The PeRIPLO Propositional Interpolator

N. Sharygina

Formal Verification and Security Group

University of Lugano

joint work with Leo Alt, Antti Hyvarinen, Grisha Fedyukovich and Simone Rollini

October 2, 2015

Outline

Outline

Outline

1. Formal Verification at USI, Lugano
2. Interpolation-based Model Checking
3. Flexible Interpolation Framework

Outline

1. Formal Verification at USI, Lugano
2. Interpolation-based Model Checking
3. Flexible Interpolation Framework

Background

Formal Verification in Lugano, Switzerland

- Program Verification

Background

Formal Verification in Lugano, Switzerland

- Program Verification
    - Model checking software (FunFrog, EvolCheck, LoopFrog), ANSI-C programs
    - Efficient decision procedures as computational engines of verification (OpenSMT)

- Program Verification
  - Model checking software (FunFrog, EvolCheck, LoopFrog), ANSI-C programs
  - Efficient decision procedures as computational engines of verification (OpenSMT)
- Abstractions

Background

Formal Verification in Lugano, Switzerland

- Program Verification
  - Model checking software (FunFrog, EvolCheck, LoopFrog), ANSI-C programs
  - Efficient decision procedures as computational engines of verification (OpenSMT)
- Abstractions
  - Interpolation-based Bounded Model Checking
    - Function summarization [ATVA'12]
    - Upgrade checking, Incremental verification [FMCAD'13], [TACAS'13]
    - Recursion depth detection [STTT'15]
    - Verification-aided regression testing [ISSTA'13]

- Abstractions
  - Leveraging Interpolant strength [CAV'12]
  - Loop Summarization [ATVA'08], [ASE'09]
    - Program Termination [CAV'10], [TACAS'11]

- Abstractions
  - Leveraging Interpolant strength [CAV'12]
  - Loop Summarization [ATVA'08], [ASE'09]
    - Program Termination [CAV'10], [TACAS'11]
  - Synergy of Abstractions [STTT'10]

Background

Formal Verification in Lugano, Switzerland

- An SMT-based verification framework for software systems handling arrays [FMSD'15]
  - A quantifier-free interpolation procedure extending Lazy Abstraction [McMillan'06] to a quantified level [LPAR'12]
  - Identification of a class of relations over arrays admitting definable first-order acceleration [TACAS'13]
  - Booster: An Acceleration-Based Verification Framework for Array Programs [ATVA'14]

- Boolean and Theory Reasoning (SAT/SMT)

Background

Formal Verification in Lugano, Switzerland

- Boolean and Theory Reasoning (SAT/SMT)
  - Proof reduction and proof manipulation for interpolation [FMSD'15]
  - Proof Sensitive Interpolation [VSTTE'15]
  - Search-Space Partitioning for Parallelizing SMT Solvers [SAT'15]
  - Procedure for bit-vector extraction and concatenation [ICCAD'09]
  - Generation of explanations in theory propagation [MEMOCODE'10]

- Boolean and Theory Reasoning (SAT/SMT)
    - Solver, *OpenSMT*, combines MiniSAT2 SAT-Solver with state-of-the-art decision procedures for QF EUF, LRA, BV, RDL, IDL
        - *Extensible*: the SAT-to-theory interface facilites design and plug-in of new decision procedures
        - *Incremental*: suitable for incremental verification
        - *Open-source*: available under MIT license
        - *Parallelized*: efficient search space partitioning
        - *Efficient*: competitive open-source SMT Solver for QF UF, IDL, RDL, LRA according to SMT-Comp.

Outline

Interpolation

Background

- WIde application in symbolic model checking

Interpolation
Background

- WIde application in symbolic model checking
  - Bounded model checking: approximate cheaper reachability set computation [McMillan03]

Interpolation

Background

- WIde application in symbolic model checking
  - Bounded model checking: approximate cheaper reachability set computation [McMillan03]
  - Predicate abstraction refinement based on spurious behaviors [Henzinger04]

Interpolation

Background

- WIde application in symbolic model checking
  - Bounded model checking: approximate cheaper reachability set computation [McMillan03]
  - Predicate abstraction refinement based on spurious behaviors [Henzinger04]
  - Property-based transition relation approximation [Jhala05]
  - ...

