Unification with Abstraction and Theory
Instantiation in Saturation-based Reasoning

Giles Reger¹, Martin Suda², and Andrei Voronkov¹,²,³

¹School of Computer Science, University of Manchester, UK
²TU Wien, Vienna, Austria
³EasyChair

Matryoshka 2018
This is a (slightly) extended version of the talk given at TACAS 2018

Thank you to Martin Suda for preparing the slides
I also stole some from Martin Riener

All mistakes are my own
What is Vampire:

- Automatic Theorem Prover (ATP) for first-order logic
- Main paradigm: superposition calculus + saturation
- Also:
  - efficient term indexing
  - use of incomplete strategies
  - strategy scheduling
  - and theory reasoning
Introduction

What is Vampire:

- Automatic Theorem Prover (ATP) for first-order logic
- Main paradigm: superposition calculus + saturation
- Also:
  - efficient term indexing
  - use of incomplete strategies
  - strategy scheduling
  - and theory reasoning

Reasoning with Theories

- huge application demand:
  - program analysis, software verification, ...
- inherently hard, especially with quantifiers!

Now available!  http://vprover.github.io (License applies)
Competitions

- Regular successful participation at the CASC competition
- Since 2016 also participating in SMT-COMP

But it would be nice to get more ‘real’ benchmarks to demonstrate that these results generalise – SMT-COMP is better than CASC for this. Submit your problems to the libraries (if allowed)!
Two Dimensions of Complexity

∀∃

ATP
theory axioms
...

AVATAR
mod
Theories

DANGER
ZONE

SMT
E-matching
...

gnd

Z/R: +-*/
select/store

Reasoning with Quantifiers and Theories
Reasoning with Quantifiers and Theories

Two Dimensions of Complexity

∀∃

Z/R: +-*/
select/store
gnd

ATP
Two Dimensions of Complexity

∀∃
R: +-*/
select/store
gnd
ATP
SMT
Reasoning with Quantifiers and Theories

Two Dimensions of Complexity

∀∃

ATP

E
SPASS
VAMPIRE
...

CVC4
veriT
Z3
...

SMT
select/store

gnd
Reasoning with Quantifiers and Theories

Two Dimensions of Complexity

∀∃

Z / R: +-*/

select/store
gnd

ATP

SMT

select/store
Two Dimensions of Complexity

∀∃

Z/R: +-*/

E-matching...

select/store

gnd

ATP

SMT
Reasoning with Quantifiers and Theories

Two Dimensions of Complexity

∀∃

ATP
theory axioms
...

SMT
E-matching
...

Z/R: +-*/
select/store
Reasoning with Quantifiers and Theories

Two Dimensions of Complexity

∀∃

ATP
theory axioms...

AVATAR
mod
Theories

SMT
E-matching...

Z/R: +-*/
select/store
Contribution 1: Theory Instantiation Rule

\[ 14x \not\equiv x^2 + 49 \lor p(x) \Rightarrow p(7) \]

by utilising ground SMT solving (current) limitation: complete theories (e.g. arithmetic)

Contribution 2: Unification with Abstraction

extension of unification that introduces theory constraints

\[ p(2x) \text{ against } \neg p(10) \]

= \[ 2x \not\equiv 10 \]

a lazy approach to abstraction new constrains can be often "discharged" by 1.
Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause
Contributions:

1. **Theory Instantiation Rule**
   - Derives a simplifying instance of a non-ground clause:
     \[ 14x \not\equiv x^2 + 49 \lor p(x) \]

2. **Unification with Abstraction**
   - Extension of unification that introduces theory constraints:
     \[ p(2x) \equiv \neg p(10) \Rightarrow 2x \not\equiv 10 \]
Contribution 1: Theory Instantiation Rule

- derivs a simplifying instance of a non-ground clause

\[ 14x \not\equiv x^2 + 49 \lor p(x) \implies p(7) \]
Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause
  
  \[ 14x \not\equiv x^2 + 49 \lor p(x) \implies p(7) \]

