Verification-Aided Regression Testing

Fabrizio Pastore[1] Leonardo Mariani[1]
Antti E. J. Hyvärinen[2] Grigory Fedyukovich[2]
Natasha Sharygina[2] Ondrej Sery[2] Stephan Sehestedt[3]
Ali Muhammed[4]

[1]University of Milano-Bicocca, Italy

[2]University of Lugano, Switzerland

[3]ABB Corporate Research, Ladenburg, Germany

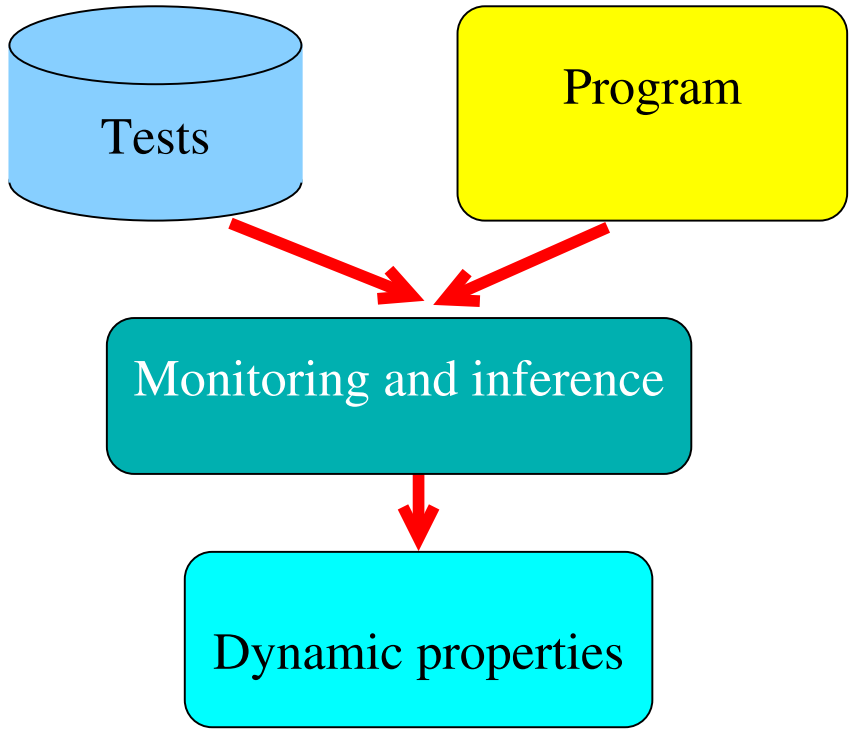
[4]VTT Technical Research Centre, Tampere, Finland

October 17, 2013

Motivation

- Regression testing is an integral part of many software development processes
  - Given an *upgrade* of a software, does it satisfy a *validation test suite* passed by the *base* version of the software
- The detection of faults depends critically on the quality of the validation test suite
- This work aims at reducing the dependency on the test suite by
  - (i) automatically producing properties that hold for the base version
  - (ii) automatically identifying and checking on the upgraded program only the properties that the developer intends the upgrade to preserve
  - (iii) Reporting faults not revealed by the regression tests
- We use dynamic property generation together with bounded model checking to achieve the goal.

