Andrew Healy
Principles of Programming Group
Supervisors: Dr Rosemary Monahan & Dr James Power

An empirical study into how modularisation can be applied to program verification

CS Research Postgraduate Workshop 2015

Maynooth University
National University of Ireland Maynooth

Outline

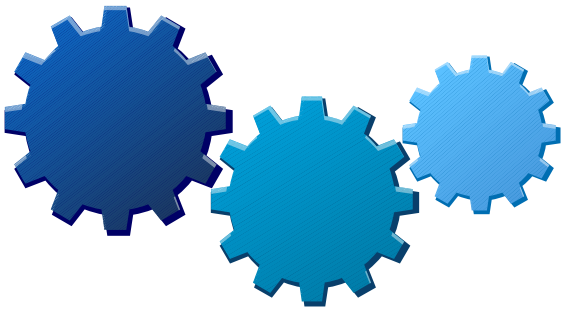
- Brief Personal Introduction
- Modularisation in Software Development
- Overview of Software Verification
- Modularisation and Related Issues I have looked at this year
- Current / Future Work

Personal Introduction

- Background in visual communication and media art
- Started using Processing about 5 years ago
- Completed HDip in IT at NUIM in 2013-14

Modularisation in Software Development

- Helps break complex problems into managable parts
- Abstraction aids understanding by avoiding unnecessary details
- Gives logical structure, promotes collaborative working, easier to debug, etc.

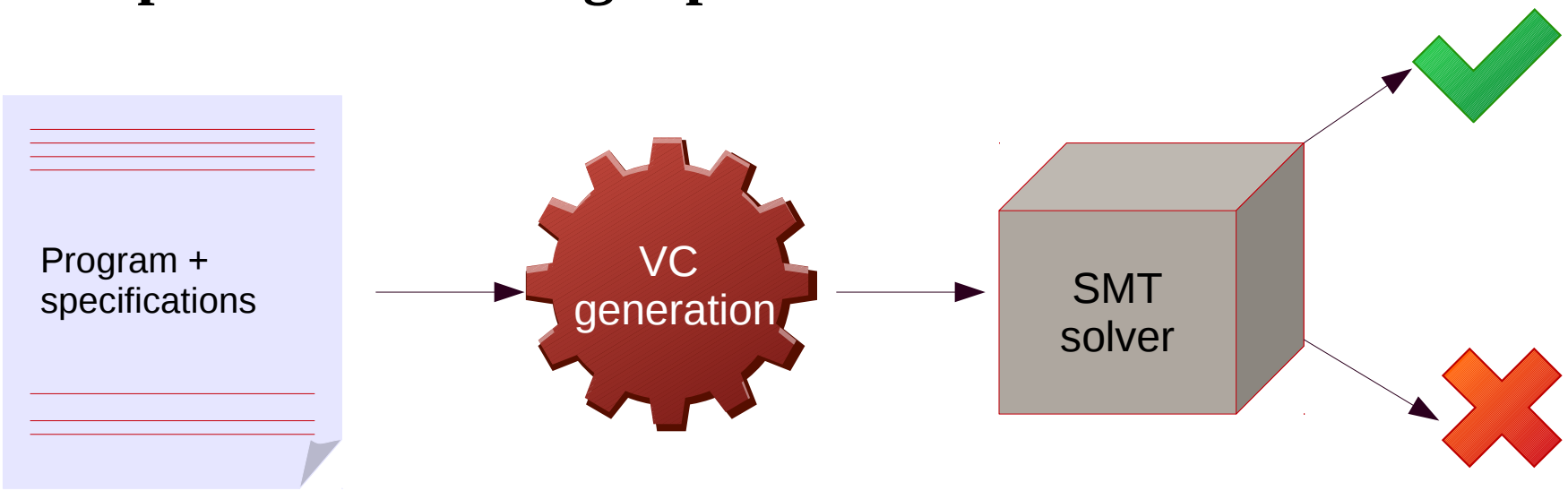


An overview of the Verification Process

- Proving the correctness of a program with respect to its formal specification
- In a deductive verification scenario, this typically involves a number of stages and transformations:
  1. Manually adding specifications to the program
  2. Using a tool to generate verification conditions
  3. Running the conditions through a theorem prover to produce a result