- by utilising ground SMT solving
Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause
  \[ 14x \not\approx x^2 + 49 \lor p(x) \implies p(7) \]
- by utilising ground SMT solving
- (current) limitation: complete theories (e.g. arithmetic)
Contribution 1: Theory Instantiation Rule
- derives a simplifying instance of a non-ground clause

\[ 14x \not\equiv x^2 + 49 \lor p(x) \implies p(7) \]
- by utilising ground SMT solving
- (current) limitation: complete theories (e.g. arithmetic)

Contribution 2: Unification with Abstraction
- extension of unification that introduces theory constraints
Contribution 1: Theory Instantiation Rule
- derives a simplifying instance of a \textit{non-ground} clause

\[ 14x \not\equiv x^2 + 49 \lor p(x) \implies p(7) \]

- by utilising \textit{ground} SMT solving
- \textit{(current) limitation: complete theories (e.g. arithmetic)}

Contribution 2: Unification with Abstraction
- extension of unification that introduces theory constraints
- \( p(2x) \) against \( \neg p(10) \)
Our Paper in One Slide

Contribution 1: Theory Instantiation Rule
- derives a simplifying instance of a non-ground clause
  \[ 14x \not\equiv x^2 + 49 \lor p(x) \implies p(7) \]
- by utilising ground SMT solving
- (current) limitation: complete theories (e.g. arithmetic)

Contribution 2: Unification with Abstraction
- extension of unification that introduces theory constraints
- \( p(2x) \) against \( \neg p(10) \implies 2x \not\equiv 10 \)
Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a \textit{non-ground} clause

\[
14x \not\approx x^2 + 49 \lor p(x) \implies p(7)
\]

- by utilising \textit{ground} SMT solving

- (current) limitation: complete theories (e.g. arithmetic)

Contribution 2: Unification with Abstraction

- extension of unification that introduces theory constraints

\[
p(2x) \text{ against } \neg p(10) \implies 2x \not\approx 10
\]

- a lazy approach to abstraction
Contribution 1: Theory Instantiation Rule
- derives a simplifying instance of a non-ground clause
  \[ 14x \not\equiv x^2 + 49 \lor p(x) \Rightarrow p(7) \]
- by utilising ground SMT solving
- (current) limitation: complete theories (e.g. arithmetic)

Contribution 2: Unification with Abstraction
- extension of unification that introduces theory constraints
- \( p(2x) \) against \( \neg p(10) \Rightarrow 2x \not\equiv 10 \)
- a lazy approach to abstraction
- new constrains can be often “discharged” by 1.
Outline

1. A Brief Introduction to Saturation-Based Proving
2. Previous Methods for Theory Reasoning in Vampire
3. Theory Instantiation and Unification with Abstraction
4. Experimental Results
5. Ongoing and Future Work
Standard form of the input:

\[ F := (\text{Axiom}_1 \land \ldots \land \text{Axiom}_n) \rightarrow \text{Conjecture} \]
Standard form of the input:

\[ F := (Axiom_1 \land \ldots \land Axiom_n) \rightarrow Conjecture \]

1. Negate \( F \) to seek a refutation:

\[ \neg F := Axiom_1 \land \ldots \land Axiom_n \land \neg Conjecture \]
Standard form of the input:

\[ F := (Axiom_1 \land \ldots \land Axiom_n) \rightarrow Conjecture \]

1. Negate F to seek a refutation:

\[ \neg F := Axiom_1 \land \ldots \land Axiom_n \land \neg Conjecture \]

2. Preprocess and transform \( \neg F \) to clause normal form (CNF)

\[ S := \{ C_1, \ldots, C_n \} \]
Standard form of the input:

\[ F := (Axiom_1 \land \ldots \land Axiom_n) \rightarrow Conjecture \]

1. Negate F to seek a refutation:

\[ \neg F := Axiom_1 \land \ldots \land Axiom_n \land \neg Conjecture \]

