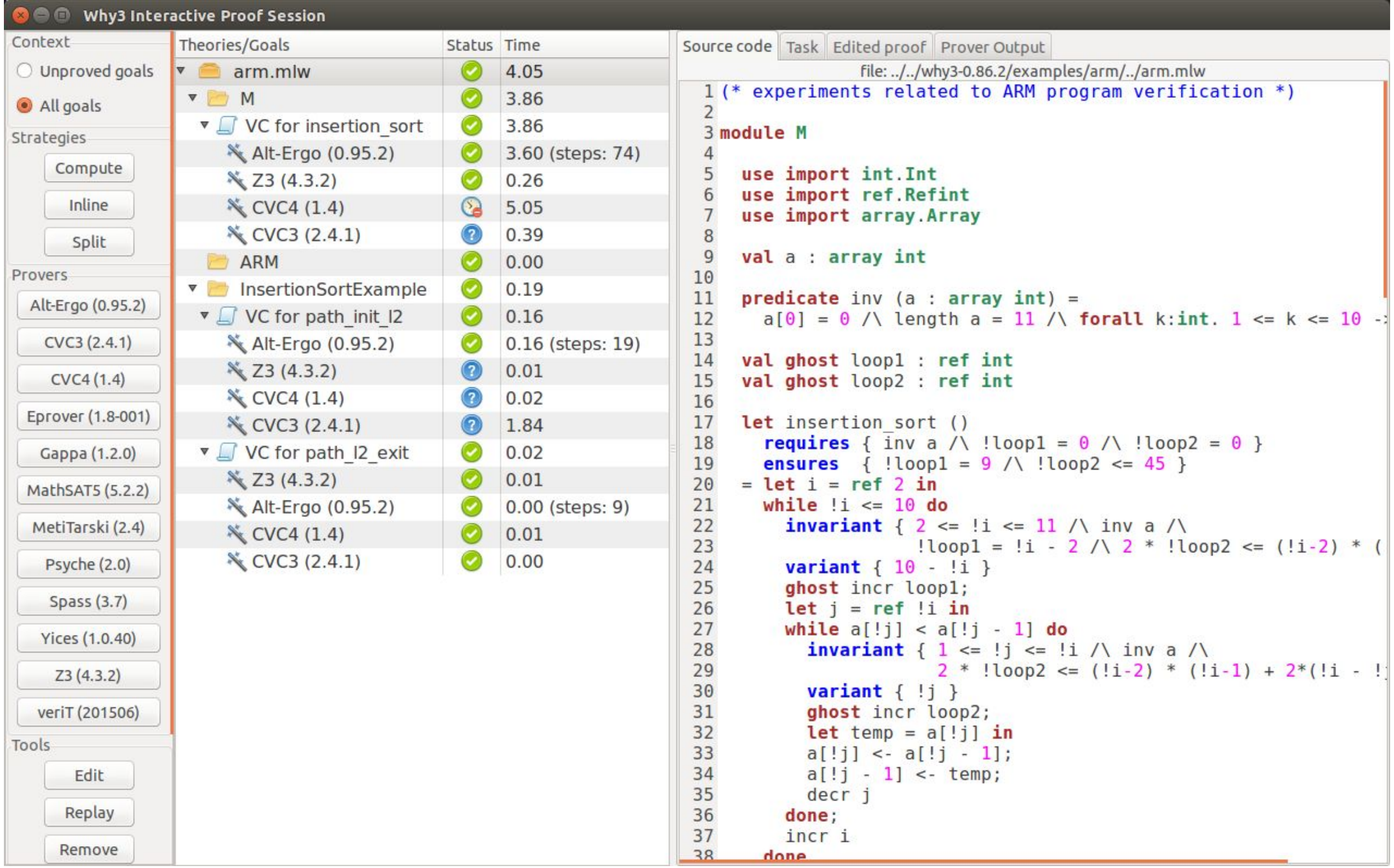
More details in [3]. Online data: **http://www.cs.nuim.ie/~ahealy/SAC2016**

Taking into account the result of the prover ...