Modularisation in the Verification Process

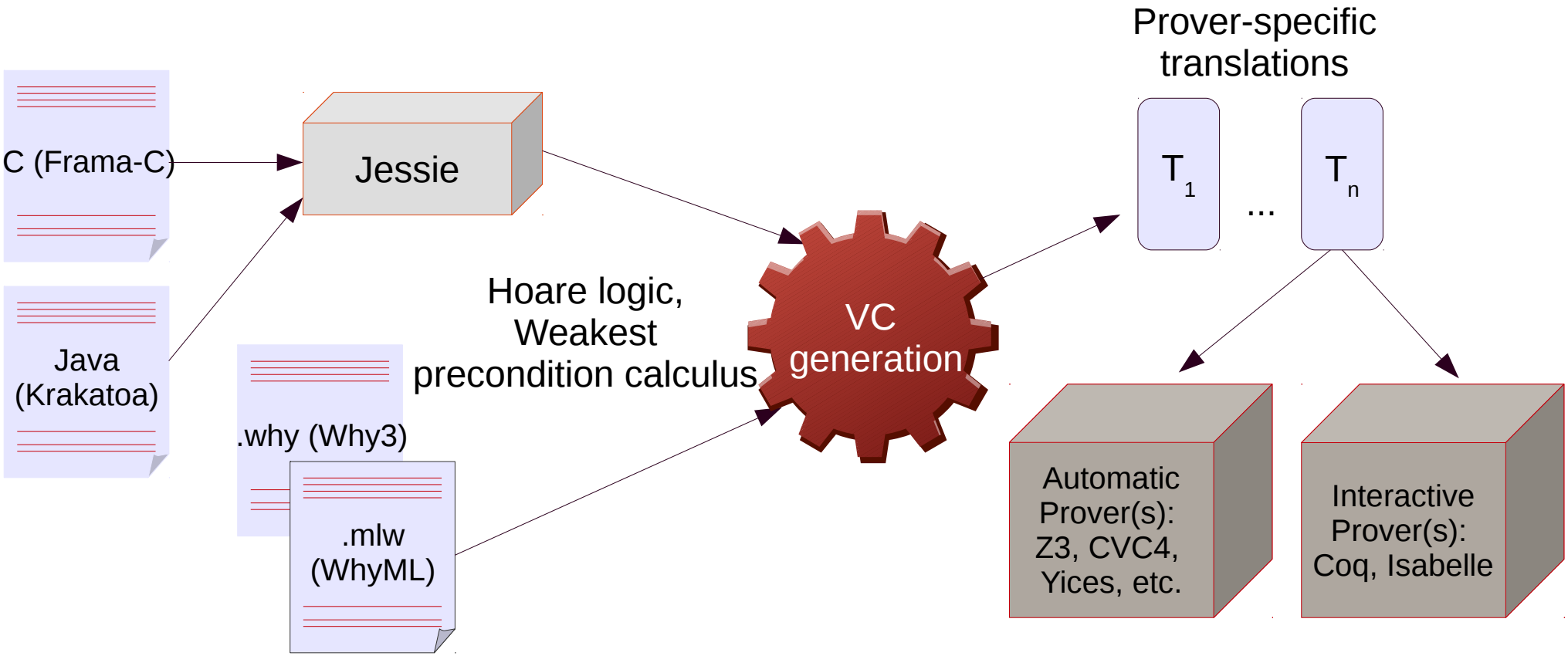
Each of these verification stages can be seen as a modular component in the larger process



Being able to reason about these components modularly brings efficiency and abstraction

Example system: The Why3 Platform

Differs from most other systems by being highly extensible (due to its modular architecture)



Introducing SMT solvers

- SMT (*Satisfiability Modulo Theories*) solvers are complex programs for determining whether a set of contraints is satisfiable or not
- Each logical theory (eg. bitvectors, uninterpreted functions, linear arithmetic) must have a decision procedure in order to be useful in this context
- Many SMT solvers extend SAT by using the Nelson-Oppen method to combine decision procedures
- Are used in problem domains such as scheduling, modelling, static analysis and software verification