2. Preprocess and transform \( \neg F \) to clause normal form (CNF)

\[ S := \{ C_1, \ldots, C_n \} \]

3. saturate \( S \) with respect to the superposition calculus

aiming to derive the obvious contradiction \( \bot \)
Saturation = fixed-point computation

Given Clause Algorithm:

- set of active clauses is stored in indexing structures
- passive works like a priority queue
- the process is “explosive” in nature
Controlling the Growth of the Search Space

Superposition rule

\[
\frac{\ld \approx r \lor C_1 \quad L[s]_p \lor C_2}{(L[r]_p \lor C_1 \lor C_2)\theta} \quad \text{or} \quad \frac{\ld \approx r \lor C_1 \quad t[s]_p \otimes t' \lor C_2}{(t[r]_p \otimes t' \lor C_1 \lor C_2)\theta},
\]

where \( \theta = \text{mgu}(l, s) \) and \( r\theta \not\supset l\theta \) and, for the left rule \( L[s] \) is not an equality literal, and for the right rule \( \otimes \) stands either for \( \approx \) or \( \not\approx \) and \( t'\theta \not\supset t[s]\theta \)
Controlling the Growth of the Search Space

Superposition rule

\[
\frac{l \simeq r \lor C_1 \quad L[s]_p \lor C_2}{(L[r]_p \lor C_1 \lor C_2)\theta} \quad \text{or} \quad \frac{l \simeq r \lor C_1 \quad t[s]_p \otimes t' \lor C_2}{(t[r]_p \otimes t' \lor C_1 \lor C_2)\theta},
\]

where \(\theta = \text{mgu}(l, s)\) and \(r\theta \nsubseteq l\theta\) and, for the left rule \(L[s] \) is not an equality literal, and for the right rule \(\otimes\) stands either for \(\simeq\) or \(\nle\) and \(t'\theta \nleq t[s]\theta\).

Saturation up to Redundancy

- redundant clauses can be safely removed
- subsumption - an example reduction:

remove \(C\) in the presence of \(D\) such that \(D\sigma \subset C\)
Controlling the Growth of the Search Space

Superposition rule

\[
\frac{l \simeq r \lor C_1 \ \ \color{red}{L[s]_p} \lor C_2}{(L[r]_p \lor C_1 \lor C_2)\theta} \quad \text{or} \quad \frac{l \simeq r \lor C_1 \ \ \color{red}{t[s]_p \otimes t'} \lor C_2}{(t[r]_p \otimes t' \lor C_1 \lor C_2)\theta},
\]

where \( \theta = \text{mgu}(l, s) \) and \( r\theta \not\geq l\theta \) and, for the left rule \( L[s] \) is not an equality literal, and for the right rule \( \otimes \) stands either for \( \simeq \) or \( \not\simeq \) and \( t'\theta \not\geq t[s]\theta \)

Saturation up to Redundancy

- redundant clauses can be safely removed
- subsumption - an example reduction:

\[
\text{remove } C \text{ in the presence of } D \text{ such that } D\sigma \subset C
\]

Completeness considerations
Outline

1. A Brief Introduction to Saturation-Based Proving
2. Previous Methods for Theory Reasoning in Vampire
3. Theory Instantiation and Unification with Abstraction
4. Experimental Results
5. Ongoing and Future Work
Basic Support for Theories

- Normalization of interpreted operations, e.g.
  \[ t_1 \geq t_2 \leadsto \lnot(t_1 < t_2) \quad a - b \leadsto a + (-b) \]

- Evaluation of ground interpreted terms, e.g.
  \[ f(1 + 2) \leadsto f(3) \quad f(x + 0) \leadsto f(x) \quad 1 + 2 < 4 \leadsto \text{true} \]

- Balancing interpreted literals, e.g.
  \[ 4 = 2 \times (x + 1) \leadsto (4 \text{ div } 2) - 1 = x \leadsto x = 1 \]

