

Machine learning for instance selection in SMT solving

(WORK IN PROGRESS)

Jasmin Christian Blanchette^{1,2} Daniel El Ouraoui² Pascal Fontaine² Cezary Kaliszyk³

Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

University of Lorraine, CNRS, Inria, and LORIA, Nancy, France

University of Innsbruck, Innsbruck, Austria

9th April 2019

Contents

- 1 Introduction
- 2 CDCL(T)
- 3 Instantiation techniques
- 4 Machine learning for instance selection
- 5 Evaluation
- 6 Conclusion

Contents

1 Introduction

2 CDCL(T)

3 Instantiation techniques

4 Machine learning for instance selection

5 Evaluation

6 Conclusion

Satisfiability modulo theories (SMT)

- Automation
 - Proof assistant
 - Verification conditions
 - Model checking
- Solvers
 - Z3, CVC4, VERIT, ...

Instantiation

- Hard for SMT solvers
- Heuristically solved

Challenge

- Improve instantiation techniques
- Solve more problems
- Be more efficient



Université de Lorraine/UFRN (<http://www.verit-solver.org>)

Contents

1 Introduction

2 CDCL(T)

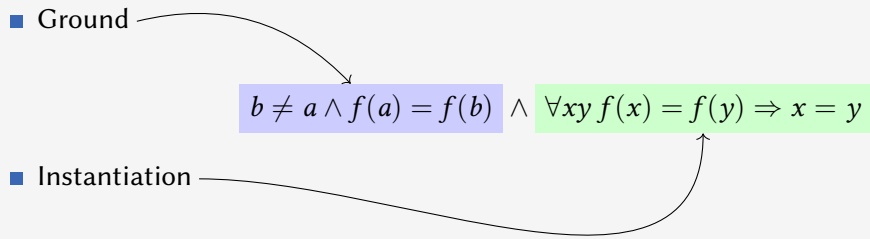
3 Instantiation techniques

4 Machine learning for instance selection

5 Evaluation

6 Conclusion

■ Ground



The diagram shows two logical expressions connected by an AND symbol. The first expression, $b \neq a \wedge f(a) = f(b)$, is highlighted in a light blue box. The second expression, $\forall xy f(x) = f(y) \Rightarrow x = y$, is highlighted in a light green box. An arrow points from the 'Ground' label to the first expression, and another arrow points from the 'Instantiation' label to the second expression.

$$b \neq a \wedge f(a) = f(b) \wedge \forall xy f(x) = f(y) \Rightarrow x = y$$

■ Instantiation

How efficiently check the satisfiability of a ground formula

$$(f(a, b) = g(a) \vee d = b) \wedge d = g(b) \wedge d \neq f(a, b) \wedge b = a \wedge d \neq g(a)$$

How efficiently check the satisfiability of a ground formula

$$\underbrace{(f(a, b) = g(a))}_{l_1} \vee \underbrace{d = b}_{l_2} \wedge \underbrace{d = g(b)}_{l_3} \wedge \underbrace{d \neq f(a, b)}_{l_4} \wedge \underbrace{b = a}_{l_5} \wedge \underbrace{d \neq g(a)}_{l_6}$$

How efficiently check the satisfiability of a ground formula

$$\underbrace{(f(a, b) = g(a))}_{l_1} \vee \underbrace{d = b}_{l_2} \wedge \underbrace{d = g(b)}_{l_3} \wedge \underbrace{d \neq f(a, b)}_{l_4} \wedge \underbrace{b = a}_{l_5} \wedge \underbrace{d \neq g(a)}_{l_6}$$

