AVATAR Modulo Theories

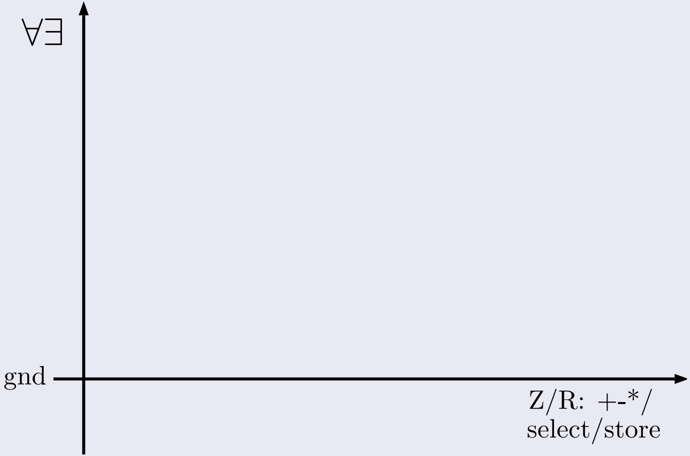
Nikolaj Bjøner[1] Giles Reger[2] Martin Suda[3]
Andrei Voronkov[2,4,5]

[1]Microsoft Research, Redmond, USA
[2]University of Manchester, Manchester, UK
[3]TU Wien, Austria
[4]Chalmers University of Technology, Gothenburg, Sweden
[5]EasyChair

GCAI 2016 – Berlin, September 30, 2016

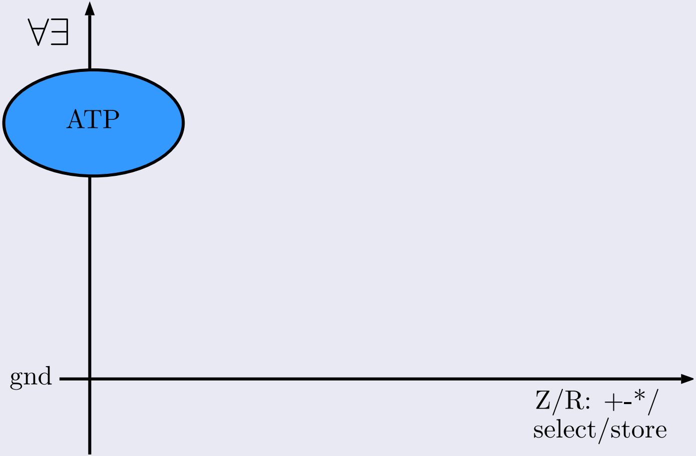
Automated reasoning for program analysis and verification

Reasoning with quantifiers and theories



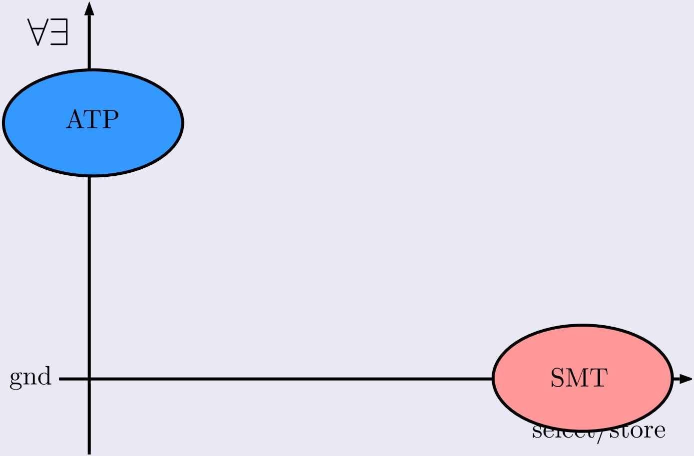
Automated reasoning for program analysis and verification