Interpolation

Background

- WIde application in symbolic model checking
    - Bounded model checking: approximate cheaper reachability set computation [McMillan03]
    - Predicate abstraction refinement based on spurious behaviors [Henzinger04]
    - Property-based transition relation approximation [Jhala05]
    - ...
- Forementioned applications involve

Interpolation
Background

- WIde application in symbolic model checking
  - Bounded model checking: approximate cheaper reachability set computation [McMillan03]
  - Predicate abstraction refinement based on spurious behaviors [Henzinger04]
  - Property-based transition relation approximation [Jhala05]
  - ...
- Forementioned applications involve
  - Problem encoding into logic (SAT, SMT)

Interpolation
Background

- WIde application in symbolic model checking
  - Bounded model checking: approximate cheaper reachability set computation [McMillan03]
  - Predicate abstraction refinement based on spurious behaviors [Henzinger04]
  - Property-based transition relation approximation [Jhala05]
  - ...
- Forementioned applications involve
  - Problem encoding into logic (SAT, SMT)
  - Problem solving by means of resolution based engines (SAT solvers, SMT solvers)

Notation

Interpolation

- Craig's interpolant I for unsatisfiable conjunction of formulae $A \wedge B$ [Craig57]

Notation

Interpolation

- Craig's interpolant I for unsatisfiable conjunction of formulae $A \wedge B$ [Craig57]
  - $A \Rightarrow I$, $I \wedge B$ unsatisfiable

Notation

Interpolation

- Craig's interpolant I for unsatisfiable conjunction of formulae $A \wedge B$ [Craig57]
  - $A \Rightarrow I$, $I \wedge B$ unsatisfiable
  - I defined over common symbols of A and B

Notation
Interpolation

- Craig's interpolant I for unsatisfiable conjunction of formulae $A \wedge B$ [Craig57]
  - $A \Rightarrow I$, $I \wedge B$ unsatisfiable
  - I defined over common symbols of A and B
  - I as over-approximation A conflicting with B

Notation
Interpolation

- Craig's interpolant I for unsatisfiable conjunction of formulae $A \wedge B$ [Craig57]
  - $A \Rightarrow I$, $I \wedge B$ unsatisfiable
  - I defined over common symbols of A and B
  - I as over-approximation A conflicting with B
- Example

Notation

Interpolation

- Craig's interpolant I for unsatisfiable conjunction of formulae $A \wedge B$ [Craig57]
  - $A \Rightarrow I$, $I \wedge B$ unsatisfiable
  - I defined over common symbols of A and B
  - I as over-approximation A conflicting with B

- Example
  - $A \triangleq (\overline{p} \vee \overline{q}) \wedge (p \vee \overline{q})$ $\qquad B \triangleq (q \vee \overline{r}) \wedge (q \vee r)$

Notation

Interpolation

- Craig's interpolant I for unsatisfiable conjunction of formulae $A \wedge B$ [Craig57]
  - $A \Rightarrow I$, $I \wedge B$ unsatisfiable
  - I defined over common symbols of A and B
  - I as over-approximation A conflicting with B

- Example
  - $A \triangleq (\overline{p} \vee \overline{q}) \wedge (p \vee \overline{q})$ $\quad B \triangleq (q \vee \overline{r}) \wedge (q \vee r)$
  - Interpolant $\overline{q}$

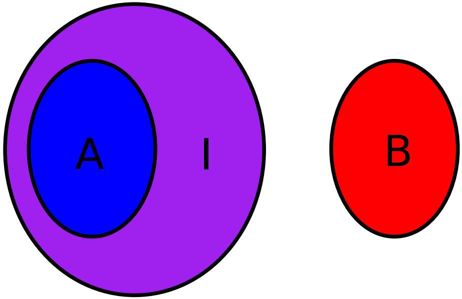
Notation

Interpolation

- Craig's interpolant I for unsatisfiable conjunction of formulae $A \wedge B$ [Craig57]
  - $A \Rightarrow I$, $I \wedge B$ unsatisfiable
  - I defined over common symbols of A and B
  - I as over-approximation A conflicting with B