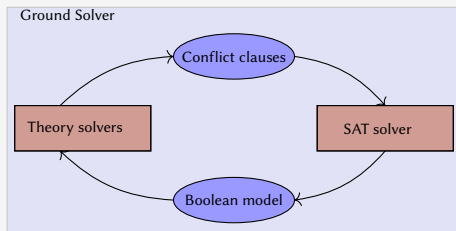


How efficiently check the satisfiability of a ground formula

$$\frac{(f(a, b) = g(a)) \vee \underline{d = b}}{l_1} \wedge \frac{d = g(b)}{l_3} \wedge \frac{d \neq f(a, b)}{l_4} \wedge \frac{b = a}{l_5} \wedge \frac{d \neq g(a)}{l_6}$$



$$(l_1 \vee \neg l_2) \wedge l_3 \wedge l_4 \wedge l_5 \wedge l_6$$

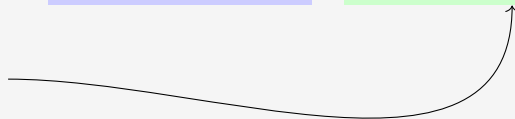


- Formulas are embedded in SAT
- SAT solver produces a boolean model
- Theory solvers produce conflict clauses
- Conflict clauses guide the SAT solver

First-Order problem

$$b \neq a \wedge f(a) = f(b) \wedge \forall xy f(x) = f(y) \Rightarrow x = y$$

■ Instantiation



How to find an instance such that the problem is UNSAT

$$b \neq a \wedge f(a) = f(b) \wedge \forall xy f(x) = f(y) \Rightarrow x = y$$

How to find an instance such that the problem is UNSAT

$$\frac{b \neq a \wedge f(a) = f(b) \wedge \forall xy f(x) = f(y) \Rightarrow x = y}{\text{SAT}}$$

How to find an instance such that the problem is UNSAT

$$\frac{b \neq a \wedge f(a) = f(b) \wedge \forall xy f(x) = f(y) \Rightarrow x = y}{\text{SAT}}$$



How to find an instance such that the problem is UNSAT

$$\frac{b \neq a \wedge f(a) = f(b) \wedge \forall xy f(x) = f(y) \Rightarrow x = y}{\text{SAT}}$$

↓

$$f(a) \neq f(b) \vee a = b$$

How to find an instance such that the problem is UNSAT

$$\frac{b \neq a \wedge f(a) = f(b) \wedge \forall xy f(x) = f(y) \Rightarrow x = y}{\text{SAT}}$$

↓

$$f(a) \neq f(b) \vee a = b$$

↓

How to find an instance such that the problem is UNSAT

$$\frac{b \neq a \wedge f(a) = f(b) \wedge \forall xy f(x) = f(y) \Rightarrow x = y}{\text{SAT}}$$

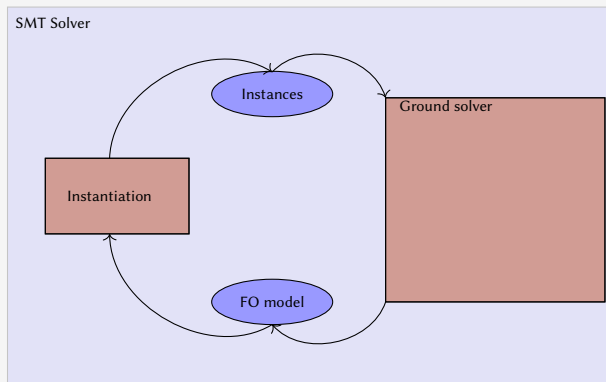
↓

$$f(a) \neq f(b) \vee a = b$$

↓

$$\frac{b \neq a \wedge f(a) = f(b) \wedge f(a) \neq f(b)}{\text{UNSAT}}$$

First-Order CDCL(T)



Contents

1 Introduction

2 CDCL(T)

3 Instantiation techniques

4 Machine learning for instance selection

5 Evaluation

6 Conclusion

- **Conflict based instantiation** Introduced by Reynolds, this technique produces relevant sets of instances. The idea is that, given a ground model \mathcal{M} and a quantified formula $\forall(\bar{x}_n : \bar{\tau}_n).\varphi$, we find a substitution σ such that $\mathcal{M} \models \neg\varphi\sigma$.
- **Congruence Closure with Free Variable (CCFV)** Introduced by Barbosa et al., generalizes the idea of Conflict based instantiation by reasoning over equivalence classes.

