Characterising the Workload of SMT Solvers for Program Verification

Andrew Healy
ahealy@cs.nuim.ie

Rosemary Monahan
rosemary@cs.nuim.ie

James F Power
jpower@cs.nuim.ie

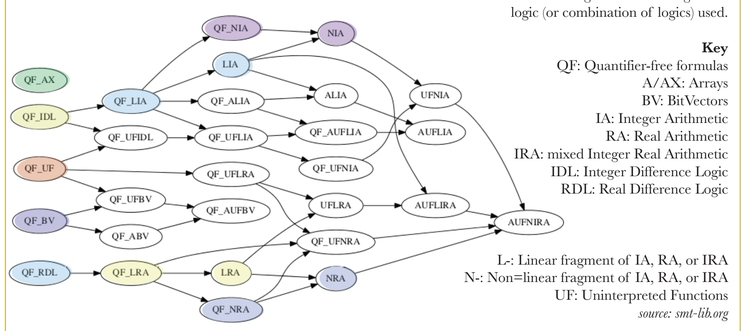
Introduction & Motivation

SMT (Satisfiability Modulo Theories) solvers are often used as back-end theorem provers for deductive software verification systems such as Spec#, KeY, Dafny etc. Such systems are often composed of (1) a front-end specification language which is used to annotate code with pre- and post-conditions, invariants and assertions, (2) a verification condition-generating stage (which may use an intermediary language such as Boogie or Why3) and (3) an SMT solver which returns the satisifiability of the constraints.

The SMT-LIB project has been successful in producing a standardised language for SMT solvers as well as a large repository of benchmark problems to aid comparitive evaluation of tools. Our project aims to make use of these standarisations in order to investigate to what extent the workload of verification problems differs from that of generic SMT benchmarks (from domains such as scheduling, micro-processor verification and model-checking).

We hope to learn whether standard SMT benchmarks can be used to measure the suitability of a variety of SMT solvers for verification tasks.

Fig. 1 The logics of SMT-LIB



Benchmarks are organised according to the logic (or combination of logics) used.

Key
QF: Quantifier-free formulas
A/AX: Arrays
BV: BitVectors
IA: Integer Arithmetic
RA: Real Arithmetic
IRA: mixed Integer Real Arithmetic
IDL: Integer Difference Logic
RDL: Real Difference Logic

L-: Linear fragment of IA, RA, or IRA
N-: Non=linear fragment of IA, RA, or IRA
UF: Uninterpreted Functions
source: smt-lib.org

Approach

Selection of SMT Solvers:
Initially, two leading SMT solvers were chosen, Z3 and CVC4, due to their similarity in terms of implementation language (C++) and SMT-LIB compliance. Both can recognise a large selection of theories, perform well in competition and are used widely in verification systems.

Selection of benchmarks:
As no standardised benchmark suite for program verification exists, we used a selection of example programs distributed with the Why3 verification platform. These were chosen for two reasons: the loarge selection of programs available and the use of solver-specific drivers to translate verification conditions to the SMT-LIB format. A random selection of benchmarks from the atomic logics of the SMT-LIB repository (Fig. 1) were selected for use (colouring shows how the QF extensions were grouped together).

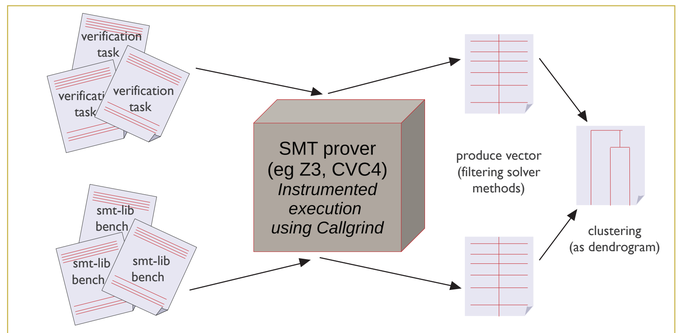


Fig. 2 High level workflow

Profiling and filtering:
We used the Callgrind profiling tool (part of the Valgrind suite) to measure the proportional number of assembly instructions called by each method of the SMT solver for each benchmark individually, averaging these results for a suite of benchmarks. A large number of methods (i.e. 2-3000) were returned: calls to system libraries were filtered out to keep the focus on the workload of the SMT solver itself.

Preliminary Results

To analyse the large amount of data collected, we used dendrogram representations of hierarchical clustering.

Fig. 3: Similarity of verification workloads to those of SMT-LIB benchmarks (Z3)

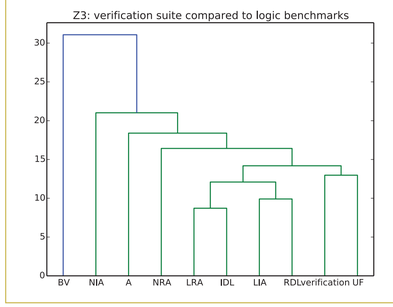


Figure 3 shows that benchmarks focussing on Uninterpreted Functions (UF) have the closest workload characteristics to verification tasks performed by Z3, while Bit Vector benchmarks display very different characteristics to software verification.

Fig. 4: Similarity of verification workloads to those of SMT-LIB benchmarks (CVC4)

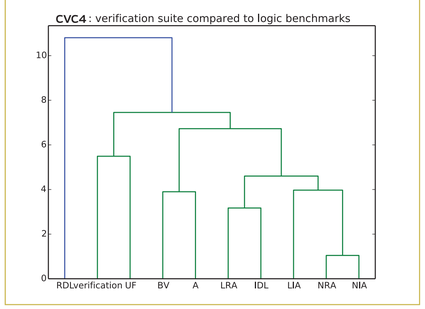


Figure 4 shows that the UF workload is closest to verification for the CVC4 tool also. The smaller scale on the y-axis indicates that the workloads observed were more similar in character than the corresponding executions using Z3.

Future work:
It is hoped that we will continue this work by looking at various groupings of benchmarks from the 100,000 (approx.) SMT files in the repository. Experiments will also be carried out using different metrics to measure similarity/difference of workloads.

Department of Computer Science, Callan Building,
Maynooth University, Maynooth, Co. Kildare, Ireland

This project is being carried out with funding provided by Science Foundation Ireland under grant number 11/RFP.1/CMS/3068