- Interpreted operations treated specially by ordering
  (make interpreted things small, do uninterpreted things first)
Adding Theory Axioms

- $x + (y + z) = (x + y) + z$
- $x + y = y + x$
- $-x = x$
- $x \cdot 0 = 0$
- $x \cdot 1 = x$
- $(x \cdot y) + (x \cdot z) = x \cdot (y + z)$
- $x < y \lor y < x \lor x = y$
- $\neg(x < y) \lor x + z < y + z$
- $x < y \lor y < x + 1$ (for ints)
- $x + 0 = x$
- $-(x + y) = (-x + -y)$
- $x + (-x) = 0$
- $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- $x \cdot y = y \cdot x$
- $x = 0 \lor (y \cdot x)/x = y$ (for reals)

- a handcrafted set
- subsets added based on the signature
- ongoing research on how to tame them [IWIL17]
The AVATAR architecture [Voronkov 2014]

- modern architecture of first-order theorem provers
- combines saturation with SAT-solving
- efficient realization of the *clause splitting rule*

\[
\forall x, z, w. \ (s(x) \lor \neg r(x, z) \lor \neg q(w))
\]

| share \(x\) and \(z\) | is disjoint |

- “propositional essence” of the problem delegated to SAT solver
The AVATAR architecture [Voronkov 2014]

- modern architecture of first-order theorem provers
- combines saturation with SAT-solving
- efficient realization of the *clause splitting rule*

\[ \forall x, z, w. \ (s(x) \lor \neg r(x, z) \lor \neg q(w)) \]

- *share* \( x \) and \( z \) is disjoint

- “propositional essence” of the problem delegated to SAT solver

AVATAR modulo Theories [Reger et al. 2016]

- use an SMT solver instead of the SAT solver
- sub-problems considered are *ground-theory-consistent*
- implemented in Vampire using Z3
One Slightly Imprecise View of AVATAR

- **Vampire**
  - Incremental Theory Solver for Quantified Formulas

- **SMT Solver**
  - Theory Solver for Arithmetic
  - Theory Solver for BitVectors
  - Theory Solver for Uninterpreted Functions

- **Core**
- **CDCL SAT Solver**

- **Quantifier Instantiation**

...but please remember: Vampire is the boss here!
One Slightly Imprecise View of AVATAR

Vampire

**Incremental** Theory Solver for Quantified Formulas

SMT Solver

- Theory Solver for Arithmetic
- Theory Solver for BitVectors
- Theory Solver for Uninterpreted Functions

Core

- Quantifier Instantiation

CDCL SAT Solver

...but please remember: Vampire is the boss here!
Outline

1. A Brief Introduction to Saturation-Based Proving
2. Previous Methods for Theory Reasoning in Vampire
3. Theory Instantiation and Unification with Abstraction
4. Experimental Results
5. Ongoing and Future Work
Example

Consider the conjecture \((\exists x)(x + x \simeq 2)\) negated and clausified to

\[ x + x \not\simeq 2. \]

It takes Vampire 15 s to solve using theory axioms deriving lemmas such as

\[ x + 1 \simeq y + 1 \lor y + 1 \leq x \lor x + 1 \leq y. \]
Does Vampire Need Instantiation?

Example

Consider the conjecture \((\exists x)(x + x \simeq 2)\) negated and clausified to

\[ x + x \not\simeq 2. \]

It takes Vampire 15 s to solve using theory axioms deriving lemmas such as

\[ x + 1 \simeq y + 1 \lor y + 1 \leq x \lor x + 1 \leq y. \]

Heuristic instantiation would help, but normally any instance of a clause is immediately subsumed by the original!
Example

Consider a problem containing

\[ 14x \neq x^2 + 49 \lor p(x) \]

