Unification with Abstraction and Theory Instantiation in Saturation-based Reasoning

Giles Reger[1], Martin Suda[2], and Andrei Voronkov[1,2,3]

[1]School of Computer Science, University of Manchester, UK
[2]TU Wien, Vienna, Austria
[3]Easychair

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

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

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)

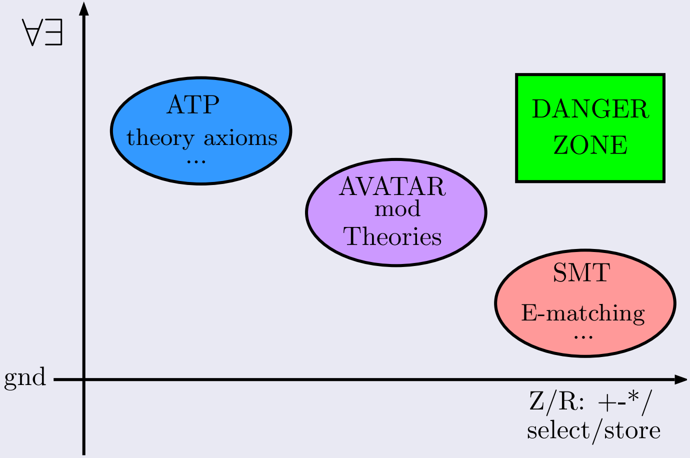
Competitions

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

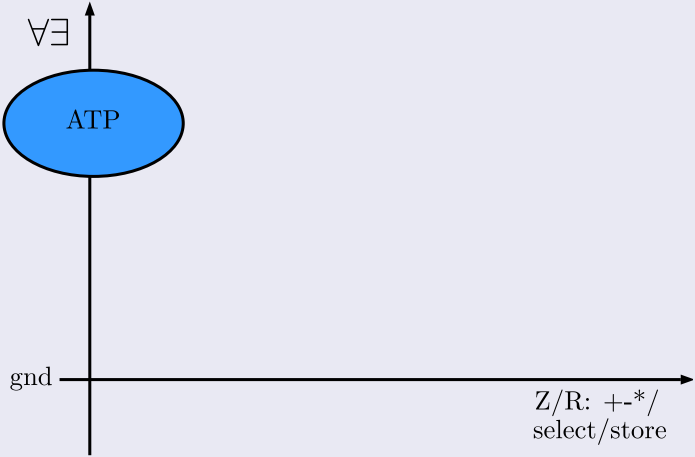
But it would be nice to get more 'real' benchmarks to demonstrate that these results generalise – SMT-COMP is better than CASC for this. Submit your problems to the libraries (if allowed)!

