Pretending to be an SMT Solver with Vampire (and How We Do Instantiation)

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

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

SMT 2017 – Heidelberg, July 22, 2017

Introducing Vampire

- Automatic Theorem Prover (ATP) for first-order logic
- Main paradigm: superposition calculus + saturation
- a.k.f.: indexing, incomplete strategies, strategy scheduling

Introducing Vampire

- Automatic Theorem Prover (ATP) for first-order logic
- Main paradigm: superposition calculus + saturation
- a.k.f.: indexing, incomplete strategies, strategy scheduling

Introducing Vampire

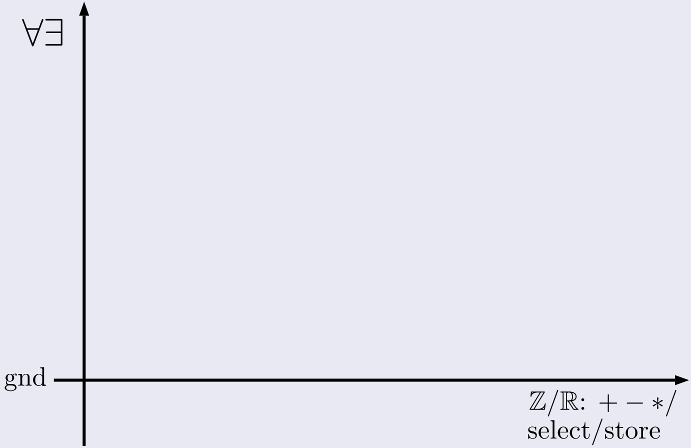
- Automatic Theorem Prover (ATP) for first-order logic
- Main paradigm: superposition calculus + saturation
- a.k.f.: indexing, incomplete strategies, strategy scheduling

Reasoning with Theories

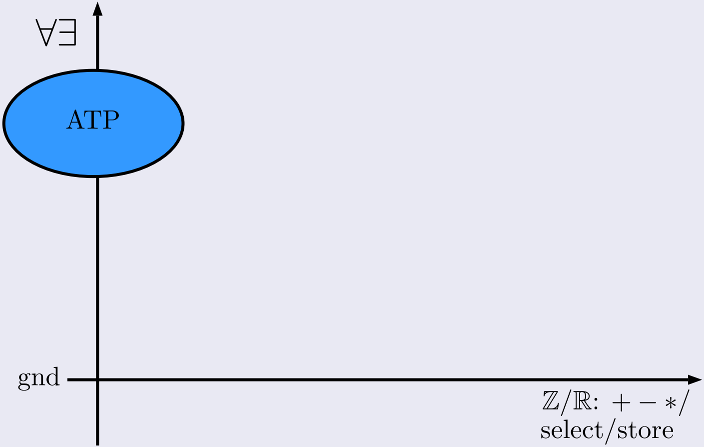
- since 2010: progressively adding support for theories
- since 2016: participating in SMT-COMP