Regression Testing & Dynamic Property Detection

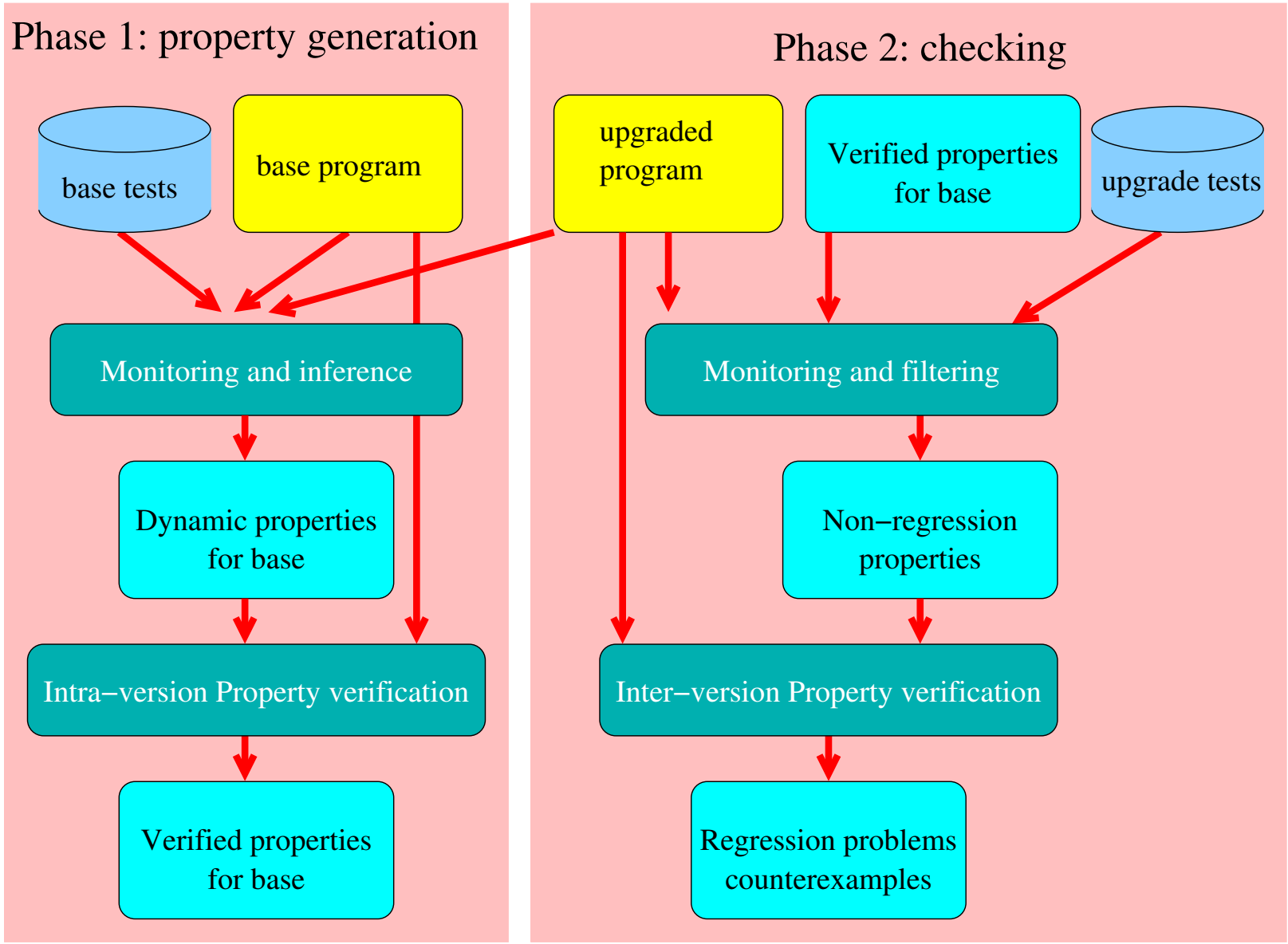


- The main purpose of regression testing is to validate that an already tested code has not been broken by an upgrade
- Property detection aims at identifying "likely invariants" by observing the program behavior on the validation suite
- This work deals with properties expressed as *assertions*