Reasoning with Quantifiers and Theories

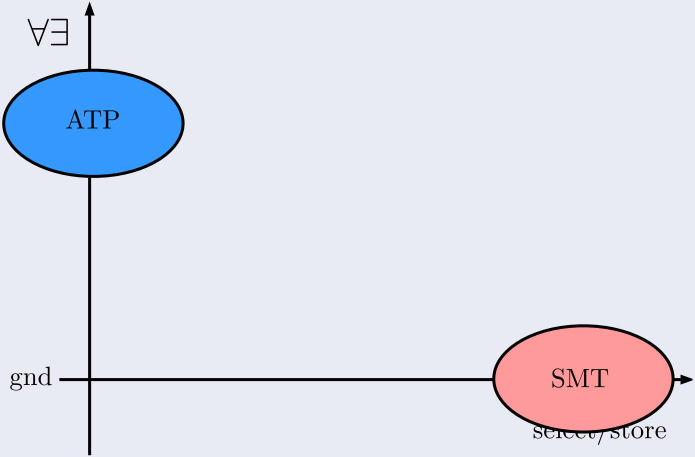
Two Dimensions of Complexity



Two Dimensions of Complexity

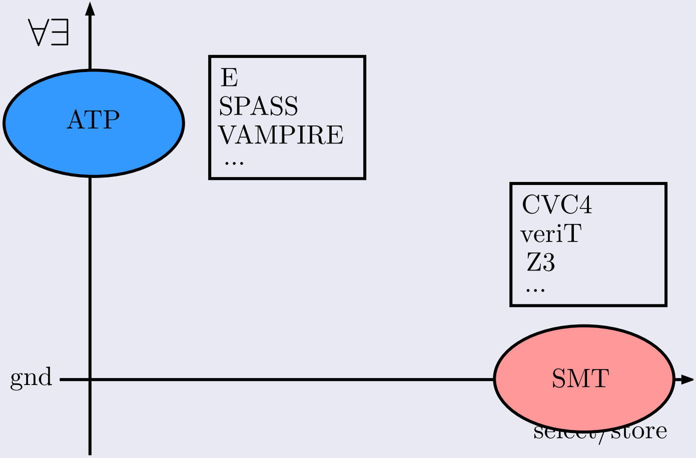


Two Dimensions of Complexity



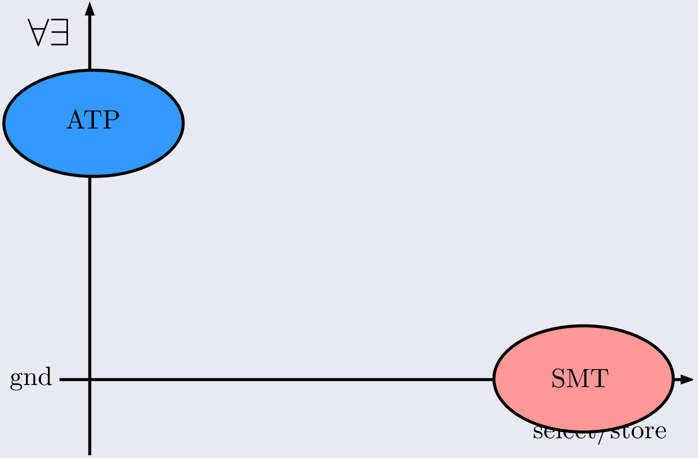
Reasoning with Quantifiers and Theories

Two Dimensions of Complexity

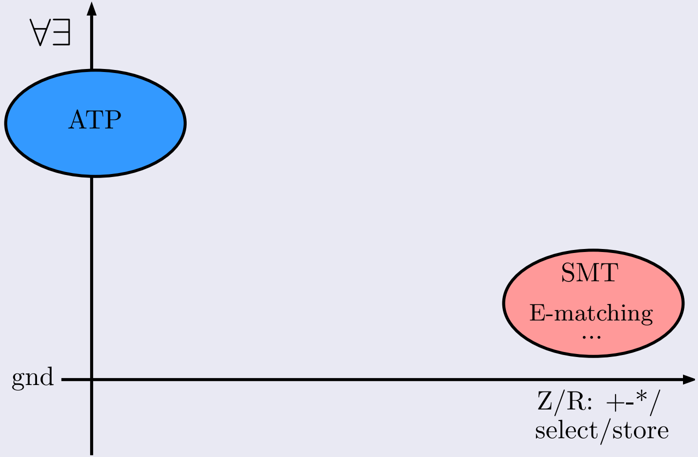


Reasoning with Quantifiers and Theories

Two Dimensions of Complexity

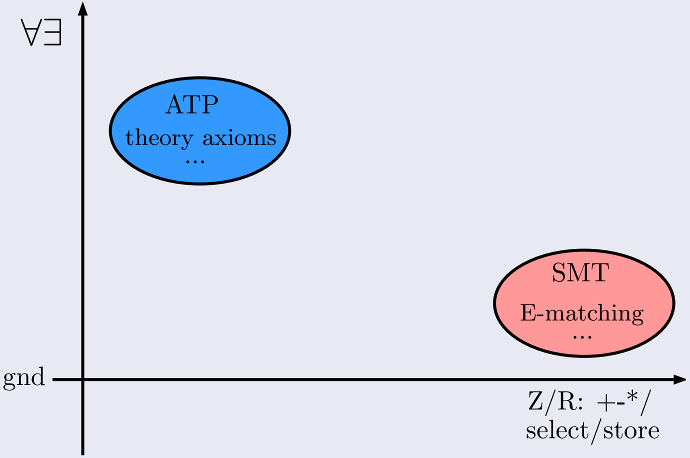


Two Dimensions of Complexity

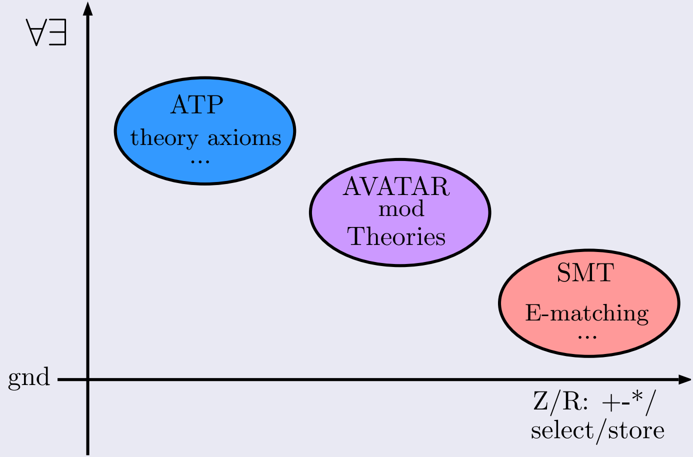


Reasoning with Quantifiers and Theories

Two Dimensions of Complexity



Two Dimensions of Complexity



Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause

Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause

$$14x \not\simeq x^2 + 49 \vee p(x)$$

Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause

$$14x \not\approx x^2 + 49 \vee p(x) \implies p(7)$$

Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause

$$14x \not\approx x^2 + 49 \vee p(x) \implies p(7)$$

- by utilising ground SMT solving

Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause

$$14x \not\approx x^2 + 49 \vee p(x) \implies p(7)$$

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

Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause

$$14x \not\simeq x^2 + 49 \vee p(x) \implies p(7)$$

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

Contribution 2: Unification with Abstraction

- extension of unification that introduces theory constraints

Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause

$$14x \not\approx x^2 + 49 \vee p(x) \implies p(7)$$

- 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 $\neg p(10)$

Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause

$$14x \not\simeq x^2 + 49 \vee p(x) \implies p(7)$$

- 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 $\neg p(10) \implies 2x \not\simeq 10$

Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause

$$14x \not\simeq x^2 + 49 \vee p(x) \implies p(7)$$

- 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 $\neg p(10) \implies 2x \not\simeq 10$
- a lazy approach to abstraction

Our Paper in One Slide

Contribution 1: Theory Instantiation Rule

- derives a simplifying instance of a non-ground clause

$$14x \not\simeq x^2 + 49 \vee p(x) \implies p(7)$$

- 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 $\neg p(10) \implies 2x \not\simeq 10$
- a lazy approach to abstraction
- new constrains can be often “discharged” by 1.

Outline

Theorem Proving Pipeline in One Slide

Standard form of the input:

$$F \quad := \quad (Axiom_1 \wedge \ldots \wedge Axiom_n) \rightarrow Conjecture$$

Theorem Proving Pipeline in One Slide

Standard form of the input:

$$F \quad := \quad (Axiom_1 \wedge \ldots \wedge Axiom_n) \rightarrow Conjecture$$

1. Negate F to seek a refutation:

$$\neg F \quad := \quad Axiom_1 \wedge \ldots \wedge Axiom_n \wedge \neg Conjecture$$

Theorem Proving Pipeline in One Slide

Standard form of the input:

$$F \quad := \quad (Axiom_1 \wedge \ldots \wedge Axiom_n) \rightarrow Conjecture$$