- Example
  - $A \triangleq (\overline{p} \vee \overline{q}) \wedge (p \vee \overline{q})$ $\qquad B \triangleq (q \vee \overline{r}) \wedge (q \vee r)$
  - Interpolant $\overline{q}$
  - $A \Rightarrow \overline{q}$ $\qquad \overline{q} \wedge B$ unsatisfiable

- Craig's interpolant I for unsatisfiable conjunction of formulae $A \wedge B$ [Craig57]
  - I as over-approximation A conflicting with B



- Problems
  - Size affects efficiency
  - Interpolants different in their logical strength are needed
  - Collection of individual algorithms, no possibilities for adjustments wrt the model checking tasks

Outline

Interpolation-based Model Checking

Motivation

- PeRIPLO is a multi-purpose interpolation framework

- PeRIPLO is a multi-purpose interpolation framework
  - aims at producing interpolants that are suitable for the whole spectrum of interpolation applications
    - emphasis on constructing *small* interpolants
    - flexibility in *strength*

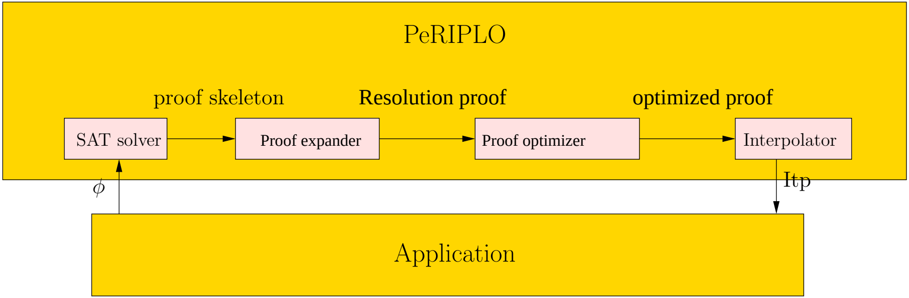
- PeRIPLO is a multi-purpose interpolation framework
  - aims at producing interpolants that are suitable for the whole spectrum of interpolation applications
    - emphasis on constructing *small* interpolants
    - flexibility in *strength*
  - Pre-processing approaches
    - proof reduction and compression
    - proof manipulation for interpolation

Interpolation-based Model Checking

Motivation

- PeRIPLO is a multi-purpose interpolation framework
  - aims at producing interpolants that are suitable for the whole spectrum of interpolation applications
    - emphasis on constructing *small* interpolants
    - flexibility in *strength*
  - Pre-processing approaches
    - proof reduction and compression
    - proof manipulation for interpolation
  - Proof senstive Interpolation

The Bird's Eye View to PeRIPLO



- Given an unsatisfiable propositional formula ϕ PeRIPLO constructs an interpolant in circuit form by

  1. Solving ϕ and extracting a compact proof skeleton from the SAT solver
  2. Expanding the proof skeleton to a resolution proof
  3. Optimizing the resolution proof to a smaller proof
  4. Constructing an interpolant from the optimized proof

PeRIPLO Features

- Basic interpolation:
  - $A \wedge B \rightarrow \bot$
  - $A \rightarrow I$ and $I \wedge B \rightarrow \bot$
  - $var(I) \subseteq var(A) \cap var(B)$

PeRIPLO Features

- Basic interpolation:
  - $A \wedge B \rightarrow \bot$
  - $A \rightarrow I$ and $I \wedge B \rightarrow \bot$
  - $var(I) \subseteq var(A) \cap var(B)$
- Variations:
  - Path, Tree and DAG interpolation