Reasoning with quantifiers and theories



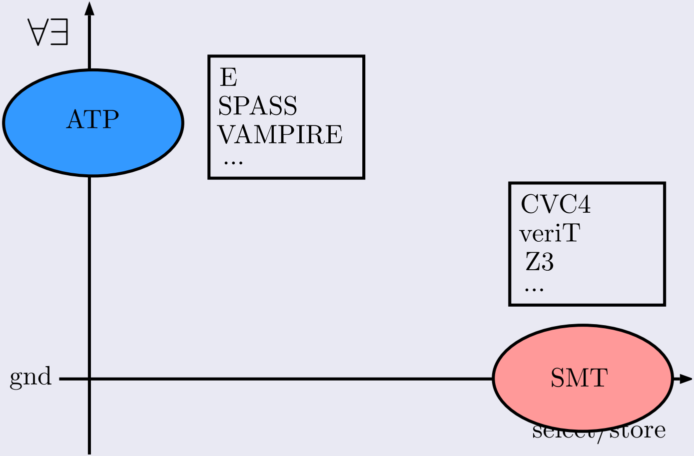
Automated reasoning for program analysis and verification

Reasoning with quantifiers and theories

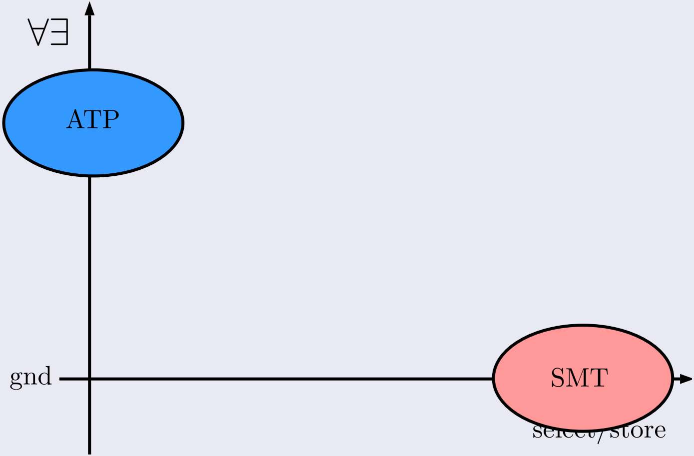


Automated reasoning for program analysis and verification

Reasoning with quantifiers and theories

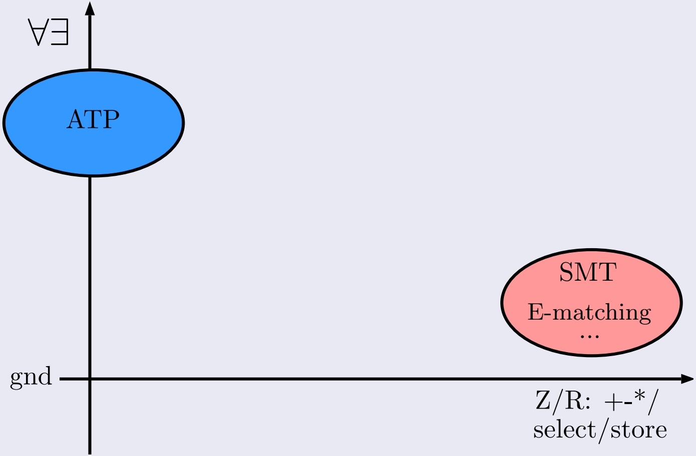


Reasoning with quantifiers and theories



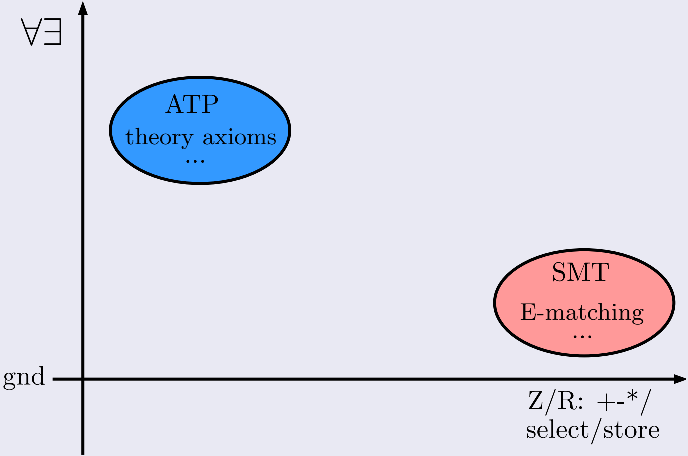
Automated reasoning for program analysis and verification