Enumerative instantiation $\forall(x : \tau).\psi[x] \equiv \bigwedge_{t \in \mathcal{D}_\tau} \psi[t]$

Enumerate all ground terms over the domain of x (aka. Herbrand universe)

Trigger based instantiation

Triggers

A trigger T for a quantified formula $\forall \bar{x}_n.\psi$ is a set of non-ground terms $u_1, \dots, u_n \in \mathbf{T}(\psi)$ such that: $\{\bar{x}\} \subseteq \text{FV}(u_1) \cup \dots \cup \text{FV}(u_n)$.

$$E = f(a) \simeq g(b), \quad a \simeq g(b)$$

$$Q = \forall x f(g(x)) \not\approx g(x)$$

$$T = f(g(x))$$

$f(a)$ E -matches $f(g(x))$ under $x \mapsto b$

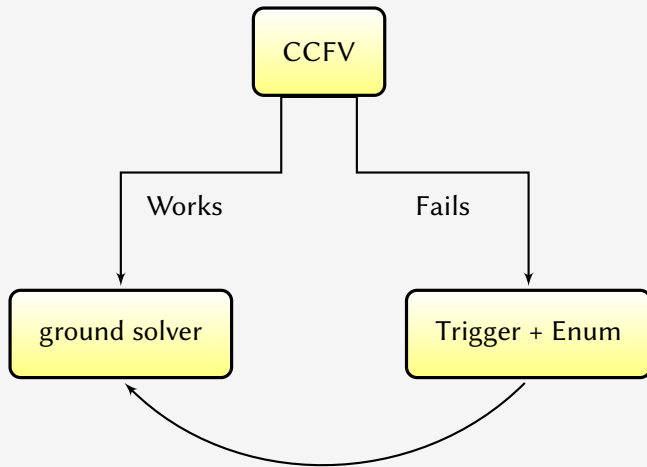


Figure: Instantiation strategie

Conflict based instantiation and CCFV :

Pro Efficient, if find substitution kill the model

Pro All generated instances are useful

Cons Finds contradiction involving only one instance

Enumerative and Trigger based instantiation :

Pro Useful when CCFV fail

Cons Many heuristics

Cons Generates a lot of junk, and many instances

Summarize

Conflict based instantiation and CCFV :

Pro Efficient, if find substitution kill the model

Pro All generated instances are useful

Cons Finds contradiction involving only one instance

Enumerative and Trigger based instantiation :

Pro Useful when CCFV fail

Cons Many heuristics

Cons Generates a lot of junk, and many instances

Indeed

This is what we want improve!

Contents

1 Introduction

2 CDCL(T)

3 Instantiation techniques

4 Machine learning for instance selection

5 Evaluation

6 Conclusion

- **How many lemmas are generated to solve a problem?**
 - around 300 for the UF category of the SMT-LIB
 - some generate more than 100 000 instances
- **How many lemmas are needed to solve a problem?**
 - Only 10% of this number, and sometimes much less

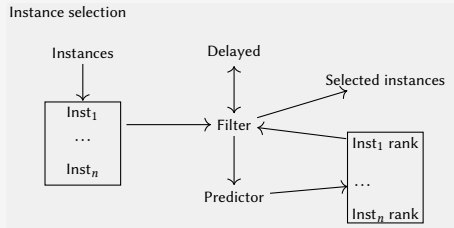
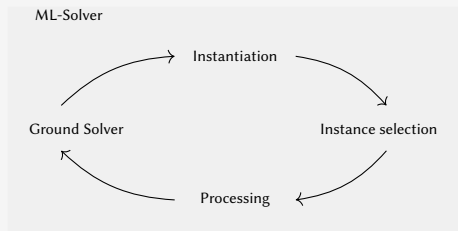
- **How many lemmas are generated to solve a problem?**
 - around 300 for the UF category of the SMT-LIB
 - some generate more than 100 000 instances
- **How many lemmas are needed to solve a problem?**
 - Only 10% of this number, and sometimes much less

Question

Could we select the good one?