1. Negate F to seek a refutation:

$$\neg F \quad := \quad Axiom_1 \wedge \ldots \wedge Axiom_n \wedge \neg Conjecture$$

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

$$\mathcal{S} \quad := \quad \{C_1, \ldots, C_n\}$$

Theorem Proving Pipeline in One Slide

Standard form of the input:

$$F \quad := \quad (Axiom_1 \wedge \ldots \wedge Axiom_n) \rightarrow Conjecture$$

1. Negate F to seek a refutation:

$$\neg F \quad := \quad Axiom_1 \wedge \ldots \wedge Axiom_n \wedge \neg Conjecture$$

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

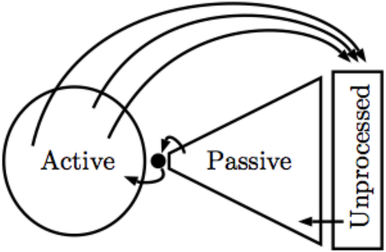
$$\mathcal{S} \quad := \quad \{C_1, \ldots, C_n\}$$

3. saturate $\mathcal{S}$ with respect to the superposition calculus

aiming to derive the obvious contradiction $\perp$

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

Superposition rule

$$\frac{\underline{l \simeq r} \vee C_1 \quad \underline{L[s]_p} \vee C_2}{(L[r]_p \vee C_1 \vee C_2)\theta} \quad \textit{or} \quad \frac{\underline{l \simeq r} \vee C_1 \quad \underline{t[s]_p \otimes t'} \vee C_2}{(t[r]_p \otimes t' \vee C_1 \vee C_2)\theta},$$

where $\theta = \mathrm{mgu}(l, s)$ *and* $r\theta \not\succeq l\theta$ *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* $t'\theta \not\succeq t[s]\theta$

Controlling the Growth of the Search Space

Superposition rule

$$\frac{\underline{l \simeq r} \vee C_1 \quad \underline{L[s]_p} \vee C_2}{(L[r]_p \vee C_1 \vee C_2)\theta} \quad \textit{or} \quad \frac{\underline{l \simeq r} \vee C_1 \quad \underline{t[s]_p \otimes t'} \vee C_2}{(t[r]_p \otimes t' \vee C_1 \vee C_2)\theta},$$

where $\theta = \mathrm{mgu}(l, s)$ and $r\theta \not\succeq l\theta$ 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 $t'\theta \not\succeq t[s]\theta$

Saturation up to Redundancy

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

remove C in the presence of D such that $D\sigma \subset C$

Superposition rule

$$\frac{\underline{l \simeq r} \vee C_1 \quad \underline{L[s]_p} \vee C_2}{(L[r]_p \vee C_1 \vee C_2)\theta} \quad \textit{or} \quad \frac{\underline{l \simeq r} \vee C_1 \quad \underline{t[s]_p \otimes t'} \vee C_2}{(t[r]_p \otimes t' \vee C_1 \vee C_2)\theta},$$

where $\theta = \mathrm{mgu}(l, s)$ *and* $r\theta \not\succeq l\theta$ *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* $t'\theta \not\succeq t[s]\theta$

Saturation up to Redundancy

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

remove C in the presence of D such that $D\sigma \subset C$

Completeness considerations

Outline

Basic Support for Theories

- Normalization of interpreted operations, e.g.

$$t_1 \geq t_2 \rightsquigarrow \neg(t_1 < t_2) \qquad a - b \rightsquigarrow a + (-b)$$

- Evaluation of ground interpreted terms, e.g.

$$f(1+2) \rightsquigarrow f(3) \qquad f(x+0) \rightsquigarrow f(x) \qquad 1+2 < 4 \rightsquigarrow \mathit{true}$$

- Balancing interpreted literals, e.g.