Reasoning with quantifiers and theories



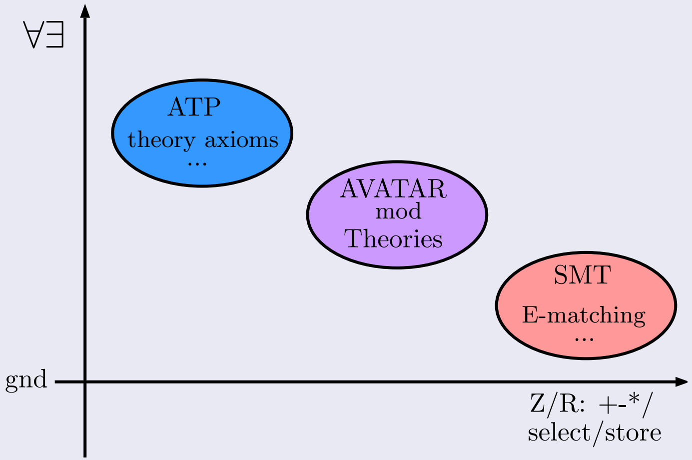
Automated reasoning for program analysis and verification

Reasoning with quantifiers and theories



Automated reasoning for program analysis and verification

Reasoning with quantifiers and theories



What is AVATAR?

What is AVATAR?

AVATAR [Voronkov'14]

- modern architecture of first order theorem provers
- integrates saturation with a SAT solver
- efficient realization of the *clause splitting rule*
- instead of one monolithic proof search
  a sequence of proof searches on (much) smaller sub-problems

- implemented in theorem prover Vampire
- shown highly successful in practice

What is AVATAR?

AVATAR [Voronkov'14]

- modern architecture of first order theorem provers
- integrates saturation with a SAT solver
- efficient realization of the *clause splitting rule*
- instead of one monolithic proof search
  a sequence of proof searches on (much) smaller sub-problems

- implemented in theorem prover Vampire
- shown highly successful in practice

AVATAR modulo Theories

- use an SMT solver instead of the SAT solver
- thus FO solver only considers gnd-theory-consistent sub-problems
- implemented in Vampire using SMT solver Z3

Outline

- Vampire: saturation primer, theory reasoning
- AVATAR architecture: splitting, overview, SAT / SMT abstractions
- Implementation: Z3, implementing abstraction, incompleteness
- Experiments: TPTP results, SMTLIB results
- Conclusion

Outline

- Vampire: saturation primer, theory reasoning
- AVATAR architecture: splitting, overview, SAT / SMT abstractions
- Implementation: Z3, implementing abstraction, incompleteness
- Experiments: TPTP results, SMTLIB results
- Conclusion

Main technology behind Vampire

Saturation-based proving

- turn input formula into an equisatisfiable set of clauses
- inference system: ordered resolution, superposition
- keep adding conclusions
- keep removing redundant clauses
- until the empty clause is derived ...
- ... or a fixed point is reached

Main technology behind Vampire

Saturation-based proving

- turn input formula into an equisatisfiable set of clauses
- inference system: ordered resolution, superposition
- keep adding conclusions
- keep removing redundant clauses
- until the empty clause is derived ...
- ...or a fixed point is reached

Theory reasoning in Vampire (prior to AVATAR)

- add theory axioms: $X + 0 = X$, $X + Y = Y + X$, ...
- evaluate ground terms: $1 + 1 \longrightarrow 2$
- normalization of interpreted operations, i.e. only use $\leq$
- interpreted operations treated specially by ordering

