Quantifier handling in SMT

Pascal Fontaine

Univ. of Lorraine, CNRS, Inria, LORIA

IFSE: journées FAC

Toulouse, 28 March 2019

Misc.

Thanks!

Presentation based on the work and material of many. Among others:

- Andrew J. Reynolds
- Haniel Barbosa
- Leonardo de Moura
- Bruno Dutertre
- ...

SMT = SAT + expressiveness

- SAT solvers

$$\neg\big[\,(p \Rightarrow q) \Rightarrow \big[\,(\neg p \Rightarrow q) \Rightarrow q\big]\big]$$

- Congruence closure (uninterpreted symbols + equality)

$$a = b \land \big[f(a) \neq f(b) \lor (q(a) \land \neg q(b))\big]$$

- adding arithmetic

$$a \leq b \land b \leq a + x \land x = 0 \land \big[f(a) \neq f(b) \lor (q(a) \land \neg q(b + x))\big]$$

- ...
- What about quantifiers?

Quantifiers in SMT

Why?

- SMT theories are often not sufficient
  What if you need your own ones?
- Verification: e.g. reasoning about all processes ($\forall p$)
- Expressivity

This talk is not about:

- quantifier elimination, e.g. for Presburger or real closed fields
- SMT finite model finding [Reynolds13]
- superposition
- extensions of SAT/ground SMT towards full FOL and a long list of works in between FOL ATP and SMT, e.g. Avatar [Voronkov14], Inst-Gen [Korovin13], SGGS [Bonacina17], Model-Evolution [Baumgartner14], SUP(LA) [Althaus09],...
- ...

Quantifiers in SMT

- ☹ Full first-order logic is undecidable
  there is no decision procedure that always terminates, and always provide a SAT or UNSAT answer
- ☺ First-order logic is semi-decidable
  refutationally complete procedures terminate on UNSAT
- ☺ if finite model property, then decidable
- ☹ Presburger with even one unary predicate is not even semi-decidable [Halper91]
- ☺ Pragmatic approaches are quite successful

Why does the pragmatic SMT approach work?

- Verification problems are big and shallow
- FOL provers more suitable to find intricate proofs
- SMT solvers good to deal with long, mostly ground, reasoning

Working hypothesis

Quantifier handling for pure FOL will work well enough for SMT

Outline

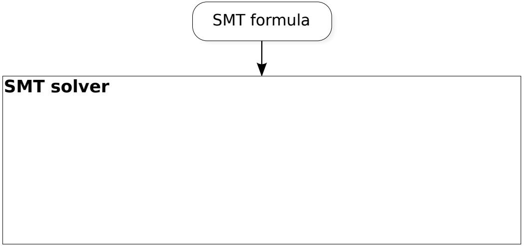
Outline

Standard techniques

- Moving quantifiers around: prenex form
- Eliminating one kind of quantifiers: Skolemization
- From arbitrary Boolean combination to sets of clauses: CNF transformation

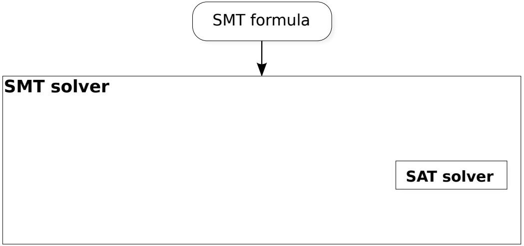
We will assume when needed that quantified formulas are universally quantified clauses

From SAT to SMT,. . . and then to quantified SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x)) \right]$

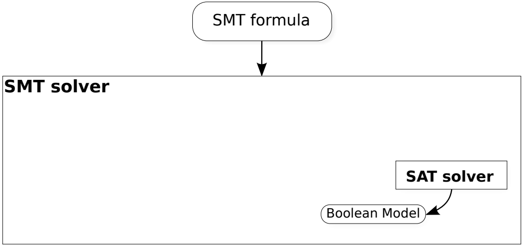
From SAT to SMT,. . . and then to quantified SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$

To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$

From SAT to SMT,. . . and then to quantified SMT

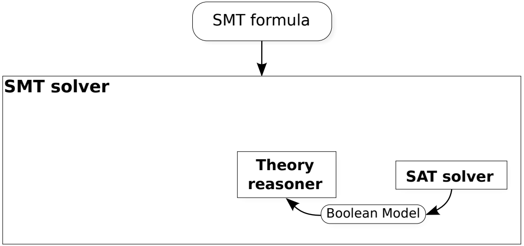


Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$

To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$

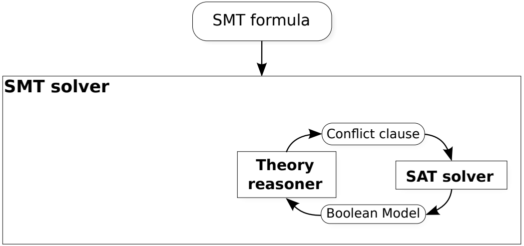
Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

From SAT to SMT,. . . and then to quantified SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x)) \right]$
To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)}) \right]$
Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$
Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

From SAT to SMT,... and then to quantified SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$

To SAT solver: $p_{a\leq b} \wedge p_{b\leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$

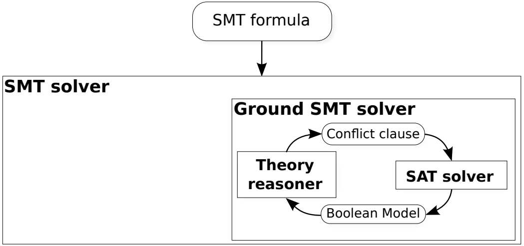
Boolean model: $p_{a\leq b}, p_{b\leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

New clause: $\neg p_{a\leq b} \vee \neg p_{b\leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

Conflict clauses are negation of unsatisfiable conjunctive sets of literals

From SAT to SMT,...and then to quantified SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b+x)) \right]$

To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)}) \right]$

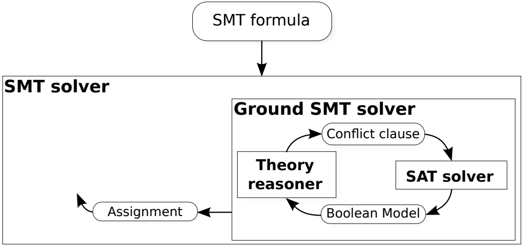
Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

New clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

Conflict clauses are negation of unsatisfiable conjunctive sets of literals

From SAT to SMT,. . . and then to quantified SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$

To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$

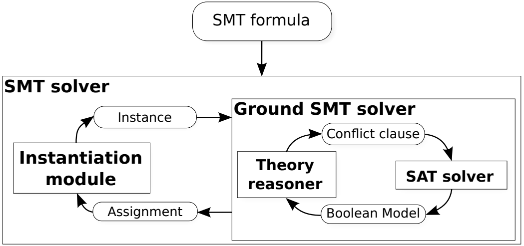
Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

New clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

Conflict clauses are negation of unsatisfiable conjunctive sets of literals

From SAT to SMT,. . . and then to quantified SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$

To SAT solver: $p_{a\leq b} \wedge p_{b\leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$

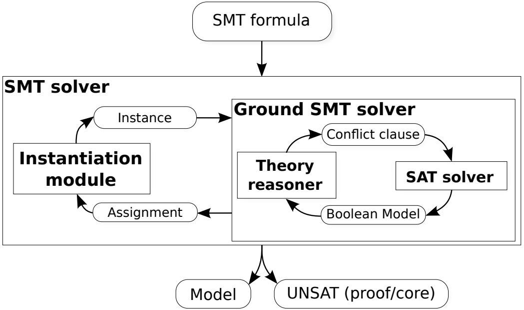
Boolean model: $p_{a\leq b}, p_{b\leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

New clause: $\neg p_{a\leq b} \vee \neg p_{b\leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

Conflict clauses are negation of unsatisfiable conjunctive sets of literals

From SAT to SMT,... and then to quantified SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x)) \right]$

To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)}) \right]$

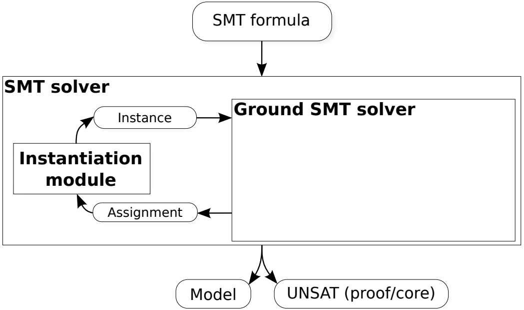
Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

New clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

Conflict clauses are negation of unsatisfiable conjunctive sets of literals

From SAT to SMT,. . . and then to quantified SMT



Input: $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \left[f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\right]$

To SAT solver: $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \left[\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\right]$

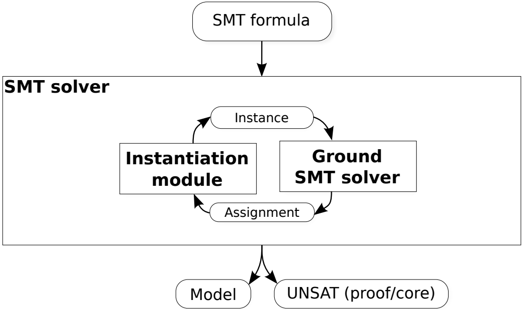
Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$ unsatisfiable

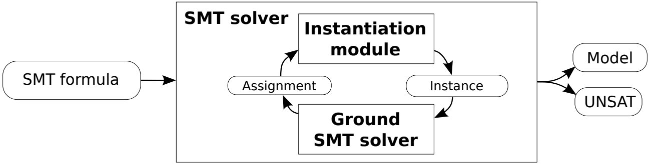
New clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

Conflict clauses are negation of unsatisfiable conjunctive sets of literals

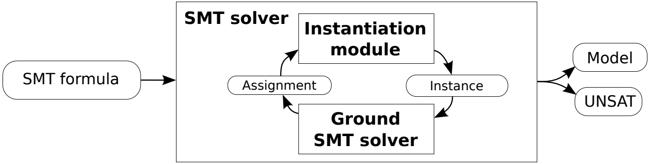
From SAT to SMT,...and then to quantified SMT



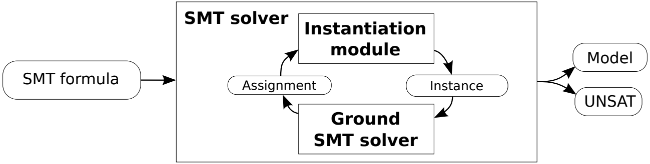
From SAT to SMT,. . . and then to quantified SMT



Instance?

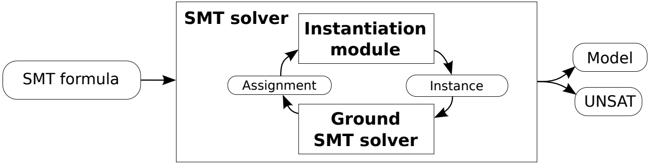

SMT formula
SMT solver
Instantiation module
Assignment
Instance
Ground SMT solver
Model
UNSAT

Instance?



Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \big[\forall x\, Q(x) \vee \forall x\,.\, S(x) \equiv R(x)\big]$

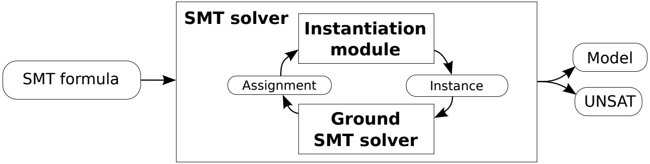
Instance?



Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \left[\forall x\, Q(x) \vee \forall x \,.\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \wedge p_{S(b)} \wedge \neg p_{Q(a)} \wedge \neg p_{R(a)} \wedge \left[p_{\forall x\, Q(x)} \vee p_{\forall x \,.\, S(x) \equiv R(x)}\right]$

Instance?

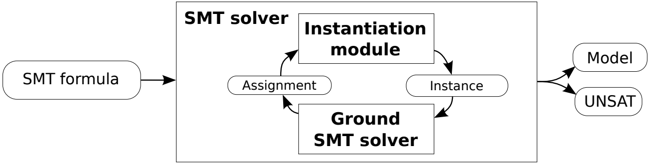


Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \left[\forall x\, Q(x) \vee \forall x\,.\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \wedge p_{S(b)} \wedge \neg p_{Q(a)} \wedge \neg p_{R(a)} \wedge \left[p_{\forall x\, Q(x)} \vee p_{\forall x\,.\, S(x) \equiv R(x)}\right]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x\,.\, S(x) \equiv R(x)}$

Instance?



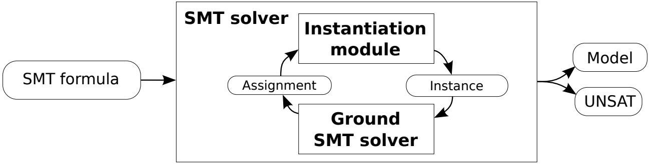
Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \left[\forall x\, Q(x) \vee \forall x \,.\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \wedge p_{S(b)} \wedge \neg p_{Q(a)} \wedge \neg p_{R(a)} \wedge \left[p_{\forall x\, Q(x)} \vee p_{\forall x \,.\, S(x) \equiv R(x)}\right]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x \,.\, S(x) \equiv R(x)}$

Theory reasoner: fine! ...but does not understand $\forall x \,.\, S(x) \equiv R(x)$

Instance?



Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \left[\forall x\, Q(x) \vee \forall x \,.\, S(x) \equiv R(x)\right]$

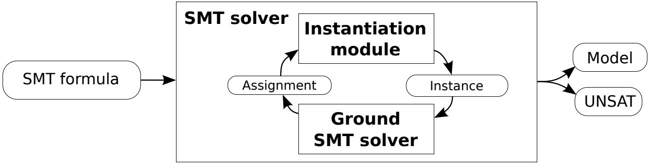
To SAT solver: $p_{a=b} \wedge p_{S(b)} \wedge \neg p_{Q(a)} \wedge \neg p_{R(a)} \wedge \left[p_{\forall x\, Q(x)} \vee p_{\forall x \,.\, S(x) \equiv R(x)}\right]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x \,.\, S(x) \equiv R(x)}$

Theory reasoner: fine! . . . but does not understand $\forall x \,.\, S(x) \equiv R(x)$

Instantiation module: there is something to do with $\forall x \,.\, S(x) \equiv R(x)$

Instance?



Input: $a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land \left[\forall x\, Q(x) \lor \forall x \,.\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land \left[p_{\forall x\, Q(x)} \lor p_{\forall x \,.\, S(x) \equiv R(x)}\right]$

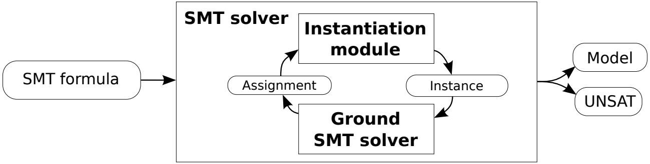
Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x \,.\, S(x) \equiv R(x)}$

Theory reasoner: fine! ...but does not understand $\forall x \,.\, S(x) \equiv R(x)$

Instantiation module: there is something to do with $\forall x \,.\, S(x) \equiv R(x)$

New clause:$\neg p_{a=b}, \neg p_{S(b)} \lor p_{R(a)} \lor \neg p_{\forall x \,.\, S(x) \equiv R(x)}$

Instance?



Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \left[\forall x\, Q(x) \vee \forall x\, .\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \wedge p_{S(b)} \wedge \neg p_{Q(a)} \wedge \neg p_{R(a)} \wedge \left[p_{\forall x\, Q(x)} \vee p_{\forall x\, .\, S(x) \equiv R(x)}\right]$

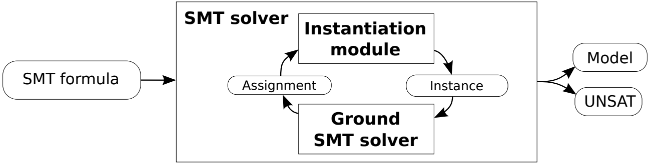
Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x\, .\, S(x) \equiv R(x)}$

Theory reasoner: fine! . . . but does not understand $\forall x\, .\, S(x) \equiv R(x)$

Instantiation module: there is something to do with $\forall x\, .\, S(x) \equiv R(x)$

New clause:$\neg p_{a=b}, \neg p_{S(b)} \vee p_{R(a)} \vee \neg p_{\forall x\, .\, S(x) \equiv R(x)}$
. . . too complicated to find/generate

Instance?



Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \left[\forall x\, Q(x) \vee \forall x \,.\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \wedge p_{S(b)} \wedge \neg p_{Q(a)} \wedge \neg p_{R(a)} \wedge \left[p_{\forall x\, Q(x)} \vee p_{\forall x \,.\, S(x) \equiv R(x)}\right]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x \,.\, S(x) \equiv R(x)}$

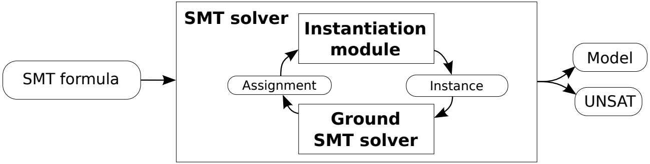
Theory reasoner: fine! . . . but does not understand $\forall x \,.\, S(x) \equiv R(x)$

Instantiation module: there is something to do with $\forall x \,.\, S(x) \equiv R(x)$

New clause:$\neg p_{a=b}, \neg p_{S(b)} \vee p_{R(a)} \vee \neg p_{\forall x \,.\, S(x) \equiv R(x)}$
. . . too complicated to find/generate

What is the right formula to generate?

Instance?



Input: $a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land \left[\forall x\, Q(x) \lor \forall x\, .\, S(x) \equiv R(x)\right]$

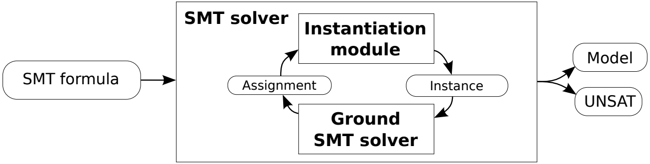
To SAT solver: $p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land \left[p_{\forall x\, Q(x)} \lor p_{\forall x\, .\, S(x) \equiv R(x)}\right]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x\, .\, S(x) \equiv R(x)}$

Instantiation module: there is something to do with $\forall x\, .\, S(x) \equiv R(x)$

What is the right formula to generate?

Instance?



Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \left[\forall x\, Q(x) \vee \forall x\,.\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \wedge p_{S(b)} \wedge \neg p_{Q(a)} \wedge \neg p_{R(a)} \wedge \left[p_{\forall x\, Q(x)} \vee p_{\forall x\,.\, S(x) \equiv R(x)}\right]$

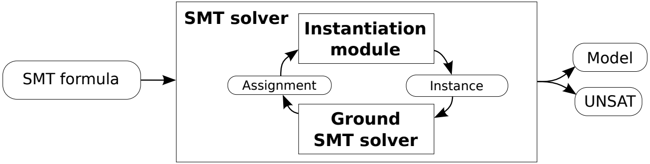
Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x\,.\, S(x) \equiv R(x)}$

Instantiation module: there is something to do with $\forall x\,.\, S(x) \equiv R(x)$

What is the right formula to generate?

$S(a) \equiv R(a)$ is not right

Instance?



Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \left[\forall x\, Q(x) \vee \forall x \,.\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \wedge p_{S(b)} \wedge \neg p_{Q(a)} \wedge \neg p_{R(a)} \wedge \left[p_{\forall x\, Q(x)} \vee p_{\forall x \,.\, S(x) \equiv R(x)}\right]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x \,.\, S(x) \equiv R(x)}$

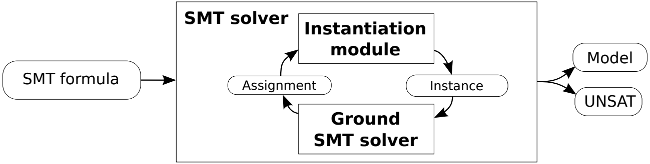
Instantiation module: there is something to do with $\forall x \,.\, S(x) \equiv R(x)$

What is the right formula to generate?

$S(a) \equiv R(a)$ is not right

We want $S(a) \equiv R(a)$ whenever $p_{\forall x \,.\, S(x) \equiv R(x)}$ is in the Boolean model

Instance?



Input: $a = b \land S(b) \land \neg Q(a) \land \neg R(a) \land \left[\forall x\, Q(x) \lor \forall x\,.\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \land p_{S(b)} \land \neg p_{Q(a)} \land \neg p_{R(a)} \land \left[p_{\forall x\, Q(x)} \lor p_{\forall x\,.\, S(x) \equiv R(x)}\right]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x\,.\, S(x) \equiv R(x)}$

Instantiation module: there is something to do with $\forall x\,.\, S(x) \equiv R(x)$

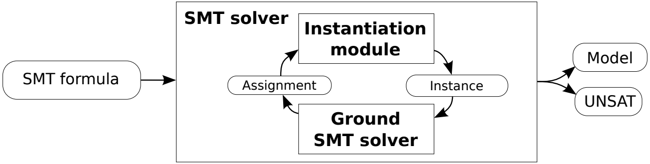
What is the right formula to generate?

$S(a) \equiv R(a)$ is not right

We want $S(a) \equiv R(a)$ whenever $p_{\forall x\,.\, S(x) \equiv R(x)}$ is in the Boolean model

$(\forall x\,.\, S(x) \equiv R(x)) \Rightarrow (S(a) \equiv R(a))$ would do

Instance?



Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \left[\forall x\, Q(x) \vee \forall x\,.\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \wedge p_{S(b)} \wedge \neg p_{Q(a)} \wedge \neg p_{R(a)} \wedge \left[p_{\forall x\, Q(x)} \vee p_{\forall x\,.\, S(x) \equiv R(x)}\right]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x\,.\, S(x) \equiv R(x)}$

Instantiation module: there is something to do with $\forall x\,.\, S(x) \equiv R(x)$

What is the right formula to generate?

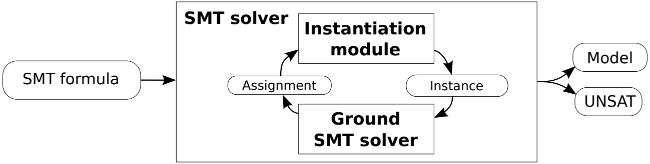
$S(a) \equiv R(a)$ is not right

We want $S(a) \equiv R(a)$ whenever $p_{\forall x\,.\, S(x) \equiv R(x)}$ is in the Boolean model

$(\forall x\,.\, S(x) \equiv R(x)) \Rightarrow (S(a) \equiv R(a))$ would do

$\neg p_{\forall x\,.\, S(x) \equiv R(x)} \vee \left(p_{S(a)} \equiv p_{R(a)}\right)$ at the propositional level

Instance?



Input: $a = b \wedge S(b) \wedge \neg Q(a) \wedge \neg R(a) \wedge \left[\forall x\, Q(x) \vee \forall x\,.\, S(x) \equiv R(x)\right]$

To SAT solver: $p_{a=b} \wedge p_{S(b)} \wedge \neg p_{Q(a)} \wedge \neg p_{R(a)} \wedge \left[p_{\forall x\, Q(x)} \vee p_{\forall x\,.\, S(x) \equiv R(x)}\right]$

Boolean model: $p_{a=b}, p_{S(b)}, \neg p_{Q(a)}, \neg p_{R(a)}, p_{\forall x\,.\, S(x) \equiv R(x)}$

Instantiation module: there is something to do with $\forall x\,.\, S(x) \equiv R(x)$

What is the right formula to generate?

$S(a) \equiv R(a)$ is not right

We want $S(a) \equiv R(a)$ whenever $p_{\forall x\,.\, S(x) \equiv R(x)}$ is in the Boolean model

$(\forall x\,.\, S(x) \equiv R(x)) \Rightarrow (S(a) \equiv R(a))$ would do

$\neg p_{\forall x\,.\, S(x) \equiv R(x)} \vee \left(p_{S(a)} \equiv p_{R(a)}\right)$ at the propositional level

Together with $\forall x\, Q(x) \Rightarrow Q(a)$, this grounds the problem

Instance in an SMT context

$$\forall \bar{x}\, \varphi(\bar{x}) \Rightarrow \varphi\sigma$$

where σ is a ground substitution for variables $\bar{x}$

E.g. $\forall \bar{x}\, \varphi(\bar{x})$ is $\forall x \,.\, S(x) \equiv R(x)$, σ is $x \mapsto a$, $\varphi\sigma$ is $S(a) \equiv R(a)$

Remarks

- Above formula is a FOL tautology. E.g. $(\forall x \,.\, S(x) \equiv R(x)) \Rightarrow (S(a) \equiv R(a))$
- $\forall \bar{x}\, \varphi(\bar{x})$ gets abstracted as a propositional variable in the SAT solver, that has a meaning only for the instantiation module
- $\varphi\sigma$ gets abstracted as a Boolean combination of propositional variables. . .
- . . . that have meaning at the level of the *ground* theory reasoner
- $\varphi\sigma$ gets "activated"/relevant only in the models where $p_{\forall \bar{x}\, \varphi(\bar{x})}$ is true.

We might refer to $\varphi\sigma$ as the instance, but remember: all is fine at the level of the SAT solver/ground SMT solver

Outline

Introduction

Quantifiers and SMT: the basics

Instantiation techniques

- E-matching/trigger-based instantiation (**e**)
- Conflict-based instantiation (**c**)
- Model-based instantiation (**m**)
- Enumerative instantiation (**u**)
- Experimental evaluation

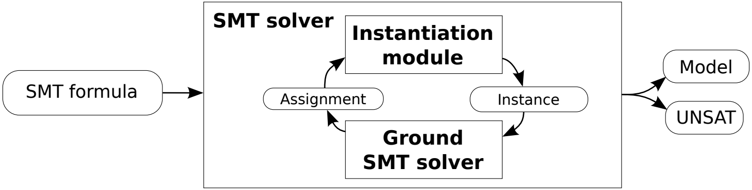
Conclusion

References

Outline

Instantiation techniques

The framework



Ground SMT solver enumerates assignments $E \cup Q$

- E set of ground literals
- Q set of quantified clauses

Instantiation module generates instances of Q that will further feed E

classic Herbrand Theorem: instantiate with all possible terms in language

Outline

E-matching/Trigger-based instantiation (**e**) [Detlefs05]