Reasoning with quantifiers and theories

Two Dimensions of Complexity

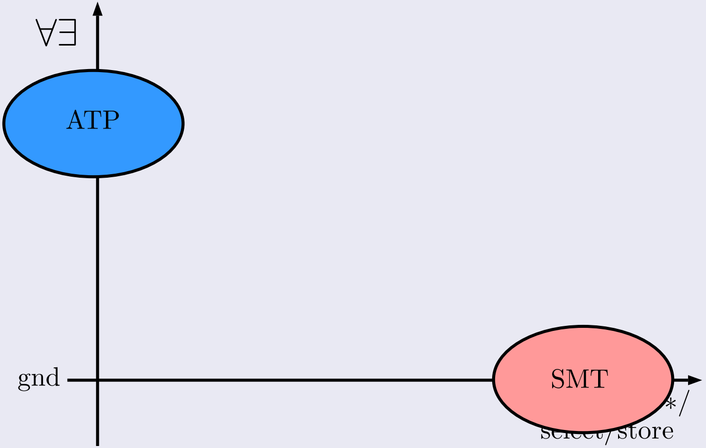


Two Dimensions of Complexity



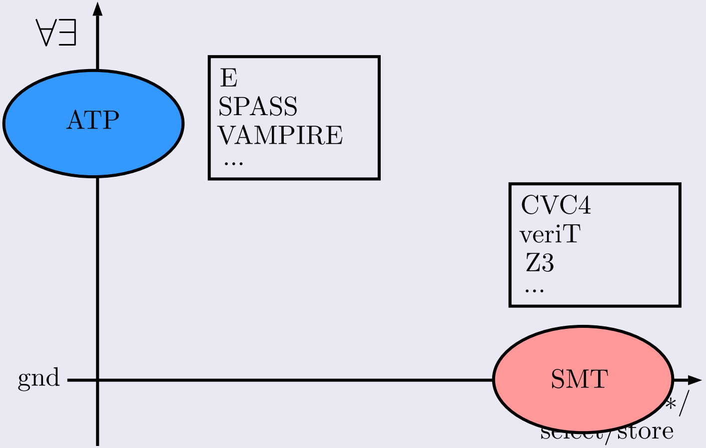
Reasoning with quantifiers and theories

Two Dimensions of Complexity



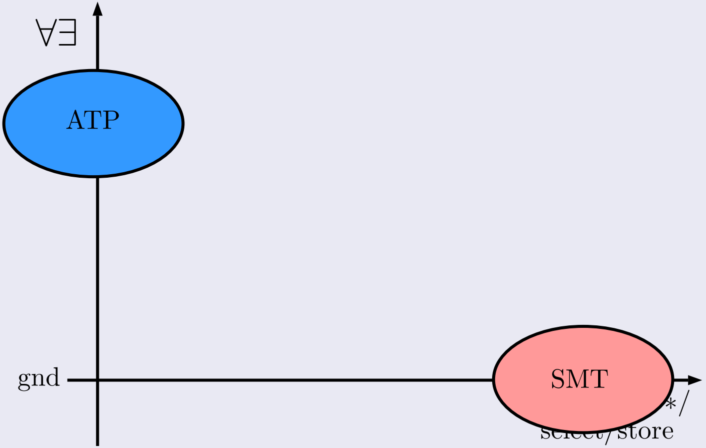
Reasoning with quantifiers and theories

Two Dimensions of Complexity

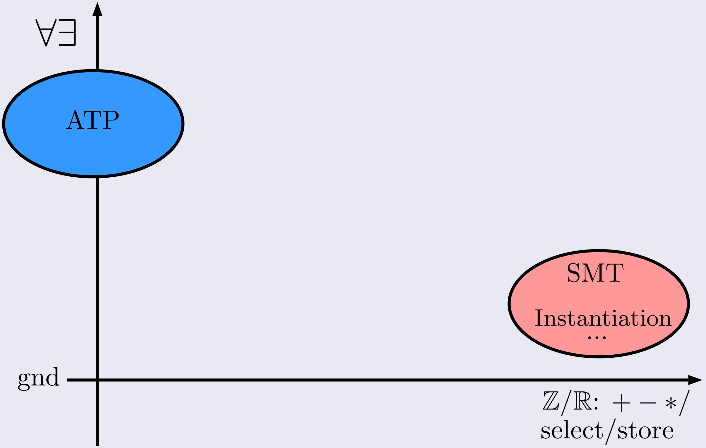


Reasoning with quantifiers and theories

Two Dimensions of Complexity

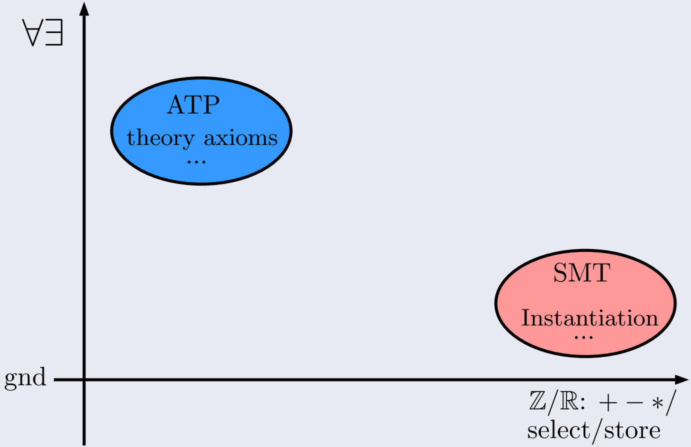


Two Dimensions of Complexity



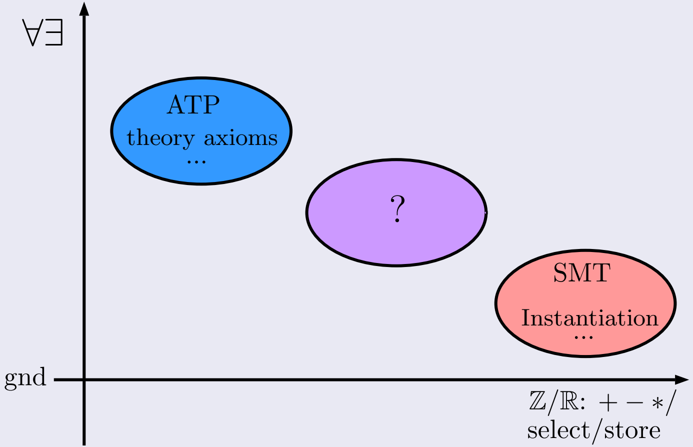
Reasoning with quantifiers and theories

Two Dimensions of Complexity



Reasoning with quantifiers and theories

Two Dimensions of Complexity



Outline

1. A Brief Introduction to Saturation-Based Proving
2. Theory Reasoning in Vampire
3. Theory Instantiation and Unification with Abstraction
4. Where We Currently Stand

Theorem Proving Pipeline in One Slide

Standard form of the input:

$$F \quad := \quad (Axiom_1 \land \ldots \land Axiom_n) \to Conjecture$$

Theorem Proving Pipeline in One Slide

Standard form of the input:

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

1. Negate F to seek a refutation:

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

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)

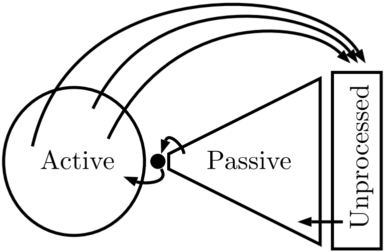
$$\mathcal{S} \quad := \quad \{C_1, \ldots, C_n\}$$

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

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

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

1. A Brief Introduction to Saturation-Based Proving
2. Theory Reasoning in Vampire
3. Theory Instantiation and Unification with Abstraction
4. Where We Currently Stand

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

Adding Theory Axioms

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

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

The AVATAR architecture [Voronkov14]

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

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

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

AVATAR modulo Theories

The AVATAR architecture [Voronkov14]

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

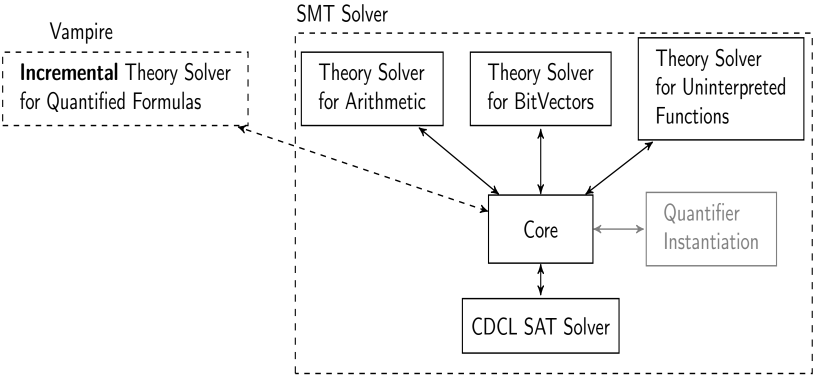
$$\forall x, z, w.\ \underbrace{s(x) \vee \neg r(x, z)}_{\textit{share } x \textit{ and } z} \vee \underbrace{\neg q(w)}_{\textit{is disjoint}}$$