PeRIPLO Features

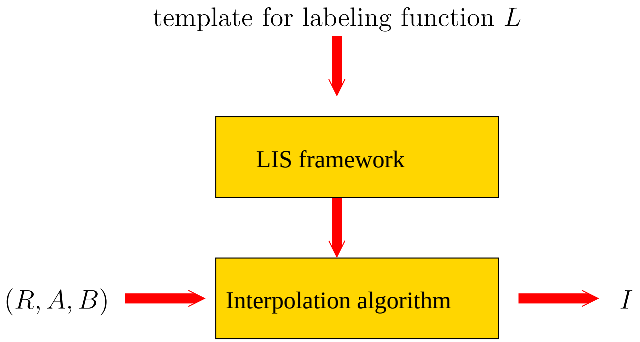
- Basic interpolation:
  - $A \wedge B \rightarrow \bot$
  - $A \rightarrow I$ and $I \wedge B \rightarrow \bot$
  - $var(I) \subseteq var(A) \cap var(B)$
- Variations:
  - Path, Tree and DAG interpolation
- Proof Optimization [FMSD15, ICCAD10]:
  - removing resolution steps which reintroduce already resolved pivot variables
  - postponing unit resolution steps until the end of the resolution proof
  - using different local rewriting rules which preserve the validity of the proof

PeRIPLO Interface

Features

- Multifaceted interface
  - A clear API for C++ for tuning the interpolation algorithm, inserting a formula to PeRIPLO, and fetching the interpolant from PeRIPLO
  - Reading and writing smtlib2
  - Reading and writing the Aiger format

- Labeled Interpolation System (LIS) framework [D'Silva et al. 2010]
  - construction of interpolation algorithms from labeling functions
  - generalization of various interpolation algorithms (i.e., M_s [McMillan03], P [Pudlak97], M_w [D'Silva10])



- Given a resolution proof R, (v, C) denotes that the variable v occurs in a clause C of R
- The labeling function L assigns a label from the set $\{a, b, ab\}$ to each occurrence (v, C) in R
- Given a propositional formula $A \wedge B$, a variable is either *local*, if it occurs only in A or B, or *shared* if it occurs in both A and B

Interpolation in PeRIPLO

Labeling Functions [VSSTE15]

- The label $L(v, C) = b$ if v is local to A and $L(v, C) = a$ if v is local to B.
- The label can be chosen freely for occurrences of shared variables
  - *Tuning* the label for the shared occurrences results in *different* interpolation algorithms

PeRIPLO and Different Interpolation Algorithms

- Labeling all shared variable occurrences as
  - *a* results in the weakest interpolant M_w available in LIS
  - *b* results in a strong interpolant M_s available in LIS
  - *ab* results in an interpolant P that is somewhere in the middle
- The aforementioned approaches are fixed schemas with no possibility for adopt to the task

PeRIPLO and Proof-Sensitive Interpolation

- PeRIPLO offers certain labeling functions that specifically address the interpolant size:
  - Labeling all occurrences in A as a and B as b results in an interpolant with minimum number of distinct variables
  - By analyzing the number of occurrences in the A and B part of the proof R it is possible to construct interpolants that have a small number of connectives

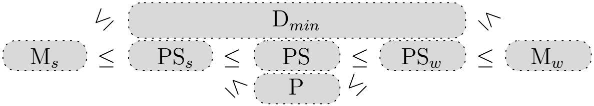
The Proof-Sensitive (PS) Labeling Functions

- PeRIPLO implements the proof-sensitive labeling functions specifically targeted for constructing small interpolants
- let $f_A(p) = |\{(p, C) \mid C \in A\}|$ be the number of times the variable p occurs in A, and $f_B(p) = |\{(p, C) \mid C \in B\}|$ the number the variable p occurs in B.
- The proof-sensitive labeling function L_{PS} is defined as

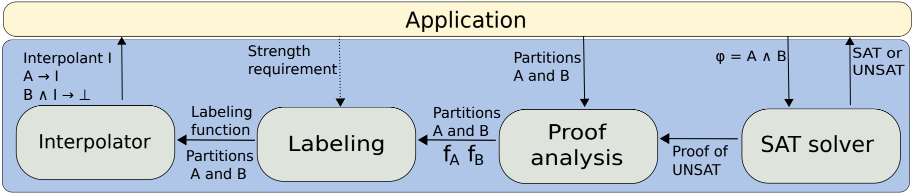
$$L_{PS}(p, C) = \begin{cases} a & \text{if } f_A(p) \geq f_B(p) \\ b & \text{if } f_A(p) < f_B(p). \end{cases}$$