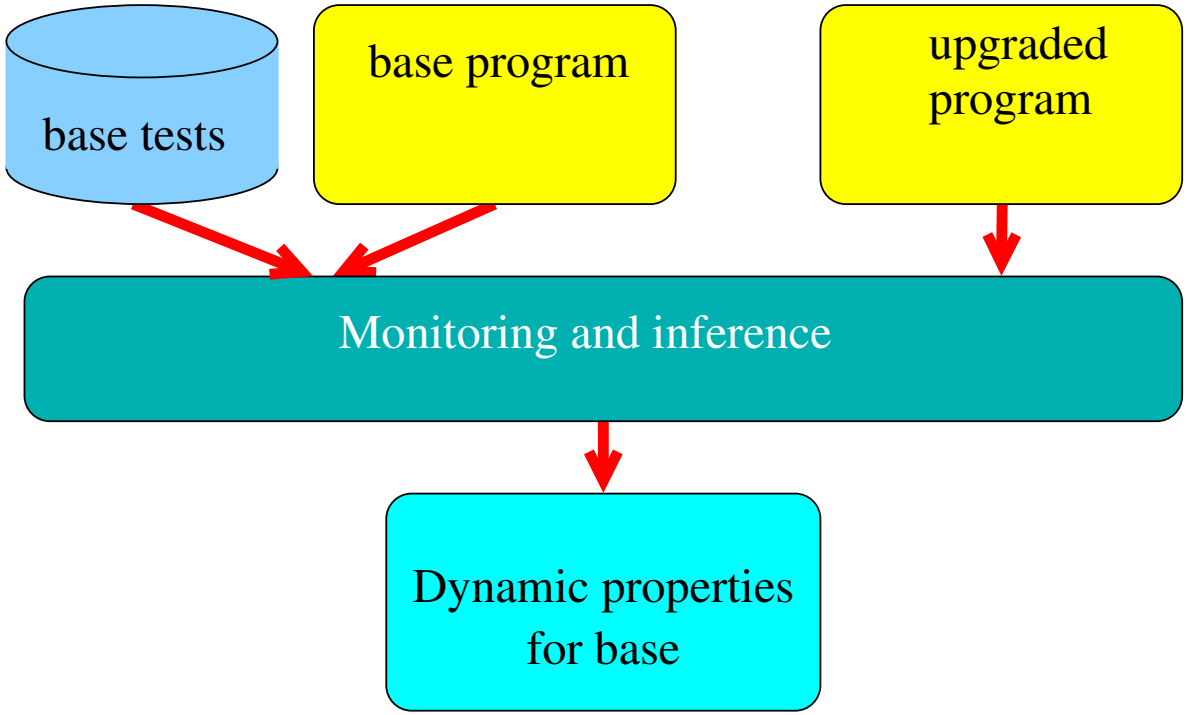
Bounded Model Checking

- Given the C source code of a program P, we generate Boolean representation ϕ_P of an *unwound* version of the program
  - Each loop is inlined up to a fixed bound k
  - Each function call is inlined
  - The inlined version is converted to a bit-precise representation as an instance of the propositional satisfiability problem
  - Heap operations and reference arguments are mostly ignored
- Any assertion a in the source code is converted into a Boolean formula ϕ_a, negated, and conjoined with the program, resulting in $\phi_P \wedge \neg\phi_a$
- The satisfying truth assignments of $\phi_P \wedge \neg\phi_a$ correspond to the executions of P which repeat each loop at most k times and violate the assertion a

Verification-Aided Regression Testing (VART)



VART Phase 1: monitoring and inference

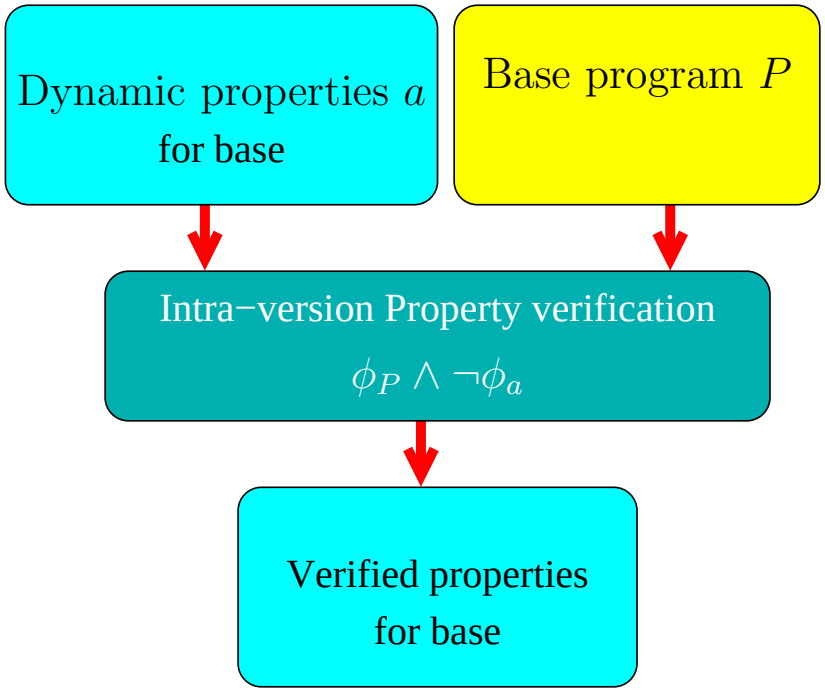


- Generates a large number of dynamic properties
- Based on observing the base program behavior in the regression test suite
- To limit the number of generated properties, only locations "likely affected by the change" are monitored
- Uses the Daikon invariant generator

VART Phase 1: Detecting Dynamic Properties

- Dynamic properties are collected by monitoring the base version while it executes its regression test suite
- To keep number of generated assertions sustainable, the property generation is localized to places affected by the change
- The modified functions are identified, and monitoring is done on unchanged statements in functions
  - that contain changes
  - that call functions that contain changes; and
  - that are called by the functions that contain changes.