$$4 = 2 \times (x+1) \rightsquigarrow (4 \text{ div } 2) - 1 = x \rightsquigarrow x = 1$$

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

Adding Theory Axioms

$$
\begin{array}{cc}
x + (y + z) = (x + y) + z & x + 0 = x \\
x + y = y + x & -(x + y) = (-x + -y) \\
- - x = x & x + (-x) = 0 \\
x * 0 = 0 & x * (y * z) = (x * y) * z \\
x * 1 = x & x * y = y * x \\
(x * y) + (x * z) = x * (y + z) & \neg(x < y) \vee \neg(y < z) \vee \neg(x < z) \\
x < y \vee y < x \vee x = y & \neg(x < y) \vee \neg(y < x + 1) \\
\neg(x < y) \vee x + z < y + z & \neg(x < x) \\
x < y \vee y < x + 1 \ (\mathit{for\ ints}) & x = 0 \vee (y * x)/x = y \ (\mathit{for\ reals})
\end{array}
$$

- a handcrafted set
- subsets added based on the signature
- ongoing research on how to tame them [IWIL17]

AVATAR modulo Theories (since 2015)

The AVATAR architecture [Voronkov 2014]

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

$$\forall x, z, w.\ \underbrace{s(x) \vee \neg r(x, z)}_{\textit{share } x \textit{ and } z} \vee \underbrace{\neg q(w)}_{\textit{is disjoint}}$$

- "propositional essence" of the problem delegated to SAT solver

AVATAR modulo Theories (since 2015)

The AVATAR architecture [Voronkov 2014]

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

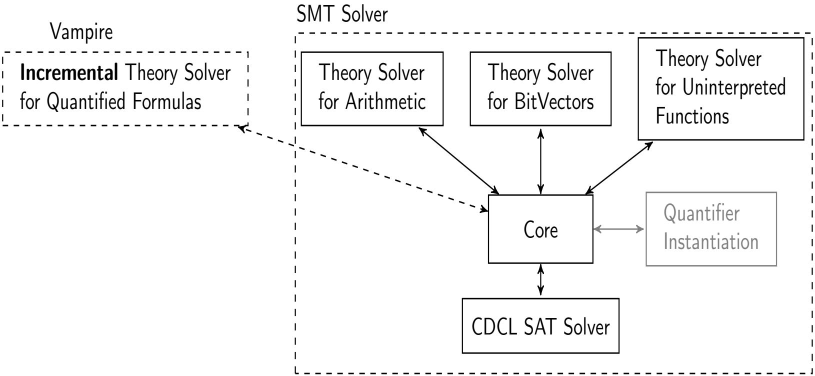
$$\forall x, z, w.\ \underbrace{s(x) \lor \neg r(x, z)}_{\textit{share } x \textit{ and } z} \lor \underbrace{\neg q(w)}_{\textit{is disjoint}}$$

- "propositional essence" of the problem delegated to SAT solver

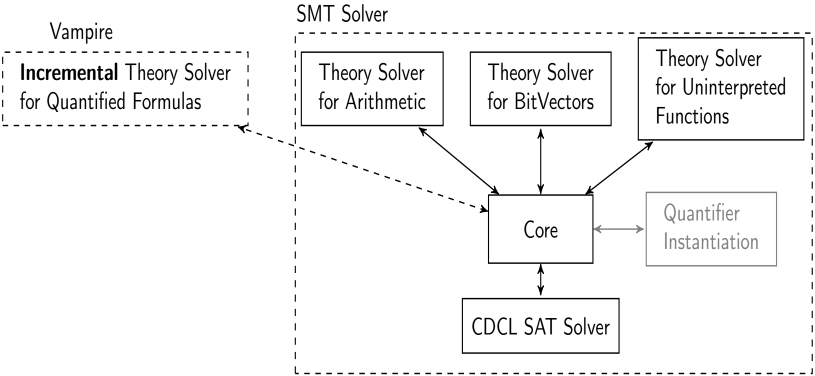
AVATAR modulo Theories [Reger et al. 2016]