It takes a long time to derive \( p(7) \) whereas if we had guessed \( x = 7 \) we immediately get

\[ 14 \cdot 7 \neq 7^2 + 49 \lor p(7) \]
Consider a problem containing

\[ 14x \neq x^2 + 49 \lor p(x) \]

It takes a long time to derive \( p(7) \) whereas if we had guessed \( x = 7 \) we immediately get

\[ 14 \cdot 7 \neq 7^2 + 49 \lor p(7) \]

\[ \Downarrow \]

\[ 98 \neq 98 \lor p(7) \]

\( \Rightarrow \) evaluate
Another Example

Example

Consider a problem containing

\[ 14x \neq x^2 + 49 \vee p(x) \]

It takes a long time to derive \( p(7) \) whereas if we had guessed \( x = 7 \) we immediately get

\[ 14 \cdot 7 \neq 7^2 + 49 \vee p(7) \]

\[ \Downarrow \]

\[ 98 \neq 98 \vee p(7) \]

\[ \Downarrow \]

\[ p(7) \]

evaluate

remove trivial inequality

How do we guess \( x = 7 \)?
Another Example

Consider a problem containing

\[ 14x \neq x^2 + 49 \lor p(x) \]

It takes a long time to derive \( p(7) \) whereas if we had guessed \( x = 7 \) we immediately get

\[ 14 \cdot 7 \neq 7^2 + 49 \lor p(7) \]

\[ \Downarrow \]

\[ 98 \neq 98 \lor p(7) \]

\[ \Downarrow \]

\[ p(7) \]

How do we guess \( x = 7 \)?
Instantiation which makes some theory literals immediately false

As an inference rule:

\[ P \lor D \]

\[ D \theta \]

\[ \text{TheoryInst} \]

where

\[ P \]

contains only pure theory literals and

\[ \neg P \theta \]

is valid in

\[ T \]

Implementation:

Collect relevant pure theory literals

\[ L_1, \ldots, L_n \]

Run an SMT solver on the ground

\[ \neg P \mid x = \neg L_1 \land \ldots \land \neg L_n \]

If the SMT solver returns a model, transform it into a substitution

\[ \theta \]

and produce an instance.
Theory Instantiation

Instantiation which makes some theory literals immediately false

As an inference rule

\[
\frac{P \vee D}{D\theta} \quad \text{TheoryInst}
\]

where \( P \) contains only pure theory literals and \( \neg P\theta \) is valid in \( T \)
Theory Instantiation

Instantiation which makes some theory literals immediately false

As an inference rule

\[ \frac{P \lor D}{D \theta} \quad \text{TheoryInst} \]

where \( P \) contains only pure theory literals and \( \neg P \theta \) is valid in \( \mathcal{T} \)

Implementation:

- Collect relevant pure theory literals \( L_1, \ldots, L_n \)
- Run an SMT solver on the ground \( \neg P[x] = \neg L_1 \land \ldots \land \neg L_n \)
- If the SMT solver returns a model, transform it into a substitution \( \theta \) and produce an instance
Theory Instantiation

Instantiation which makes some theory literals immediately false

As an inference rule

\[
\frac{P \lor D}{D\theta} \quad \text{TheoryInst}
\]

where \( P \) contains only pure theory literals and \( \neg P\theta \) is valid in \( T \)

Implementation:

- Collect relevant pure theory literals \( L_1, \ldots, L_n \)
- Run an SMT solver on the ground \( \neg P[x] = \neg L_1 \land \ldots \land \neg L_n \)
- If the SMT solver returns a model, transform it into a substitution \( \theta \) and produce an instance
Suppose we want to resolve

\[ r(14y) \]
\[ \neg r(x^2 + 49) \lor p(x) \]

⇒ No pure literals
Problems with Abstraction

- Suppose we want to resolve
  \[ r(14y) \]
  \[ \neg r(x^2 + 49) \lor p(x) \]
  \[ \Rightarrow \text{No pure literals} \]
- Abstract to
  \[ z \neq 14y \lor r(z) \]
  \[ u \neq x^2 + 49 \lor \neg r(u) \lor p(x) \]