Search for relevant instances according to a set of triggers and E-matching

E-matching/Trigger-based instantiation ($\mathbf{e}$) [Detlefs05]

Search for relevant instances according to a set of triggers and E-matching

- $E = \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and $Q = \{\forall x.\, P(x) \vee R(x)\}$
- Assume trigger $P(x)$
- Find substitution σ for x such $P(x)$ is a know term (in E)
- Suitable substitutions are $x \mapsto a$, $x \mapsto b$, or $x \mapsto c$
  E.g. $E \models P(x)[x/a] = P(a)$ and $P(a) \in E$
- Formally

$\mathbf{e}(E, \forall \bar{x}.\, \varphi)$

1. Select a set of triggers $\{\bar{t}_1, \ldots \bar{t}_n\}$ for $\forall \bar{x}.\, \varphi$
2. For each $i = 1, \ldots, n$, select a set of substitutions S_i s.t for each $\sigma \in S_i$, $E \models \bar{t}_i \sigma = \bar{g}_i$ for some tuple $\bar{g}_i \in \mathcal{T}_E$.
3. Return $\bigcup_{i=1}^{n} S_i$

E-matching/Trigger-based instantiation

Ideal for expanding definitions/rewriting rules

- Example

$$\forall x \forall y \,.\, \mathrm{sister}(x, y) \equiv \\ (\mathrm{female}(x) \wedge \mathrm{mother}(x) = \mathrm{mother}(y) \wedge \mathrm{father}(x) = \mathrm{father}(y))$$

$\mathrm{sister}(\mathrm{Eliane}, \mathrm{Eloïse})$

$\mathrm{sister}(\mathrm{Eloïse}, \mathrm{Elisabeth})$

$\neg\mathrm{sister}(\mathrm{Eliane}, \mathrm{Elisabeth})$

- Adding trigger $\mathrm{sister}(x, y)$ to quantified formula suffices for SMT solver to prove unsatisfiability

Remarks

- Decision procedure for, e.g., expressive arrays, lists [Dross16]
- Mostly efficient (see later evaluation)
- But can easily blow or avoid the right instances
- Requires triggers (human or auto-generated)

Outline

Conflict-based instantiation (**c**) [Reynolds14]

Search for *one* instance of one quantified formula in Q that makes E unsatisfiable

Conflict-based instantiation (c) [Reynolds14]

Search for *one* instance of one quantified formula in Q that makes E unsatisfiable

- $E = \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and $Q = \{\forall x.\, P(x) \vee R(x)\}$

- Since E, $P(b) \vee R(b) \models \bot$, this strategy returns $x \mapsto b$

- Formally

  $\mathbf{c}(E, \forall \bar{x}.\, \varphi)$ Either return σ where $E \models \neg\varphi\sigma$, or return $\emptyset$

c: solving the problem

$$E \models \neg\psi\sigma, \text{ for some } \forall\bar{x}\,\psi \in Q$$

c: solving the problem

$$E \models \neg\psi\sigma, \text{ for some } \forall \bar{x}\, \psi \in Q$$

$$E = \{f(a) = f(b),\, g(b) \neq h(c)\},\, Q = \{\forall xyz.\ f(x) = f(z) \to h(y) = g(z)\}$$

C: solving the problem

$$E \models \neg\psi\sigma, \text{ for some } \forall\bar{x}\,\psi \in Q$$

$E = \{f(a) = f(b),\, g(b) \neq h(c)\},\ Q = \{\forall xyz.\ f(x) = f(z) \to h(y) = g(z)\}$

$$f(a) = f(b) \wedge g(b) \neq h(c) \models (f(x) = f(z) \wedge h(y) \neq g(z))\,\sigma$$

C: solving the problem

$$E \models \neg\psi\sigma, \text{ for some } \forall\bar{x}\,\psi \in Q$$

$E = \{f(a) = f(b),\, g(b) \neq h(c)\},\, Q = \{\forall xyz.\ f(x) = f(z) \rightarrow h(y) = g(z)\}$

$$f(a) = f(b) \land g(b) \neq h(c) \models (f(x) = f(z) \land h(y) \neq g(z))\,\sigma$$

- Each literal in the right hand side restricts σ

c: solving the problem

$$E \models \neg\psi\sigma, \text{ for some } \forall\bar{x}\,\psi \in Q$$

$E = \{f(a) = f(b), g(b) \neq h(c)\}$, $Q = \{\forall xyz.\ f(x) = f(z) \to h(y) = g(z)\}$

$$f(a) = f(b) \wedge g(b) \neq h(c) \models (f(x) = f(z) \wedge h(y) \neq g(z))\,\sigma$$

- Each literal in the right hand side restricts σ
  - $f(x) = f(z)$: either $x = z$ or $x = a \wedge z = b$ or $x = b \wedge z = a$

$\mathbf{C}$: solving the problem

$$E \models \neg\psi\sigma, \text{ for some } \forall\bar{x}\,\psi \in Q$$

$E = \{f(a) = f(b),\, g(b) \neq h(c)\}$, $Q = \{\forall xyz.\ f(x) = f(z) \rightarrow h(y) = g(z)\}$

$$f(a) = f(b) \wedge g(b) \neq h(c) \models (f(x) = f(z) \wedge h(y) \neq g(z))\,\sigma$$

- Each literal in the right hand side restricts σ
  - $f(x) = f(z)$: either $x = z$ or $x = a \wedge z = b$ or $x = b \wedge z = a$
  - $h(y) \neq g(z)$: $y = c \wedge z = b$

C: solving the problem

$$E \models \neg\psi\sigma, \text{ for some } \forall\bar{x}\,\psi \in Q$$

$E = \{f(a) = f(b),\, g(b) \neq h(c)\},\ Q = \{\forall xyz.\ f(x) = f(z) \rightarrow h(y) = g(z)\}$

$$f(a) = f(b) \wedge g(b) \neq h(c) \models (f(x) = f(z) \wedge h(y) \neq g(z))\,\sigma$$

- Each literal in the right hand side restricts σ
  - $f(x) = f(z)$: either $\underline{x = z}$ or $x = a \wedge z = b$ or $x = b \wedge z = a$
  - $h(y) \neq g(z)$: $\underline{y = c \wedge z = b}$

$$\sigma = \{x \mapsto b,\ y \mapsto c,\ z \mapsto b\}$$

c: solving the problem

$$E \models \neg\psi\sigma, \text{ for some } \forall\bar{x}\,\psi \in Q$$

$E = \{f(a) = f(b),\, g(b) \neq h(c)\}$, $Q = \{\forall xyz.\ f(x) = f(z) \rightarrow h(y) = g(z)\}$

$$f(a) = f(b) \wedge g(b) \neq h(c) \models (f(x) = f(z) \wedge h(y) \neq g(z))\,\sigma$$