- 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



One Slightly Imprecise View of AVATAR



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

Outline

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

$$x + 1 \simeq y + 1 \vee y + 1 \leq x \vee x + 1 \leq y.$$

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

$$x + 1 \simeq y + 1 \vee y + 1 \leq x \vee x + 1 \leq y.$$

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

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

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

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

$$\begin{array}{ll} 14 \cdot 7 \neq 7^2 + 49 \vee p(7) & \\ \rightsquigarrow & \text{evaluate} \\ 98 \neq 98 \vee p(7) & \end{array}$$

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

$$\begin{array}{cc}
14 \cdot 7 \neq 7^2 + 49 \vee p(7) & \\
\rightsquigarrow & \text{evaluate} \\
98 \neq 98 \vee p(7) & \\
\rightsquigarrow & \text{remove trivial inequality} \\
p(7) &
\end{array}$$

How do we guess $x = 7$?

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

$$\begin{array}{cl} 14 \cdot 7 \neq 7^2 + 49 \vee p(7) & \\ \rightsquigarrow & \text{evaluate} \\ 98 \neq 98 \vee p(7) & \\ \rightsquigarrow & \text{remove trivial inequality} \\ p(7) & \end{array}$$

How do we guess $x = 7$?

Theory Instantiation

Instantiation which makes some theory literals immediately false

Theory Instantiation

Instantiation which makes some theory literals immediately false

As an inference rule

$$\frac{P \vee D}{D\theta}\ \mathit{TheoryInst}$$

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

Theory Instantiation

Instantiation which makes some theory literals immediately false

As an inference rule

$$\frac{P \vee D}{D\theta}\ \mathit{TheoryInst}$$

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 $\neg P[\mathbf{x}] = \neg L_1 \wedge \ldots \wedge \neg L_n$
- If the SMT solver returns a model, transform it into a substitution θ and produce an instance

Theory Instantiation

Instantiation which makes some theory literals immediately false

As an inference rule

$$\frac{P \vee D}{D\theta} \; \mathit{TheoryInst}$$

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 $\neg P[\mathbf{x}] = \neg L_1 \wedge \ldots \wedge \neg L_n$
- If the SMT solver returns a model, transform it into a substitution θ and produce an instance

Problems with Abstraction

- Suppose we want to resolve

$$r(14y)$$

$$\neg r(x^2 + 49) \vee p(x)$$

$\Rightarrow$ No pure literals

Problems with Abstraction

- Suppose we want to resolve

$$r(14y)$$
$$\neg r(x^2 + 49) \vee p(x)$$

$\Rightarrow$ No pure literals

- Abstract to

$$z \neq 14y \vee r(z)$$
$$u \neq x^2 + 49 \vee \neg r(u) \vee p(x)$$

- (We discuss abstraction more later)

- Instantiation undoes abstraction:

$$\begin{array}{cl} p(1,5) & \\ \rightsquigarrow & \qquad \text{abstract} \\ x \neq 1 \vee y \neq 5 \vee p(x,y) & \\ \rightsquigarrow & \qquad \text{instantiate} \\ p(1,5) & \end{array}$$

Updated Rule

$$\frac{P \vee D}{D\theta} \text{ theory instance}$$

- $P\theta$ 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

Flavours of Theory Instantiation

Example

A possible instance: $x \not\approx 1 + y \vee p(x, y) \implies p(1, 0)$ vs.
the "more general" instance $p(y + 1, y)$

Flavours of Theory Instantiation

Example

A possible instance: $x \not\approx 1 + y \vee p(x, y) \implies p(1, 0)$ vs.
the "more general" instance $p(y + 1, y)$

Example (some literals constrain less/more than others)

$$(x \not\approx 0) \rightarrow p(x)$$

Flavours of Theory Instantiation

Example

A possible instance: $x \not\simeq 1 + y \vee p(x, y) \implies p(1, 0)$ vs.
the "more general" instance $p(y + 1, y)$