The Proof-Sensitive Labeling Functions

- PeRIPLO also provides weak and strong versions of Proof-Sensitive Approach
- Hierarchy of Interpolation Algorithms provided by PeRIPLO



PeRIPLO API



- The API of PeRIPLO provides the application with a full control over the interpolant generation
- Many of the more routine tasks are implemented efficiently inside PeRIPLO so that the user does not need to take care of such details
- The system makes it comfortable to construct and experiment with new labeling functions

Reduction Approach Evaluation [ICCAD'10]

Experimental results over SMT: QF_UF, QF_IDL, QF_LRA, QF_RDL

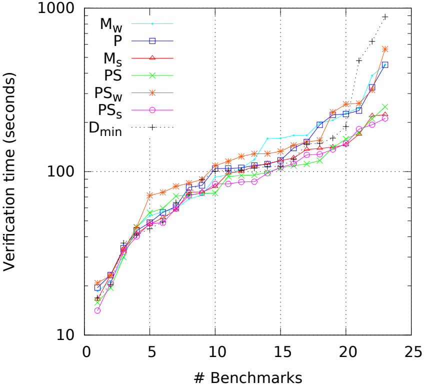
	#	Avg_{nodes}	Avg_{edges}	Avg_{core}	$T(s)$	Max_{nodes}	Max_{edges}	Max_{core}
RP	1370	6.7%	7.5%	1.3%	1.7	65.1%	68.9%	39.1%
Ratio								
0.01	1366	8.9%	10.7%	1.4%	3.4	66.3%	70.2%	45.7%
0.025	1366	9.8%	11.9%	1.5%	3.6	77.2%	79.9%	45.7%
0.05	1366	10.7%	13.0%	1.6%	4.1	78.5%	81.2%	45.7%
0.075	1366	11.4%	13.8%	1.7%	4.5	78.5%	81.2%	45.7%
0.1	1364	11.8%	14.4%	1.7%	5.0	78.8%	83.6%	45.7%
0.25	1359	13.6%	16.6%	1.9%	7.6	79.6%	84.4%	45.7%
0.5	1348	15.0%	18.4%	2.0%	11.5	79.1%	85.2%	45.7%
0.75	1341	16.0%	19.5%	2.1%	15.1	79.9%	86.1%	45.7%
1	1337	16.7%	20.4%	2.2%	18.8	79.9%	86.1%	45.7%

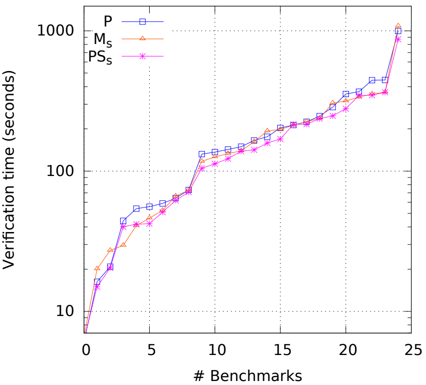
- *Ratio* — time threshold as fraction of solving time
- # — number of benchmarks solved
- Avg_{nodes}, Avg_{edges}, Avg_{core} — average reduction in proof size
- $T(s)$ — average transformation time in seconds
- Max_{nodes}, Max_{edges}, Max_{core} — max reduction in proof size

Applications - FunFrog [FMCAD12] and eVolCheck [TACAS13]

- Bounded Model Checkers
- Interpolants used as Function Summaries
- FunFrog - C Incremental Checker
  - Stronger interpolants suit better [CAV12]
  - http://verify.inf.usi.ch/funfrog
- eVolCheck - C Upgrade Checker
  - Weaker interpolants suit better [CAV12]
  - http://verify.inf.usi.ch/evolcheck

Applications - FunFrog and eVolCheck





PS and PS_s consistently lead to better verification time

Conclusions

- PeRIPLO is an interpolation tool for propositional logic
  - Generic and flexible framework for producing interpolants on demand
  - Provides an API, an smtlib2, and an AIGER interface for communicating with other tools
  - Particular emphasis on constructing small interpolants while maintaining guarantees of interpolant strength
  - For more information see http://verify.inf.usi.ch/periplo!

Future work

- Extending the interpolation to first-oder logics in SMT

Thank you for your attention!