Outline

- Vampire: saturation primer, theory reasoning
- AVATAR architecture: splitting, overview, SAT / SMT abstractions
- Implementation: Z3, implementing abstraction, incompleteness
- Experiments: TPTP results, SMTLIB results
- Conclusion

Clause splitting

Central idea

Let C_1 and C_2 are variable disjoint. Then the clause set

$$S \cup \{C_1 \vee C_2\} \text{ is unsatisfiable}$$

if and only if

$$\text{both } S \cup \{C_1\} \text{ and } S \cup \{C_2\} \text{ are unsatisfiable.}$$

Clause splitting

Central idea

Let C_1 and C_2 are variable disjoint. Then the clause set

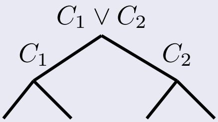
$$S \cup \{C_1 \vee C_2\} \text{ is unsatisfiable}$$

if and only if

$$\text{both } S \cup \{C_1\} \text{ and } S \cup \{C_2\} \text{ are unsatisfiable.}$$

Previous approaches to splitting

- splitting with backtracking [Wei01]



- splitting without backtracking [RV01]

$$p_1 \vee p_2 \quad \neg p_1 \vee C_1 \quad \neg p_2 \vee C_2$$

Splitting as much as possible

Components of a clause

- (non-empty) sub-clauses which do not share a variable
- finest decomposition with this property

Splitting as much as possible

Components of a clause

- (non-empty) sub-clauses which do not share a variable
- finest decomposition with this property

Example (a clause splittable into two components)

$$\forall X, Y, Z \quad p(X, f(Y)) \;\vee\; \neg q(Y) \;\vee\; c \simeq Z$$

Splitting as much as possible

Components of a clause

- (non-empty) sub-clauses which do not share a variable
- finest decomposition with this property

Example (a clause splittable into two components)

$$\forall X, Y, Z \quad p(X, f(Y)) \lor \neg q(Y) \lor c \simeq Z$$

$$\equiv$$

$$\forall X, Y\ [p(X, f(Y)) \lor \neg q(Y)] \quad \lor \quad \forall Z\ c \simeq Z$$

Building blocks of AVATAR

The central idea

- use a SAT / SMT solver to make splitting decisions
- delegate the "base essence" of the given problem to the efficient dedicated solver

Building blocks of AVATAR

The central idea

- use a SAT / SMT solver to make splitting decisions
- delegate the "base essence" of the given problem to the efficient dedicated solver

Enabling ingredient 1: naming of components

- map each component C_i to an abstracted literal $[C_i]_B$
- $[C_i]_B$ can be represented in the base solver

Building blocks of AVATAR

The central idea

- use a SAT / SMT solver to make splitting decisions
- delegate the "base essence" of the given problem to the efficient dedicated solver

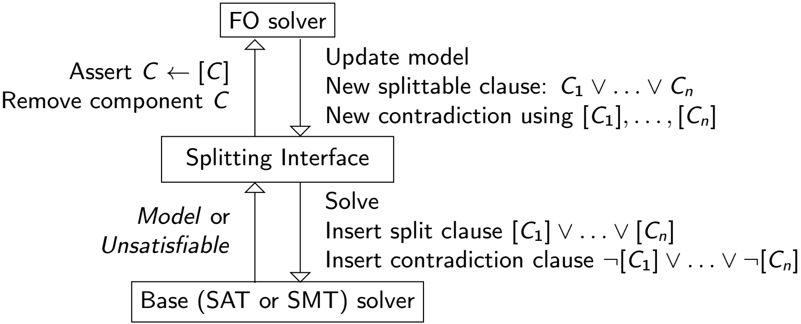
Enabling ingredient 1: naming of components

- map each component C_i to an abstracted literal $[C_i]_B$
- $[C_i]_B$ can be represented in the base solver