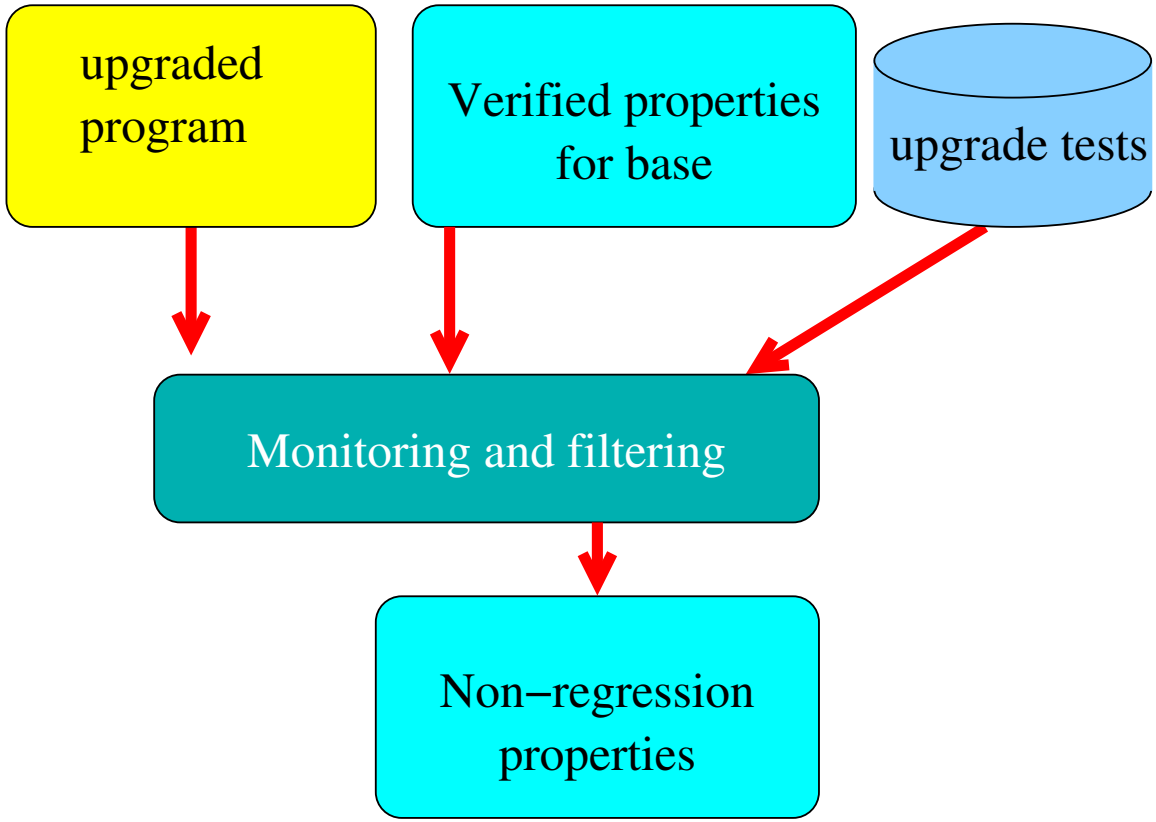
VART Phase 1: Generating Verified Properties



- Dynamic properties often overfit the regression test, resulting in large number of false positives
- We reduce the number of false positives with BMC, passing forward only *true* assertions a (for which the SAT check $\phi_P \wedge \neg\phi_a$ returns *unsatisfiable*).