- Each literal in the right hand side restricts σ
  - $f(x) = f(z)$: either $x = z$ or $\underline{x = a \wedge z = b}$ or $x = b \wedge z = a$
  - $h(y) \neq g(z)$: $\underline{y = c \wedge z = b}$

$$\sigma = \{x \mapsto b,\, y \mapsto c,\, z \mapsto b\}$$

or

$$\sigma = \{x \mapsto a,\, y \mapsto c,\, z \mapsto b\}$$

c: solving the problem

$$E \models \neg\psi\sigma, \text{ for some } \forall\bar{x}\,\psi \in Q$$

$E = \{f(a) = f(b),\, g(b) \neq h(c)\},\, Q = \{\forall xyz.\ f(x) = f(z) \rightarrow h(y) = g(z)\}$

$$f(a) = f(b) \wedge g(b) \neq h(c) \models (f(x) = f(z) \wedge h(y) \neq g(z))\,\sigma$$

- Each literal in the right hand side restricts σ
  - $f(x) = f(z)$: either $x = z$ or $x = a \wedge z = b$ or $\underline{x = b \wedge z = a}$
  - $h(y) \neq g(z)$: $\underline{y = c \wedge z = b}$

$$\sigma = \{x \mapsto b,\ y \mapsto c,\ z \mapsto b\}$$

or

$$\sigma = \{x \mapsto a,\ y \mapsto c,\ z \mapsto b\}$$

c: solving the problem with E-ground (dis)unification

Given conjunctive sets of equality literals E and L, with E ground, find substitution σ s.t. $E \models L\sigma$

c: solving the problem with E-ground (dis)unification

Given conjunctive sets of equality literals E and L, with E ground, find substitution σ s.t. $E \models L\sigma$

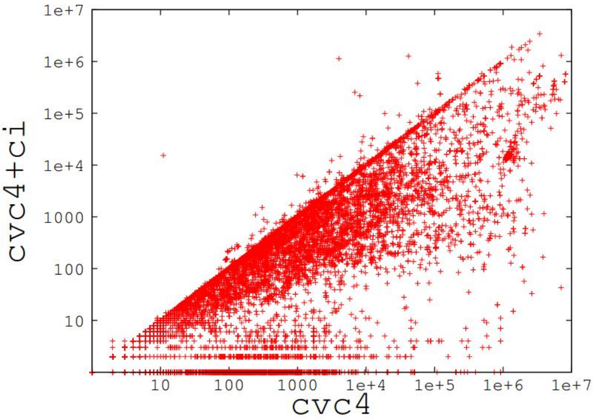
- Variant of classic (non-simultaneous) rigid E-unification

c: solving the problem with E-ground (dis)unification

Given conjunctive sets of equality literals E and L, with E ground, find substitution σ s.t. $E \models L\sigma$

- Variant of classic (non-simultaneous) rigid E-unification
- NP-complete
  - NP: solutions can be restricted to ground terms in $E \cup L$
  - NP-hard: reduction of 3-SAT
- CCFV: congruence closure with free variables [Barbosa17]
  - sound, complete and terminating calculus for solving E-ground (dis)unification
  - goal oriented
  - efficient in practice

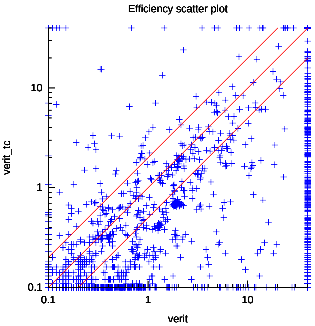
c evaluation (1/2) [Reynolds14]

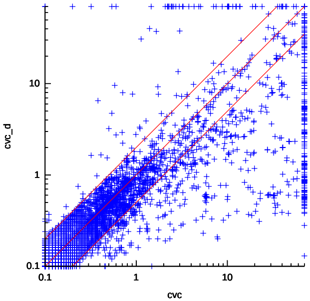


Reported number of instances.

- Evaluation on SMT-LIB, TPTP, Isabelle benchmarks
- Using conflict-based instantiation (cvc4+ci), require an order of magnitude fewer instances to prove unsatisfiability w.r.t. E-matching alone

c evaluation (2/2) [Barbosa17]





veriT: + 800 out of $1\,785$ unsolved problems

CVC4: + 200 out of 745 unsolved problems

* experiments in the "UF", "UFLIA", "UFLRA" and "UFIDL" categories of SMT-LIB, which have $10\,495$ benchmarks annotated as *unsatisfiable*, with 30s timeout.

Outline

Model-based instantiation/MBQI (**m**) [Ge09]

Build a candidate model for $E \cup Q$ and instantiate with counter-examples from model checking

Model-based instantiation/MBQI ($\mathbf{m}$) [Ge09]

Build a candidate model for $E \cup Q$ and instantiate with counter-examples from model checking

- $E = \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and $Q = \{\forall x.\, P(x) \vee R(x)\}$
- Assume that $P^{\mathcal{M}} = \lambda x.\, \text{ite}(x = c,\, \top,\, \bot)$ and $R^{\mathcal{M}} = \lambda x.\, \bot$
- Since $\mathcal{M} \models \neg\,(P(a) \vee R(a))$, this strategy may return $x \mapsto a$
- Formally

$\mathbf{m}(E,\, \forall \bar{x}.\, \varphi)$
1. Construct a model $\mathcal{M}$ for E
2. Return $\bar{x} \mapsto \bar{t}$ where $\bar{t} \in \mathcal{T}(E)$ and $\mathcal{M} \models \neg\varphi[\bar{x}/\bar{t}]$, or $\emptyset$ if none exists

Outline

Why can't we directly use Herbrand instantiation?

Theorem (Herbrand)

A finite set of Skolem formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of Herbrand instances

Why can't we directly use Herbrand instantiation?

Theorem (Herbrand)

A finite set of Skolem formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of Herbrand instances

- The earliest theorem provers relied on *Herbrand instantiation*
  - Instantiate with all possible terms in the language
- Enumerating all instances is unfeasible in practice!
- Enumerative instantiation was then discarded

Why can't we directly use Herbrand instantiation?