SMT-LIB

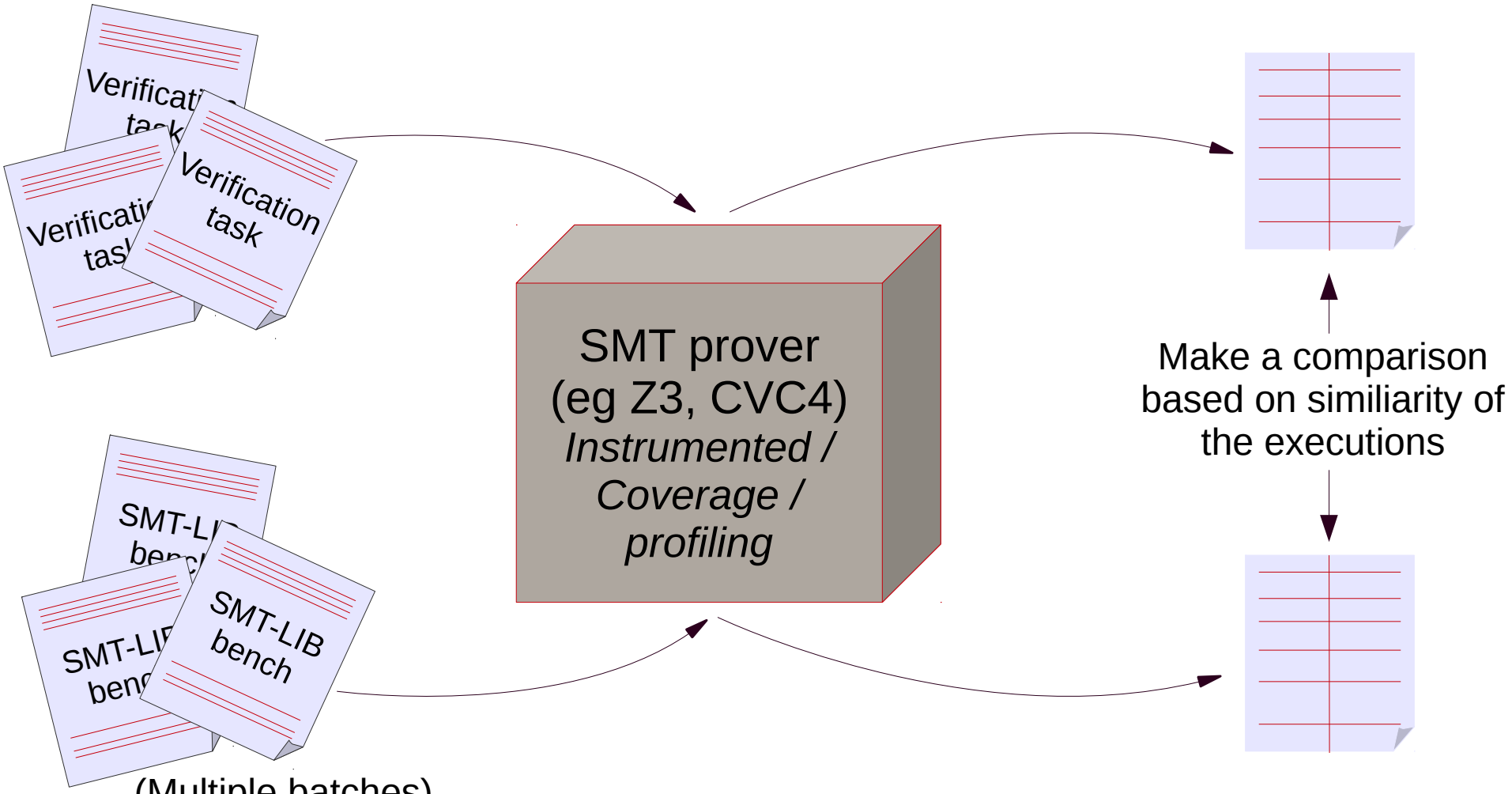
- A set of over 90,000 benchmarks for tool evaluation
- A standardised input language for SMT solvers

```
> (set-logic QF_LIA)
> (declare-fun x () Int)
> (declare fun y () Int)
> (assert (= (+ x (* 2 y)) 20))
> (assert (= (- x y) 2))
> (check-sat)
sat
> (get-value (x y))
((x 8)(y 6))
```

Current work

- Question: Can we evaluate the suitability of SMT solvers for software verification tasks by using generic SMT-LIB benchmarks?
- Making use of standard input language: one of the barriers to making a comparitive evaluation of verification systems
- Another question: What can we learn about the execution of SMT solvers by doing this comparison?
- Making use of profiling and code coverage tools to characterise the workload of SMT solvers

A proposed workflow



References

- Bobot, F., Filliâtre, J-C., Marché, C., Melquiond, Paskevich, A: **Why3: Shepard Your Herd of Provers**. In *Boogie 2011: First International Workshop on Intermediate Theorem Provers*, pp. 53-64, Wroclaw, Poland, August 2011
- Hoare, T., Misra, J., Leavens, G., Shankar, N., **The Verified Sofware Initiative: A Manifesto**, In *ACM Computing Surveys; 41*, *4* New York, USA, October 2009
- Cok, D. **The SMT-LIB v2 Language and Tools: A Tutorial** (version 1.2.1 – March 18, 2013) available at http://www.grammatech.com/resource/smt/SMTLIBTutorial.pdf
- Monahan, R. **CS605: Rigorous Software Process: Module Overview** (lecture slides) Maynooth University, October 2014

Thanks!

Questions / Comments...