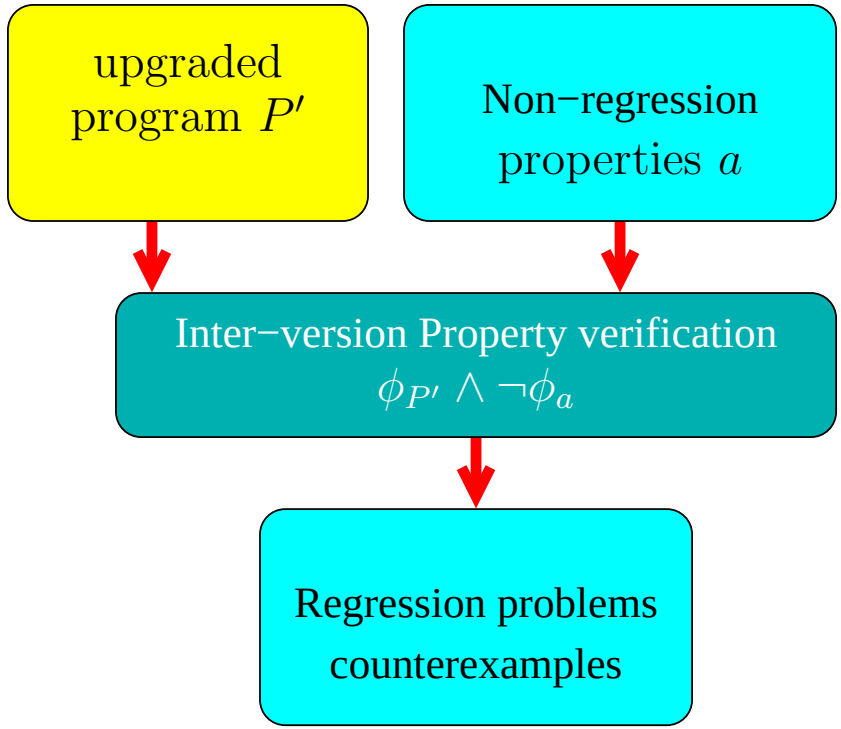
- The scope of BMC is limited to the call trees rooted at the callers of the function containing the changes
- Rest of the program treated non-deterministically

VART Phase 2: Filtering Verified Properties



- Some properties that hold for the previous version might be intentionally broken by the developer
- The regression test suite for the upgrade is used to filter out such verified but outdated properties

VART Phase 2: Upgrade Checking



- Finally, the non-regression properties are checked against the upgrade P' using BMC
- Properties reported as *false* or *unreachable* indicate the presence of faults

Implementation

- VART is implemented for C programs
- Generation of dynamic properties is implemented on top of the Radar tool [PMG13] using GDB and Daikon [ECGN01]
- Model checking with eVolCheck [FSS13]
- Support also for CBMC

[PMG13] F. Pastore, L. Mariani, and A. Goffi. *RADAR a tool for debugging regression problems in C/C++ Software*. ICSE Tool Demo Track, 2013.
[ECGN01] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. *Dynamically discovering likely program invariants to support program evolution*. IEEE Transactions on Software Engineering, 27(2): 99-123, 2001.
[FSS13] G. Fedyukovich, O. Sery, and N. Sharygina: *eVolCheck: Incremental Upgrade Checker for C*. TACAS 2013.

Empirical Evaluation: Insufficient test suite

- We test VART in detecting faults in the implementation of the Grep utility
- Different degrees of coverage using Grep regression test suite
- Faults are injected from the SIR repository (total 11)

	Revealed Faults			
Test suite	Testing	VART	TP	FP
Cov20	3	5	5	0
Cov50	7	8	2	0
MRT	10	10	0	0

- Cov20 — 20 % coverage, Cov50 — 50 % coverage MRT — smallest subset of tests that gives the same coverage as full test suite
- TP — true positives, FP – false positives

Empirical evaluation: case studies

App.	Subject Size (LOCS)	Test suite Size	Dyn. Prop	Non-Reg Prop	TP	FP
VTT	488	1000	1045	658	15	0
Sort	4653	427	356	2	1	0
Grep	590	817	3303	51	3	0

- VTT is a motion trajectory control system executed by a robotic arm designed to perform maintenance tasks in the Iter fusion reactor
  - Regression test consist of random inputs as 12 numbers
- Grep and Sort are the GNU coreutil tools with their respective test suites
  - Faults inserted from mailing lists and SIR
  - Identified faults are not revealed by the available test suites

Conclusions

- Regression testing is widely used, but compelling test suites are difficult to design
- VART can detect faults that are undetected by the test suites by
  - Automatically producing properties from the base version test suite
  - filtering out the properties intentionally broken by the upgrade
  - reporting faults and counterexamples not revealed by tests
- Empirical evaluation shows that VART complements and increases the effectiveness of regression testing