Example (some literals constrain less/more than others)

$$(x \not\simeq 0) \to p(x)$$

Three options for `thi`:

- `strong`: Only select <u>strong</u> 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

Interacting with the SMT solver

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

- 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 $T[\mathrm{x}] = \neg L_1 \wedge \ldots \wedge \neg L_n$

- 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:

$$1/x \not\simeq 0 \vee p(x) \qquad 1/x \simeq 0 \vee \neg p(x).$$

Interacting with the SMT solver

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

- 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:

$$1/x \not\simeq 0 \vee p(x) \qquad 1/x \simeq 0 \vee \neg p(x).$$

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 $T[\mathsf{x}] = \neg L_1 \wedge \ldots \wedge \neg L_n$

- 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:

$$1/x \not\simeq 0 \vee p(x) \qquad 1/x \simeq 0 \vee \neg p(x).$$

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

Unification with Abstraction

Recall the abstraction rule

$$L[t] \vee C \implies x \not\approx t \vee L[x] \vee C,$$

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

Unification with Abstraction

Recall the abstraction rule

$$L[t] \vee C \implies x \not\approx t \vee L[x] \vee C,$$

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

Example

Consider two clauses

$$r(14y) \qquad \neg r(x^2 + 49) \vee p(x)$$

Unification with Abstraction

Recall the abstraction rule

$$L[t] \vee C \implies x \not\simeq t \vee L[x] \vee C,$$

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

Example

Consider two clauses

$$r(14y) \qquad \neg r(x^2 + 49) \vee p(x)$$

We could fully abstract them to obtain:

$$r(u) \vee u \not\simeq 14y \qquad \neg r(v) \vee v \not\simeq x^2 + 49 \vee p(x),$$

Unification with Abstraction

Recall the abstraction rule

$$L[t] \vee C \implies x \not\simeq t \vee L[x] \vee C,$$

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

Example

Consider two clauses

$$r(14y) \qquad \neg r(x^2 + 49) \vee p(x)$$

We could fully abstract them to obtain:

$$r(u) \vee u \not\simeq 14y \qquad \neg r(v) \vee v \not\simeq x^2 + 49 \vee p(x),$$

then resolve to get

$$u \not\simeq 14y \vee u \not\simeq x^2 + 49 \vee p(x)$$

Unification with Abstraction

Explicit abstraction may be harmful:

- fully abstracted clauses are typically much longer
- abstraction destroys ground literals
- theory part requires special treatment

Unification with Abstraction

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

Unification with Abstraction

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 $1 \simeq 2$ is not useful in any context.

Four option 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.

Implementation

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

Comparing New Options:

- `uwa`, `thi`, `fta`

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

fta	uwa	thi	solutions
on	off	all	252
on	off	overlap	265
on	off	strong	266
on	off	off	276
off	all	all	333
off	all	overlap	351
off	all	strong	354
off	one_side_interpreted	all	364
off	all	off	364
off	one_side_constant	all	374
off	interpreted_only	all	379
off	one_side_interpreted	overlap	385
off	one_side_interpreted	strong	387
off	off	all	392
off	one_side_constant	strong	397
off	one_side_constant	overlap	401
off	interpreted_only	overlap	407
off	one_side_interpreted	off	407
off	interpreted_only	strong	409
off	one_side_constant	off	417
off	off	overlap	428
off	interpreted_only	off	430
off	off	strong	431
off	off	off	450

Contribution to Strategy Building

SMT-LIB

Logic	New solutions	Uniquely solved
ALIA	1	0
LIA	14	0
LRA	4	0
UFDTLIA	5	0
UFLIA	28	14
UFNIA	13	4

TPTP

Category	New solutions	Uniquely solved
ARI	13	0
NUM	1	1
SWW	3	1

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

Tighter 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