Theorem (Herbrand)

A finite set of Skolem formulas is unsatisfiable if and only if there exists a finite unsatisfiable set of Herbrand instances

- The earliest theorem provers relied on *Herbrand instantiation*
  - Instantiate with all possible terms in the language
- Enumerating all instances is unfeasible in practice!
- Enumerative instantiation was then discarded

Revisiting enumerative instantiation with benefits:

- strengthening of Herbrand theorem
- efficient implementation techniques

Theorem (Strengthened Herbrand)

If R is a (possibly infinite) set of instances of Q closed under Q-instantiation w.r.t. itself and if $E \cup R$ is satisfiable, then $E \cup Q$ is satisfiable.

Theorem (Strengthened Herbrand)

If there exists an infinite sequence of finite satisfiable sets of ground literals E_i and of finite sets of ground instances Q_i of Q such that

- $Q_i = \left\{ \varphi\sigma \;\middle|\; \forall \bar{x}.\, \varphi \in Q,\, \mathsf{dom}(\sigma) = \{\bar{x}\} \wedge \mathsf{ran}(\sigma) \subseteq \mathcal{T}(E_i) \right\}$*;*
- $E_0 = E,\ E_{i+1} \models E_i \cup Q_i$*;*

then $E \cup Q$ is satisfiable in the empty theory with equality

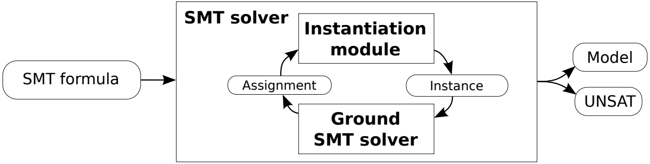
Theorem (Strengthened Herbrand)

If there exists an infinite sequence of finite satisfiable sets of ground literals E_i and of finite sets of ground instances Q_i of Q such that

- $Q_i = \{\varphi\sigma \mid \forall \bar{x}.\, \varphi \in Q, \mathsf{dom}(\sigma) = \{\bar{x}\} \wedge \mathsf{ran}(\sigma) \subseteq \mathcal{T}(E_i)\}$;
- $E_0 = E$, $E_{i+1} \models E_i \cup Q_i$;

then $E \cup Q$ is satisfiable in the empty theory with equality

Direct application to



- Ground solver enumerates assignments $E \cup Q$
- Instantiation module generates instances of Q

Enumerative instantiation ($\mathbf{u}$) [Reynolds18]

$\mathbf{u}(E, \forall \bar{x}.\, \varphi)$

1. Choose an ordering $\preceq$ on tuples of ground terms
2. Return $\bar{x} \mapsto \bar{t}$ where $\bar{t}$ is a minimal tuple of terms w.r.t $\preceq$, such that $\bar{t} \in \mathcal{T}(E)$ and $E \not\models \varphi[\bar{x}/\bar{t}]$, or $\emptyset$ if none exist

- $E = \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and $Q = \{\forall x.\, P(x) \vee R(x)\}$
- $\mathbf{u}$ chooses an ordering on tuples of terms, e.g. $a \prec b \prec c$
- Since $E \not\models P(a) \vee R(a)$, enumerative instantiation returns $x \mapsto a$

u as an alternative for **m**

- Enumerative instantiation plays a similar role to **m**
- It can also serve as a "completeness fallback" to **c** and **e**
- However, **u** has advantages over **m** for UNSAT problems
- And it is significantly simpler to implement
  - no model building
  - no model checking

Example

$$
\begin{array}{rcl}
E & = & \{\neg P(a),\ R(b),\ S(c)\} \\
Q & = & \{\forall x.\ R(x) \vee S(x),\ \forall x.\ \neg R(x) \vee P(x),\ \forall x.\ \neg S(x) \vee P(x)\} \\
M & = & \left\{ \begin{array}{rcl} P^{\mathcal{M}} & = & \lambda x.\ \bot, \\ R^{\mathcal{M}} & = & \lambda x.\ \mathsf{ite}(x = b,\ \top,\ \bot), \\ S^{\mathcal{M}} & = & \lambda x.\ \mathsf{ite}(x = c,\ \top,\ \bot) \end{array} \right\}, \qquad a \prec b \prec c
\end{array}
$$

φ	x s.t. $\mathcal{M} \models \neg\varphi$	x s.t. $E \not\models \varphi$	$\mathbf{m}(E, \forall x.\ \varphi)$	$\mathbf{u}(E, \forall x.\ \varphi)$
$R(x) \vee S(x)$	a	a	$x \mapsto a$	$x \mapsto a$
$\neg R(x) \vee P(x)$	b	a, b, c	$x \mapsto b$	$x \mapsto a$
$\neg S(x) \vee P(x)$	c	a, b, c	$x \mapsto c$	$x \mapsto a$

- **u** instantiates uniformly so that less new terms are introduced
- **m** instantiates depending on how model was built
- **u** directly leads to $E \wedge Q[x/a] \models \bot$

Advanced **u**: restricting enumeration space

- Strengthened Herbrand Theorem allows restriction to $\mathcal{T}(E)$
- *Sort inference* reduces instantiation space by computing more precise sort information
  - $E \cup Q = \{a \neq b,\ f(a) = c\} \cup \{P(f(x))\}$
    - $a, b, c, x : \tau$
    - $f : \tau \to \tau$ and $P : \tau \to$ Bool
  - This is equivalent to $E^s \cup Q^s = \{a_1 \neq b_1,\ f_{12}(a_1) = c_2\} \cup \{P_2(f_{12}(x_1))\}$
    - $a_1, b_1, x_1 : \tau_1$
    - $c_2 : \tau_2$
    - $f_{12} : \tau_1 \to \tau_2$ and $P : \tau_2 \to$ Bool
  - **u** would derive e.g. $x \mapsto c$ for $E \cup Q$, while for $E^s \cup Q^s$ the instantiation $x_1 \mapsto c_2$ is not well-sorted

Advanced $\mathbf{u}$: entailment checks

Two-layered method for checking whether $E \models \varphi[\bar{x}/\bar{t}]$ holds

- cache of instances already derived
- on-the-fly rewriting of $\varphi[\bar{x}/\bar{t}]$ modulo E
  with extension to other theories through theory-specific rewriting

Advanced $\mathbf{u}$: term ordering

Instances are enumerated according to the order

$$(t_1, \ldots, t_n) \prec (s_1, \ldots, s_n) \quad \text{if} \quad \begin{cases} \max_{i=1}^n t_i \prec \max_{i=1}^n s_i, \text{ or} \\ \max_{i=1}^n t_i = \max_{i=1}^n s_i \text{ and} \\ \qquad\qquad (t_1, \ldots, t_n) \prec_{\text{lex}} (s_1, \ldots, s_n) \end{cases}$$

for a given order $\preceq$ on ground terms.