Enabling ingredient 2: proving under assumptions

- keep track of dependencies on asserted components
- conditional reductions / deletions
- conditional empty clauses

The AVATAR architecture



Dealing with a splittable clause – SAT solver as base

Example

- splittable clause $C_F = C_1 \vee C_2 \vee C_3$

Dealing with a splittable clause – SAT solver as base

Example

- splittable clause $C_F = C_1 \vee C_2 \vee C_3$
- variables $[C_1]_{SAT}, [C_2]_{SAT}$ and $[C_3]_{SAT}$ introduced in the SAT solver

Dealing with a splittable clause – SAT solver as base

Example

- splittable clause $C_F = C_1 \vee C_2 \vee C_3$
- variables $[C_1]_{SAT}, [C_2]_{SAT}$ and $[C_3]_{SAT}$ introduced in the SAT solver
- clauses $C_P = [C_1]_{SAT} \vee [C_2]_{SAT} \vee [C_3]_{SAT}$ added to SAT solver

Dealing with a splittable clause – SAT solver as base

Example

- splittable clause $C_F = C_1 \vee C_2 \vee C_3$
- variables $[C_1]_{SAT}, [C_2]_{SAT}$ and $[C_3]_{SAT}$ introduced in the SAT solver
- clauses $C_P = [C_1]_{SAT} \vee [C_2]_{SAT} \vee [C_3]_{SAT}$ added to SAT solver
- SAT solver produces:
  $M_1 = \{[C_1]_{SAT} \mapsto 1, [C_2]_{SAT} \mapsto 1, [C_3]_{SAT} \mapsto 0\}$

Dealing with a splittable clause – SAT solver as base

Example

- splittable clause $C_F = C_1 \vee C_2 \vee C_3$
- variables $[C_1]_{SAT}, [C_2]_{SAT}$ and $[C_3]_{SAT}$ introduced in the SAT solver
- clauses $C_P = [C_1]_{SAT} \vee [C_2]_{SAT} \vee [C_3]_{SAT}$ added to SAT solver
- SAT solver produces:
  $M_1 = \{[C_1]_{SAT} \mapsto 1, [C_2]_{SAT} \mapsto 1, [C_3]_{SAT} \mapsto 0\}$
- add $C_1 \leftarrow [C_1]_{SAT}$ and $C_2 \leftarrow [C_2]_{SAT}$ to the FO prover

Proving under assumptions

- example generating inference: resolution

$$\frac{(P \vee C_1) \leftarrow A_1 \qquad (\neg P \vee C_2) \leftarrow A_2}{(C_1 \vee C_2) \leftarrow A_1 \cup A_2}$$

Proving under assumptions

- example generating inference: resolution

$$\frac{(P \vee C_1) \leftarrow A_1 \qquad (\neg P \vee C_2) \leftarrow A_2}{(C_1 \vee C_2) \leftarrow A_1 \cup A_2}$$

- example deleting inference: subsumption

$$\frac{C \leftarrow A_1 \qquad \cancel{D \leftarrow A_2}}{C \leftarrow A_1}$$

provided $C \subset D$.

  - unconditionally deleted when $A_1 \subseteq A_2$
  - conditionally otherwise ($\Rightarrow$ frozen)

Conditional empty clause derived

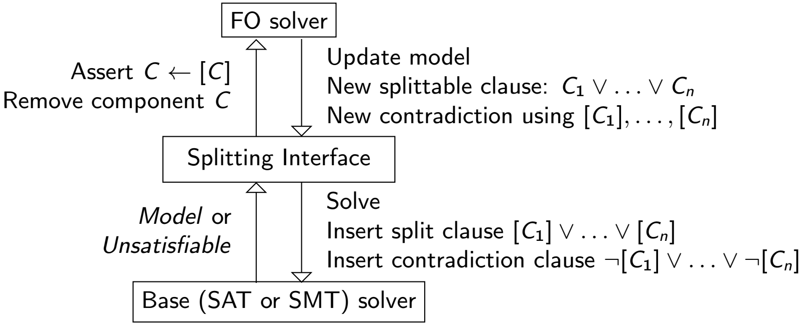
- empty clause $\perp \leftarrow [C'_1], \ldots, [C'_k]$
- add $\neg[C'_1] \vee \ldots \vee \neg[C'_k]$ to the base solver
- forces a new model