- (We discuss abstraction more later)

- Instantiation undoes abstraction:
  \[ p(1, 5) \]
  \[ \Downarrow \text{abstract} \]
  \[ x \neq 1 \lor y \neq 5 \lor p(x, y) \]
  \[ \Downarrow \text{instantiate} \]
  \[ p(1, 5) \]
Updated Rule

\[ P \lor D \quad \quad \quad \quad \quad \frac{\text{theory instance}}{D \theta} \]

- \( P \theta \) unsatisfiable in the theory
- \( P \) pure
- \( P \) does not contain trivial literals

A literal is trivial if
- Form: \( x \neq t \) (\( x \) not in \( t \))
- Pure (only theory symbols)
- \( x \) only occurs in other trivial literals or other non-pure literals
Example

A possible instance: \( x \not\approx 1 + y \lor p(x, y) \implies p(1, 0) \) vs. the “more general” instance \( p(y + 1, y) \)
Example
A possible instance: \( x \not\equiv 1 + y \lor p(x, y) \implies p(1, 0) \) vs. the “more general” instance \( p(y + 1, y) \)

Example (some literals constrain less/more than others)
\[ (x \not\equiv 0) \rightarrow p(x) \]
Flavours of Theory Instantiation

Example
A possible instance: \( x \not\equiv 1 + y \lor p(x, y) \Rightarrow p(1, 0) \) vs. the “more general” instance \( p(y + 1, y) \)

Example (some literals constrain less/more than others)
\((x \not\equiv 0) \rightarrow p(x)\)

Three options for thi:
- **strong**: Only select strong literals where a literal is strong if it is a negative equality or an interpreted literal
- **overlap**: Select all strong literals and additionally those theory literals whose variables overlap with a strong literal
- **all**: Select all non-trivial pure theory literals
Recall that we collect relevant pure theory literals $L_1, \ldots, L_n$ to run an SMT solver on $T[x] = \neg L_1 \land \ldots \land \neg L_n$

- the negation step involves Skolemization
- the we just translate the terms via Z3 API
Interacting with the SMT solver

Recall that we collect relevant pure theory literals $L_1, \ldots, L_n$ to run an SMT solver on $T[x] = \lnot L_1 \land \ldots \land \lnot L_n$

- the negation step involves Skolemization
- the we just translate the terms via Z3 API

Example (The Division by zero catch!)

The following two clauses are satisfiable:

$$
\frac{1}{x} \not\equiv 0 \lor p(x) \quad \frac{1}{x} \equiv 0 \lor \lnot p(x).
$$
Recall that we collect relevant pure theory literals $L_1, \ldots, L_n$ to run an SMT solver on $T[x] = \neg L_1 \land \ldots \land \neg L_n$

- the negation step involves Skolemization
- the we just translate the terms via Z3 API

**Example (The Division by zero catch!)**

The following two clauses are satisfiable:

$$\frac{1}{x} \not\simeq 0 \lor p(x) \quad 1/x \simeq 0 \lor \neg p(x).$$

However, instances $p(0)$ and $\neg p(0)$ could be obtained by an “unprotected” instantiation rule.
Interacting with the SMT solver

Recall that we collect relevant pure theory literals $L_1, \ldots, L_n$ to run an SMT solver on $T[x] = \neg L_1 \land \ldots \land \neg L_n$

- the negation step involves Skolemization
- the we just translate the terms via Z3 API

Example (The Division by zero catch!)

The following two clauses are satisfiable:

$$1/x \not\equiv 0 \lor p(x) \quad 1/x \equiv 0 \lor \neg p(x).$$

However, instances $p(0)$ and $\neg p(0)$ could be obtained by an “unprotected” instantiation rule.