Our approach

- Instances in a priority queue
- Encode instances
- Call predictor
- Several strategies for selection

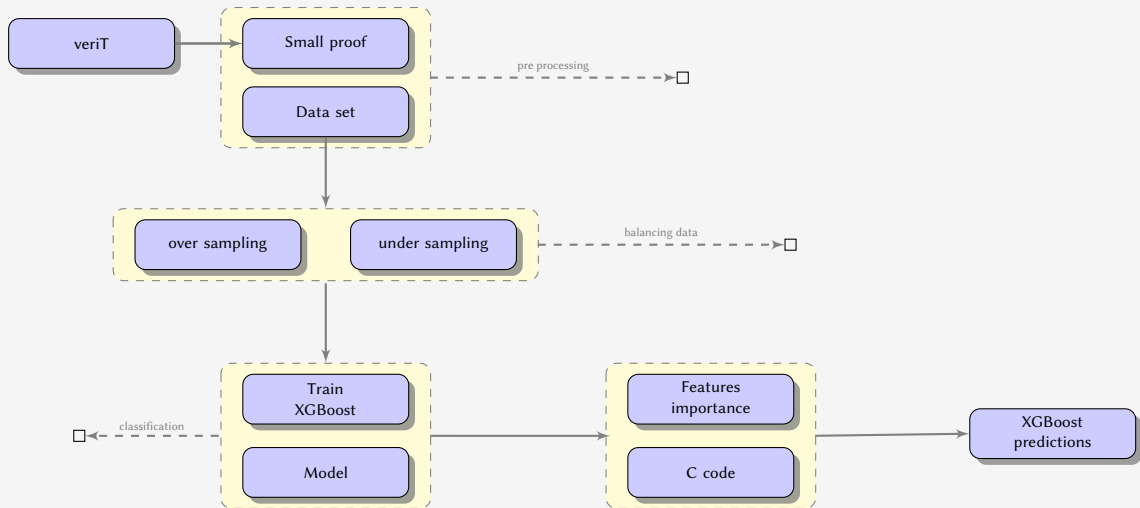


State description

$$\begin{array}{ccc} \text{Model} & \text{Formula} & \text{Instances} \\ \hline (l_1, \dots, l_n, \forall \overline{x_n} . \psi[\overline{x_n}], x_1 \mapsto t_1, \dots, x_n \mapsto t_n) \end{array}$$

$$\text{rounds} \left\{ \begin{array}{l} (model_1 \quad Qformula_1^1 \quad Inst_{1,1}^1 \quad \dots \quad Inst_{1,m}^1) \\ (model_1 \quad Qformula_1^2 \quad Inst_{1,1}^2 \quad \dots \quad Inst_{1,m}^2) \\ \dots \\ (model_k \quad Qformula_k^i \quad Inst_{k,1}^i \quad \dots \quad Inst_{k,m}^i) \end{array} \right\} \hookrightarrow \begin{bmatrix} 0 & x_{12} & x_{13} & \dots & x_{1n} \\ 1 & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & x_{d2} & x_{d3} & \dots & x_{dn} \end{bmatrix}$$

Experiments



Contents

1 Introduction

2 CDCL(T)

3 Instantiation techniques

4 Machine learning for instance selection

5 Evaluation

6 Conclusion

- Experiments run on UF SMTLIB benchmarks with 120s timeout
- veriT without learning solves 2923
- veriT with learning solves 2939 with learning

Evaluation on test + training set

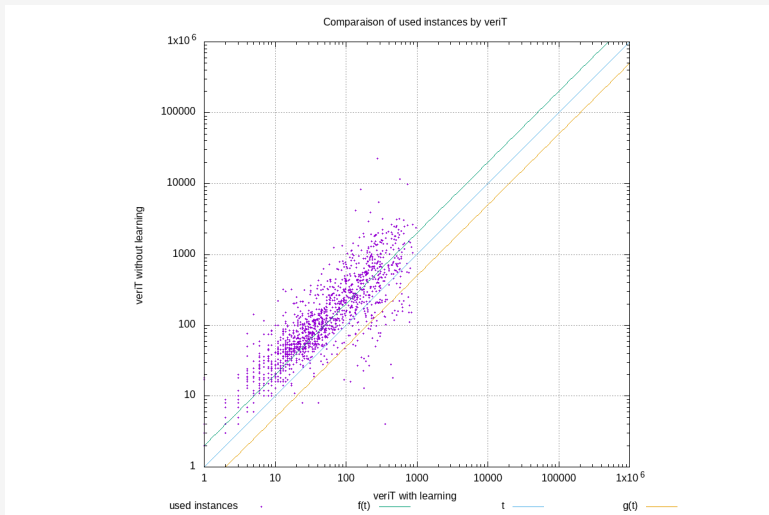


Figure: comparison of veriT configurations on UF SMT-LIB benchmarks.