Conditional empty clause derived

- empty clause $\bot \leftarrow [C'_1], \ldots, [C'_k]$
- add $\neg[C'_1] \vee \ldots \vee \neg[C'_k]$ to the base solver
- forces a new model

- removing children of components no longer in the model
- (reinserting children of reintroduced components)
- unfreezing conditionally deleted clauses

The AVATAR architecture



The SAT and SMT abstractions

Two kinds of components:

- non-ground: $p(X)$, $s(X, Y) \vee r(Y)$,
- ground (and necessarily unit): $p(f(a))$, $a > f(1 + b)$

The SAT and SMT abstractions

Two kinds of components:

- non-ground: $p(X)$, $s(X, Y) \vee r(Y)$,
- ground (and necessarily unit): $p(f(a))$, $a > f(1 + b)$

SAT

$[.]_{\mathrm{SAT}}$ is an injective mapping from components to prop. variables

- injective up to:
  variable renaming, literal reordering and symmetry of equality
- for ground components: $[\neg L]_{\mathrm{SAT}} = \neg [L]_{\mathrm{SAT}}$

The SAT and SMT abstractions

Two kinds of components:

- non-ground: $p(X)$, $s(X, Y) \vee r(Y)$,
- ground (and necessarily unit): $p(f(a))$, $a > f(1 + b)$

SAT

$[.]_{\mathrm{SAT}}$ is an injective mapping from components to prop. variables

- injective up to:
  variable renaming, literal reordering and symmetry of equality
- for ground components: $[\neg L]_{\mathrm{SAT}} = \neg [L]_{\mathrm{SAT}}$

SMT

- for non-ground, $[.]_{\mathrm{SMT}}$ works the same way as the SAT one
- for ground components: expose the term structure
  - outside the target theory(ies) treat as uninterpreted (UF)

AVATAR only needs the truth value of literals

Outline

- Vampire: saturation primer, theory reasoning
- AVATAR architecture: splitting, overview, SAT / SMT abstractions
- Implementation: Z3, implementing abstraction, incompleteness
- Experiments: TPTP results, SMTLIB results
- Conclusion

Implementation

Z3

- an SMT solver developed at Microsoft research
- supported theories we rely on:
  - linear and non-linear real and integer arithmetic,
  - uninterpreted functions
  - extensional arrays
- Z3's quantifier support is currently not utilized in Vampire

Implementation

Z3

- an SMT solver developed at Microsoft research
- supported theories we rely on:
  - linear and non-linear real and integer arithmetic,
  - uninterpreted functions
  - extensional arrays
- Z3's quantifier support is currently not utilized in Vampire

Implementing the abstraction

- most concepts map easily: sorts, numbers, (un)interpreted symbols
- some TPTP symbols are trickier, e.g. `$round`, and get "defined"
- model extraction: evaluate the respective (boolean) terms

Example (underspecified operations)

$$5/c = 2 \lor c = 0$$

Dealing with incompleteness

For the following query with a, b, c integer constants:

$$(a > 0) \wedge (b > 0) \wedge (c > 0) \wedge (a * a * a) + (b * b * b) = (c * c * c)$$

Z3 returns `unknown`.

But AVATAR cannot progress without a model!

Dealing with incompleteness

For the following query with a, b, c integer constants:

$$(a > 0) \wedge (b > 0) \wedge (c > 0) \wedge (a * a * a) + (b * b * b) = (c * c * c)$$

Z3 returns `unknown`.

But AVATAR cannot progress without a model!