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

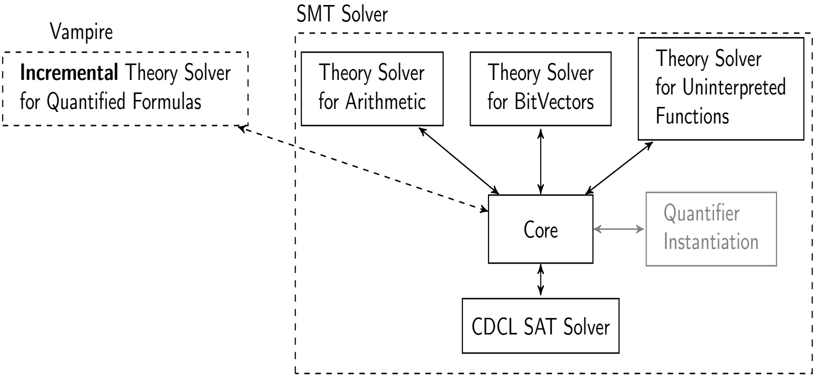
AVATAR modulo Theories

- 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



…and please remember: Vampire is the boss here!

Outline

1. A Brief Introduction to Saturation-Based Proving
2. Theory Reasoning in Vampire
3. Theory Instantiation and Unification with Abstraction
4. Where We Currently Stand

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!

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!

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.

The Theory Instantiation

Instantiation which makes some theory literals immediately false

The Theory Instantiation

Instantiation which makes some theory literals immediately false

As an inference rule

$$\frac{C}{(D[\mathsf{x}])\theta}\ \textit{TheoryInst}$$

where $T[\mathsf{x}] \rightarrow D[\mathsf{x}]$ is a (partial) abstraction of C and θ a substitution such that $T[\mathsf{x}]\theta$ is valid in the underlying theory

The Theory Instantiation

Instantiation which makes some theory literals immediately false

As an inference rule

$$\frac{C}{(D[\mathsf{x}])\theta}\ \mathit{TheoryInst}$$

where $T[\mathsf{x}] \rightarrow D[\mathsf{x}]$ is a (partial) abstraction of C and θ a substitution such that $T[\mathsf{x}]\theta$ is valid in the underlying theory

Implementation:

- Abstract relevant literals
- Collect relevant pure theory literals $L_1, \ldots, L_n$
- Run an SMT solver on $T[\mathsf{x}] = \neg L_1 \wedge \ldots \wedge \neg L_n$
- If the SMT solver returns a model, transform it into a substitution θ and produce an instance

The Theory Instantiation

Instantiation which makes some theory literals immediately false

As an inference rule

$$\frac{C}{(D[\mathsf{x}])\theta}\ \mathit{TheoryInst}$$

where $T[\mathsf{x}] \to D[\mathsf{x}]$ is a (partial) abstraction of C and θ a substitution such that $T[\mathsf{x}]\theta$ is valid in the underlying theory

Implementation:

- Abstract relevant literals
- Collect relevant pure theory literals $L_1, \ldots, L_n$
- Run an SMT solver on $T[\mathsf{x}] = \neg L_1 \wedge \ldots \wedge \neg L_n$
- If the SMT solver returns a model, transform it into a substitution θ and produce an instance

Example

Consider two clauses

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

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

Example

Consider two clauses

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

We could fully abstract them to obtain:

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

then resolve to get

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

Unification with Abstraction

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

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

Finally, Theory Instantiation could produce

$$p(7)$$

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

Outline

1 A Brief Introduction to Saturation-Based Proving

2 Theory Reasoning in Vampire

3 Theory Instantiation and Unification with Abstraction

4 Where We Currently Stand

SMT-COMP 2017 results – ∀∃ problems

Logic	Vampire	VeriT	CVC4	Z3
ALIA	36	27	**42**	**42**
AUFDTLIA	624	-	**728**	-
AUFLIA	**3**	2	**3**	2
AUFLIRA	19778	19316	19766	**19849**
AUFNIRA	**1072**	-	1052	1031
LIA	229	170	**388**	**388**
LRA	1092	-	2048	**2208**
NIA	5	-	9	**13**
NRA	3803	-	3776	**3805**
UF	**4317**	3242	4125	2846
UFDT	2283	-	**2503**	-
UFDTLIA	**75**	-	73	-
UFIDL	55	55	**60**	59
UFLIA	7559	7518	**7687**	7221
UFLRA	10	10	11	**12**
UFNIA	**2561**	-	2189	2197

Conclusion

Thank you for your attention!