If $a \prec b \prec c$, then

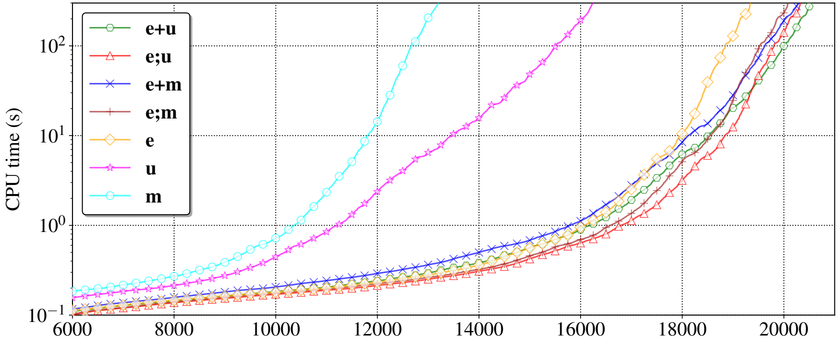
$$(a, a) \prec (a, b) \prec (b, a) \prec (b, b) \prec (a, c) \prec (c, b) \prec (c, c)$$

- instances with c considered only after considering all cases with a and b
- goal is to introduce new terms less often
- order on $\mathcal{T}(E)$ fixed for finite set of terms $t_1 \prec \ldots \prec t_n$
  - instantiate in order with $t_1, \ldots, t_n$
  - then choose new non-congruent term $t \in \mathcal{T}(E)$ and have $t_n \prec t$

Outline

Experimental evaluation (UNSAT)

CVC4 configurations on unsatisfiable benchmarks



- $42\,065$ benchmarks: $14\,731$ TPTP $+$ $27\,334$ SMT-LIB
- **e**$+$**u**: interleave **e** and **u**
- **e**;**u**: apply **e** first, then **u** if it fails
- All CVC4 configurations have **c**; as prefix

Experimental evaluation (SAT)

Library	#	**u**	**e;u**	**e+u**	**e**	**m**	**e;m**	**e+m**
TPTP	14731	471	492	464	17	930	808	829
UF	7293	39	42	42	0	70	69	65
Theories	20041	3	3	3	3	350	267	267
Total	42065	513	537	509	20	1350	1144	1161

Outline

Outline

Conclusion

- Quantifiers in SMT: handled in an ad hoc manner
- Techniques presented here are pure FOL with equality (i.e. not "Modulo Theories")
- Reasonably effective nonetheless

Coarse algorithm

- Skolemize (in a more or less clever way)
- solve ground part of the problem
- eliminate irrelevant information from ground assignment
- conflict-based instantiation
- e-matching/trigger-based instantiation
- model-based instantiation
- enumerative instantiation

Perspectives

- New instantiation techniques
  E.g. currently investigating machine learning
- More convergence with state-of-the-art FOL techniques from saturation theorem proving
- Symbiosis with quantifier elimination for theory reasoning

Unsatisfiability modulo combination of theories...

...cannot be complete (as soon as we mix UF and linear arithmetic), but can we be complete with SMT techniques at least for, e.g., the FOL theory of Presburger extended with UF?
(needs induction however)

Keep in mind, for quantifier handling:

- innovative $\neq$ improving over the best
- innovative $=$ solving what other techniques do not
- best solvers are portfolios

Finding out more about SMT / SMT-LIB

- Andrew Reynolds, VTSA 2017
- Web site of the SMT-LIB initiative:
  `http://www.smtlib.org/`
- Web site of the SMT-COMP:
  `http://www.smtcomp.org/`
- Getting the SMT-LIB input language standard:
  `http://www.smtlib.org/language.shtml`
- Getting some examples of input language:
  `http://www.smtlib.org/examples.shtml`

Outline

Outline

References I

[Althaus09] Ernst Althaus, Evgeny Kruglov, and Christoph Weidenbach.
Superposition modulo linear arithmetic SUP(LA).
In Silvio Ghilardi and Roberto Sebastiani, editors, *Frontiers of Combining Systems (FroCoS)*, volume 5749 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2009.

[Baaz01] Matthias Baaz, Uwe Egly, and Alexander Leitsch.
Normal form transformations.
In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 5, pages 273–333. Elsevier Science B.V., 2001.

[Barbosa17] Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds.
Congruence closure with free variables.
In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 10206 of *Lecture Notes in Computer Science*, pages 214–230. Springer, 2017.

[Baumgartner14] Peter Baumgartner.
Model evolution-based theorem proving.
IEEE Intelligent Systems, 29(1):4–10, 2014.

[Bonacina17] Maria Paola Bonacina and David A. Plaisted.
Semantically-guided goal-sensitive reasoning: Inference system and completeness.
J. Autom. Reasoning, 59(2):165–218, 2017.

References II

[Detlefs05] David Detlefs, Greg Nelson, and James B. Saxe.
Simplify: A Theorem Prover for Program Checking.
J. ACM, 52(3):365–473, 2005.

[Dross16] Claire Dross, Sylvain Conchon, Johannes Kanig, and Andrei Paskevich.
Adding decision procedures to SMT solvers using axioms with triggers.
J. Autom. Reasoning, 56(4):387–457, 2016.

[Ge09] Yeting Ge and Leonardo Mendonça de Moura.
Complete instantiation for quantified formulas in satisfiabiliby modulo theories.
In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification (CAV)*, volume 5643 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2009.

[Halper91] Joseph Y. Halpern.
Presburger arithmetic with unary predicates is Π^1_1 complete.
The Journal of Symbolic Logic, 56(2):637–642, June 1991.

[Korovin13] Konstantin Korovin.
Inst-gen - A modular approach to instantiation-based automated reasoning.
In Andrei Voronkov and Christoph Weidenbach, editors, *Programming Logics - Essays in Memory of Harald Ganzinger*, volume 7797 of *Lecture Notes in Computer Science*, pages 239–270. Springer, 2013.

References III

[Nonnengart01] Andreas Nonnengart and Christoph Weidenbach.
Computing small clause normal forms.
In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science B.V., 2001.

[Reynolds18] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine.
Revisiting enumerative instantiation.
In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 10806 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 2018.

[Reynolds14] Andrew Reynolds, Cesare Tinelli, and Leonardo Mendonça de Moura.
Finding conflicting instances of quantified formulas in SMT.
In *Formal Methods In Computer-Aided Design (FMCAD)*, pages 195–202. IEEE, 2014.

[Reynolds13] Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstić, Morgan Deters, and Clark Barrett.
Quantifier Instantiation Techniques for Finite Model Finding in SMT.
In MariaPaola Bonacina, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 7898 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2013.

References IV

[Voronkov14] Andrei Voronkov.
AVATAR: the architecture for first-order theorem provers.
In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification (CAV)*, volume 8559 of *Lecture Notes in Computer Science*, pages 696–710. Springer, 2014.