Evaluation may fail:

- result out of Vampire’s internal range
- result is a proper algebraic number
Recall the abstraction rule

\[ L[t] \lor C \implies x \not\equiv t \lor L[x] \lor C, \]

where \( L \) is a theory literal, \( t \) a non-theory term, and \( x \) fresh.
Recall the abstraction rule

\[ L[t] \lor C \implies x \not\equiv t \lor L[x] \lor C, \]

where \( L \) is a theory literal, \( t \) a non-theory term, and \( x \) fresh.

Example

Consider two clauses

\[ r(14y) \quad \neg r(x^2 + 49) \lor p(x) \]
Recall the abstraction rule

\[ L[t] \lor C \implies x \not\equiv t \lor L[x] \lor C, \]

where \( L \) is a theory literal, \( t \) a non-theory term, and \( x \) fresh.

Example

Consider two clauses

\[
\begin{align*}
  r(14y) & \quad \neg r(x^2 + 49) \lor p(x) \\
\end{align*}
\]

We could fully abstract them to obtain:

\[
\begin{align*}
  r(u) \lor u \not\equiv 14y & \quad \neg r(v) \lor v \not\equiv x^2 + 49 \lor p(x), \\
\end{align*}
\]
Unification with Abstraction

Recall the abstraction rule

\[ L[t] \lor C \implies x \not\equiv t \lor L[x] \lor C, \]

where \( L \) is a theory literal, \( t \) a non-theory term, and \( x \) fresh.

Example

Consider two clauses

\[ r(14y) \quad \neg r(x^2 + 49) \lor p(x) \]

We could fully abstract them to obtain:

\[ r(u) \lor u \not\equiv 14y \quad \neg r(v) \lor v \not\equiv x^2 + 49 \lor p(x), \]

then resolve to get

\[ u \not\equiv 14y \lor u \not\equiv x^2 + 49 \lor p(x) \]
Explicit abstraction may be harmful:

- fully abstracted clauses are typically much longer
- abstraction destroys ground literals
- theory part requires special treatment
Explicit abstraction may be harmful:
- fully abstracted clauses are typically much longer
- abstraction destroys ground literals
- theory part requires special treatment

Instead of full abstraction . . .
- incorporate the abstraction process into unification
- thus abstractions are “on demand” and lazy
- implemented by extending the substitution tree indexing
Explicit abstraction may be harmful:
- fully abstracted clauses are typically much longer
- abstraction destroys ground literals
- theory part requires special treatment

Instead of full abstraction . . .
- incorporate the abstraction process into unification
- thus abstractions are “on demand” and lazy
- implemented by extending the substitution tree indexing

Now the whole Superposition calculus can be extended to use
Unification with Abstraction instead of standard unification
When do we abstract?

Example (do not produce unsatisfiable constraints)

Allowing $p(1)$ and $p(2)$ to unify under the constraint that $1 \simeq 2$ is not useful in any context.
When do we abstract?

Example (do not produce unsatisfiable constraints)

Allowing $p(1)$ and $p(2)$ to unify under the constraint that $1 \approx 2$ is not useful in any context.

Four options to choose from:

- **interpreted_only**: only produce a constraint if the top-level symbol of both terms is a theory-symbol,
- **one_side_interpreted**: only produce a constraint if the top-level symbol of at least one term is a theory symbol,
- **one_side_constant**: as one_side_interpreted but if the other side is uninterpreted it must be a constant,
- **all**: allow all terms of theory sort to unify and produce constraints.
Extend Substitution Trees to generate constraints when two things don’t match

Also need to lookup next node by (interpreted) sort not just head symbol (a bit of book-keeping overhead)

Need to get these constraints to work with how Vampire implements backtracking and variable renaming (was the hardest part)
Outline

1. A Brief Introduction to Saturation-Based Proving
2. Previous Methods for Theory Reasoning in Vampire
3. Theory Instantiation and Unification with Abstraction
4. Experimental Results
5. Ongoing and Future Work
Experiment with Vampire

Comparing New Options:

- uwa, thi, fta
Experiment with Vampire

Comparing New Options:
- uwa, thi, fta

Methodology:
- take hard, but solvable problems
- randomize (problem, other options), sample the sub-cube
Experiment with Vampire

Comparing New Options:
- uwa, thi, fta

Methodology:
- take hard, but solvable problems
- randomize (problem, other options), sample the sub-cube

For this experiment:
- 24 reasonable combinations of option values: fta, uwa, thi
- approx. 100,000 runs in total
## Comparison of Three Options

<table>
<thead>
<tr>
<th>fta</th>
<th>uwa</th>
<th>thi</th>
<th>solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>off</td>
<td>all</td>
<td>252</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>overlap</td>
<td>265</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>strong</td>
<td>266</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
<td>off</td>
<td>276</td>
</tr>
<tr>
<td>off</td>
<td>all</td>
<td>all</td>
<td>333</td>
</tr>
<tr>
<td>off</td>
<td>all</td>
<td>overlap</td>
<td>351</td>
</tr>
<tr>
<td>off</td>
<td>all</td>
<td>strong</td>
<td>354</td>
</tr>
<tr>
<td>off</td>
<td>one_side_interpreted</td>
<td>all</td>
<td>364</td>
</tr>
<tr>
<td>off</td>
<td>all</td>
<td>off</td>
<td>364</td>
</tr>
<tr>
<td>off</td>
<td>one_side_constant</td>
<td>all</td>
<td>374</td>
</tr>
<tr>
<td>off</td>
<td>interpreted_only</td>
<td>all</td>
<td>379</td>
</tr>
<tr>
<td>off</td>
<td>one_side_interpreted</td>
<td>overlap</td>
<td>385</td>
</tr>
<tr>
<td>off</td>
<td>one_side_interpreted</td>
<td>strong</td>
<td>387</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>all</td>
<td>392</td>
</tr>
<tr>
<td>off</td>
<td>one_side_constant</td>
<td>strong</td>
<td>397</td>
</tr>
<tr>
<td>off</td>
<td>one_side_constant</td>
<td>overlap</td>
<td>401</td>
</tr>
<tr>
<td>off</td>
<td>interpreted_only</td>
<td>overlap</td>
<td>407</td>
</tr>
<tr>
<td>off</td>
<td>one_side_interpreted</td>
<td>off</td>
<td>407</td>
</tr>
<tr>
<td>off</td>
<td>interpreted_only</td>
<td>strong</td>
<td>409</td>
</tr>
<tr>
<td>off</td>
<td>one_side_constant</td>
<td>off</td>
<td>417</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>overlap</td>
<td>428</td>
</tr>
<tr>
<td>off</td>
<td>interpreted_only</td>
<td>off</td>
<td>430</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>strong</td>
<td>431</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
<td>450</td>
</tr>
</tbody>
</table>
## Contribution to Strategy Building

<table>
<thead>
<tr>
<th>Logic</th>
<th>SMT-LIB</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logic</td>
<td>New solutions</td>
<td>Uniquely solved</td>
</tr>
<tr>
<td>ALIA</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>LIA</td>
<td>14</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>LRA</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>UFDTLIA</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>UFLIA</td>
<td>28</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>UFNIA</td>
<td>13</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>TPTP</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Category</td>
<td>New solutions</td>
<td>Uniquely solved</td>
</tr>
<tr>
<td>ARI</td>
<td>13</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>NUM</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SWW</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Ongoing and Future Work

More theories
- Currently just implemented for arithmetic
- Currently working on arrays and datatypes
- Higher-order logic as a theory?

Handling uninterpreted symbols

Tightener connection to AVATAR modulo theories
- Incorporate background knowledge about current model

More general theory instantiation
- More than one solution (inequalities)
- All solutions?
- More ‘general’ solutions?
Conclusion

Two new techniques for reasoning with theories and quantifiers

- theory instantiation
- unification with abstraction

Experiment with Vampire: success on previously unsolved problems

Watch this space