Our recovery solution:

- use a fallback SAT solver
- only query if the SMT solver does not give a concrete answer
- (later on SMT can catch up again)
- the whole situation occurred rarely in our experiments

Outline

- Vampire: saturation primer, theory reasoning
- AVATAR architecture: splitting, overview, SAT / SMT abstractions
- Implementation: Z3, implementing abstraction, incompleteness
- Experiments: TPTP results, SMTLIB results
- Conclusion

Experiments

Two sets of experiments

1. TPTP library benchmarks and the CASC competition entrants
   - all theory benchmarks minus satisfiable ones
   - I/Q/R, L/N
   - time limit 5 minutes per problem
2. SMTLIB benchmarks and SMT solvers
   - all relevant benchmarks: quantifiers and theories (but not bitvectors)
   - exclude those known to be satisfiable
   - A, UF, N/L, IA/RA, DL
   - time limit 30 minutes per problem

Experiments

Two sets of experiments

1. TPTP library benchmarks and the CASC competition entrants
   - all theory benchmarks minus satisfiable ones
   - I/Q/R, L/N
   - time limit 5 minutes per problem
2. SMTLIB benchmarks and SMT solvers
   - all relevant benchmarks: quantifiers and theories (but not bitvectors)
   - exclude those known to be satisfiable
   - A, UF, N/L, IA/RA, DL
   - time limit 30 minutes per problem

Experimental setup

- Starexec compute service
- Intel Xeon 2.4GHz, 128GB of RAM

Results – theory problems from the TPTP library

Division	Size	Easy	Beagle	CVC4	Princess	SPASS+T	VAMPIRE	Z3	ZenonArith	ZenonModulo	Zipperposition
IL	461	213	361	355	359	358	**426**	305	149	101	281
IN	173	45	97	141	129	73	**145**	109	61	43	114
QL	121	34	120	**121**	**121**	118	120	58	116	81	-
QN	38	5	**37**	**37**	35	**37**	**37**	17	36	25	3
RL	116	66	**115**	**115**	**115**	**115**	114	114	112	78	-
RN	39	6	**39**	36	34	37	37	38	37	25	-
IRL+N	9	6	8	8	**9**	5	**9**	**9**	0	0	-
IQRL	8	0	2	2	2	0	2	0	0	0	-
IQRN	3	1	2	**3**	2	2	**3**	2	0	0	-
Total	968	376	787(1)	824	812	745	**899(37)**	652(2)	511	353	398(3)

Results – relevant theory problems from SMTLIB

	Size	CVC4	VAMPIRE	veriT	Z3
ALIA	41	**41**	40	27	**41**
AUFLIA	3	**3**	2	1	2
AUFLIRA	19,914	19,761 (**11**)	**19,777** (9)	19,259	19,751
AUFNIRA	1,491	1,041	**1,085 (45)**	-	1,034 (3)
LIA	380	86 **(21)**	65	**159**	24
LRA	605	**344** (6)	331	78	339
NIA	8	3	4	-	**5 (1)**
NRA	3,813	3,735	3,802 (4)	-	**3,806 (8)**
UFIDL	74	62	**66 (4)**	57	62
UFLIA	12,114	**8,536 (79)**	8,479 (**151**)	6,738	7,815 (3)
UFLRA	20	20	20	20	20
UFNIA	3,351	1,373 (28)	**1,777 (371)**	-	1,235 (12)
Total	41,814	35,390 (145)	**35,448 (584)**	27,844	34,386 (27)

Conclusion

Summary

- extended the AVATAR architecture to reason modulo theories
  - replace the SAT solver by an SMT solver
  - use a different abstraction
  - technical complications overcome
- implemented in Vampire with SMT solver Z3
- encouraging experimental results

Conclusion

Summary

- extended the AVATAR architecture to reason modulo theories
  - replace the SAT solver by an SMT solver
  - use a different abstraction
  - technical complications overcome
- implemented in Vampire with SMT solver Z3
- encouraging experimental results

Future work

- extract more information from the SMT solver
- enable quantifier reasoning in Z3
- E-matching hints from the FO part?