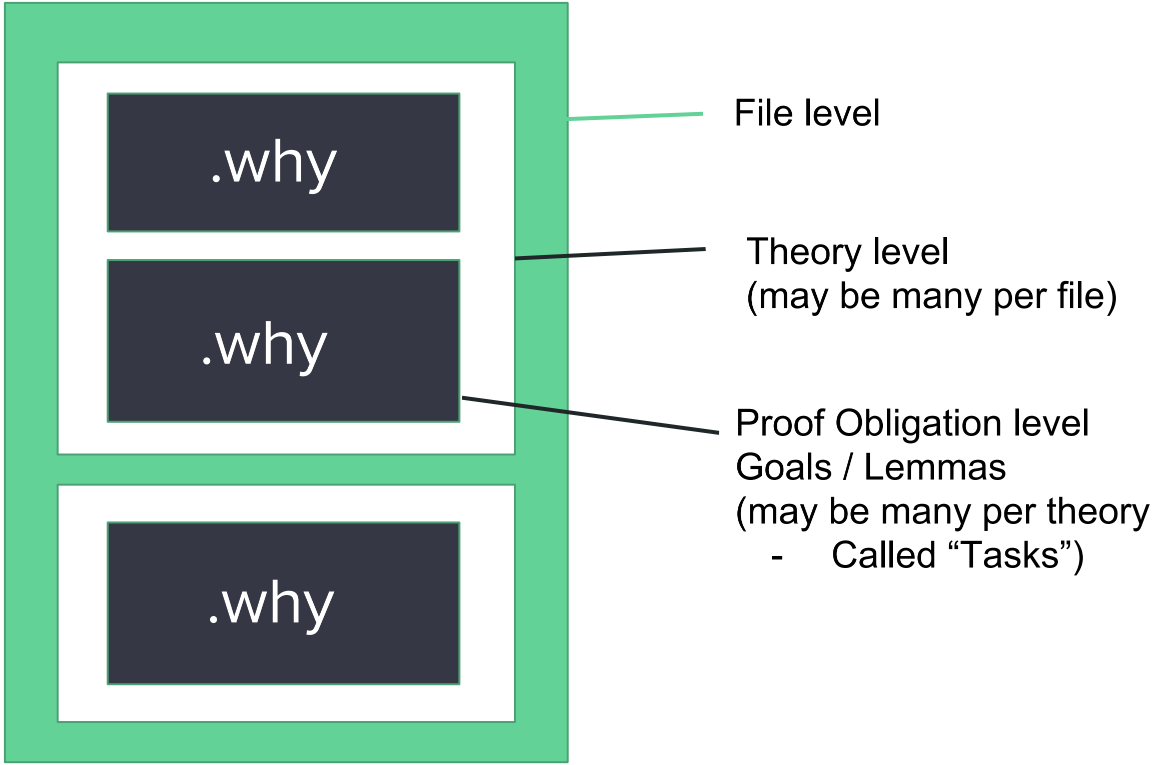
... and the time the prover took to return a result...