Evaluation on test set only

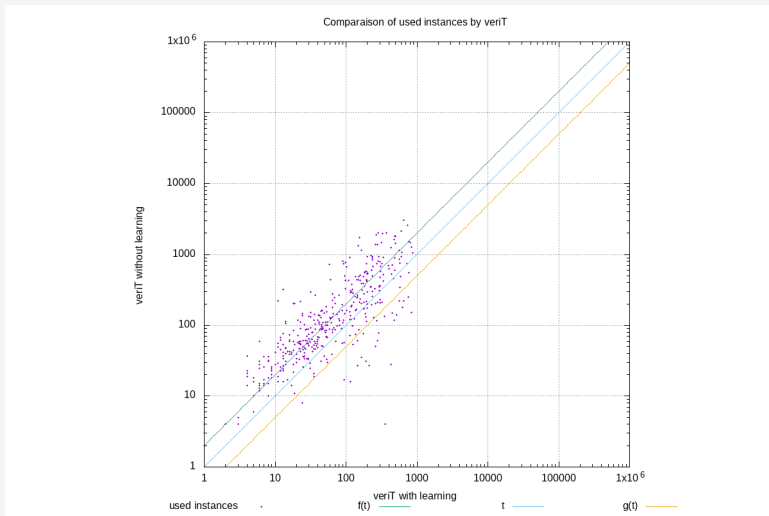


Figure: comparison of VERiT configurations on UF SMT-LIB benchmarks.

Contents

1 Introduction

2 CDCL(T)

3 Instantiation techniques

4 Machine learning for instance selection

5 Evaluation

6 Conclusion

- Could be a significant improvement
- Reduces the number of instances by two in average
- Reinforcement learning
- Features embedding can be improved

Thank you for you attention
Questions or suggestions?

	<i>All</i>			<i>Test only</i>		
	unsat	avg	less	unsat	avg	less
<i>with learning</i>	1443	113	1317	423	130	363
<i>without learning</i>	1443	318	128	423	264	62

Table: VERIT configurations on UF SMTLIB benchmarks with 30s timeout.

Features encoding

Terms abstraction

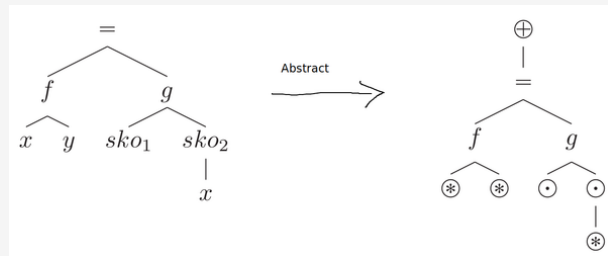
- Variables
- Skolem constants
- Polarity

Features

- FEATURE: Literal $\rightarrow \Sigma^3$
- FEATURES: $\Sigma^3 \rightarrow \mathbb{N}$
- Occurrences of term walks

Example

FEATURES $(f(x, y) = g(sk1, sk2(x))) = (\oplus, =, f) \mapsto 1, (\oplus, =, g) \mapsto 1, (=, f, *) \mapsto 2, (=, g, \odot) \mapsto 2, (g, \odot, *) \mapsto 1$



State description version 2

$$\left(\overbrace{E_{t_1}, D_{t_1}, \dots, E_{t_n}, D_{t_n}}^{\text{Model}}, \overbrace{T_1, \dots, T_n}^{\text{Triggers}}, \overbrace{x_1 \mapsto t_1, \dots, x_n \mapsto t_n}^{\text{Instances}} \right)$$

- E_{t_i} is the congruence class of t_i
- D_{t_i} is the set of all terms explicitly disequals with t_i
- T_i is the set of triggers of x_i

This description reduce drastically the size of the problem