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

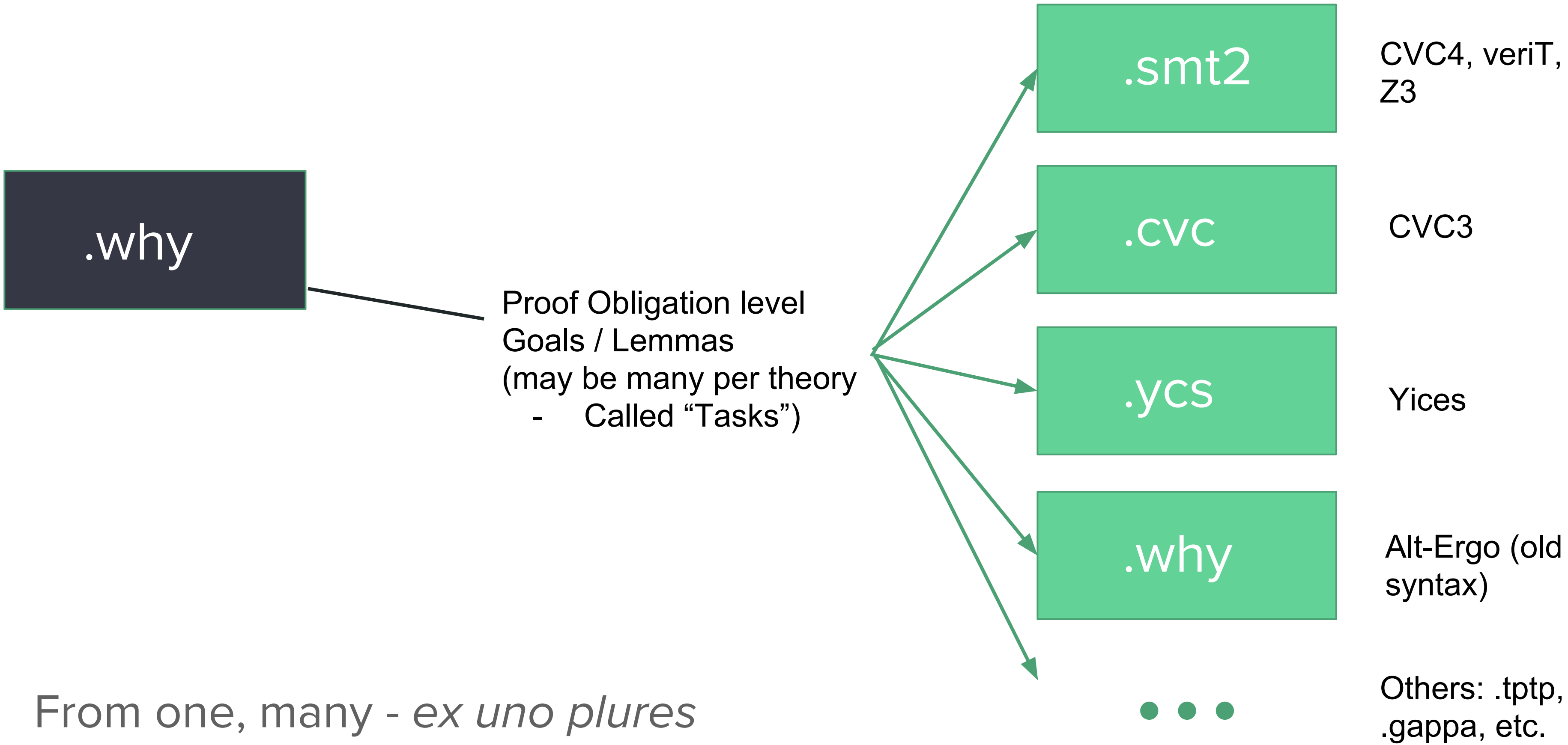
4. Why3 demonstration



What makes one prover better than another?



Anatomy of a Why3ML file



From one, many - *ex uno plures*

5. 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 [4]
- **ML4PG** (Machine Learning for Proof General) Interfaces Coq proofs developed in Proof General to MATLAB/Weka clustering algorithms, again guiding interactive proofs [5]
- **Verifolio** extracts static metrics from C programs to choose the most likely ATP to use (via Support Vector Machines) [6]

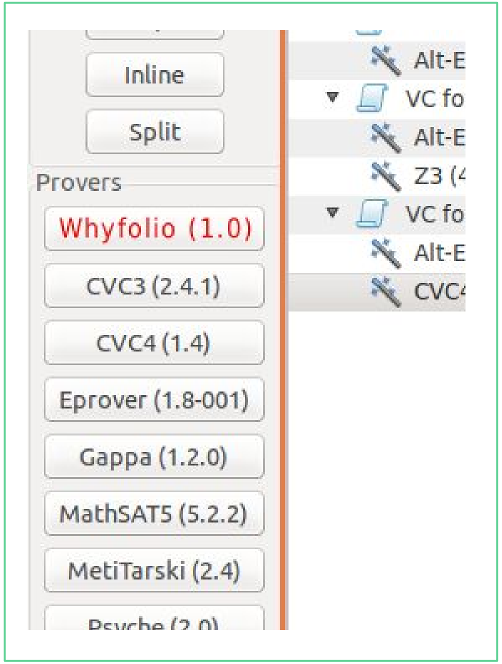
A Portfolio Solver for Why3

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 [6] 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

In a container

docker

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**.