QSMA: A New Algorithm for Quantified Satisfiability Modulo Theory and Assignment

M. P. Bonacina[1], S. Graham-Lengrand[2], Ch. Vauthier[3]

[1] Università degli Studi di Verona, Verona, Italy
[2] SRI International, Menlo Park, USA
[3] École Normale Supérieure, Paris, France

CADE 2023

Overview

The paper is about SMT-solving with quantifiers.

It describes the theory behind the YicesQS solver
that entered the SMT-comp in 2021, 2022 and 2023.

The main algorithm is QSMA
for Quantified Satisfiability Modulo Theory and Assignment.

- It is parametric over a single theory $\mathcal{T}$ (no theory combination)
  and a quantifier-free reasoner for $\mathcal{T}$+Boolean reasoning, that offers an API for answering 3 kinds of queries (not just quantifier-free satisfiability).
  The specs for theory $\mathcal{T}$ and its reasoner imply that $\mathcal{T}$ is a complete theory.

- It takes an input assignment in addition to the input formula, and answers whether the input assignment can be extended into a model of the formula.

The paper also reports on YicesQS's performance at the 2022 SMT-comp.
YicesQS won real arithmetic with quantifiers (single-track), performing extremely well in the linear case, and can also be run on integer arithmetic and bitvectors.

Main ingredients

The QSMA algorithm and its variant OptiQSMA

Experimental results (2022 SMT-comp)

Conclusion

1. Main ingredients

Background and timeline

Automated Reasoning in presence of theories and quantifiers is hard.
Approaches include:

- Augmenting a calculus for predicate logic (as in saturation-based first-order theorem proving) with native reasoning mechanisms for theories (rather than axiomatizing them)
- Augmenting an SMT calculus for quantifier-free problems (e.g., CDCL(T)) with generic quantifier instantiation mechanisms (e.g., E-matching)

QSMA belongs to a third kind: it augments a quantifier-free blackbox reasoner R($\mathcal{T}$) for a single theory $\mathcal{T}$ + Booleans, yet it does not instantiate quantifiers.
QSMA embodies a lazy approach to quantifier elimination (QE), where

- theory-specific basic blocks of QE done by R($\mathcal{T}$), (R($\mathcal{T}$) does more than simply deciding $\mathcal{T}$-satisfiability);
- theory-generic orchestration done by QSMA proper.

Background and timeline

Automated Reasoning in presence of theories and quantifiers is hard.
Approaches include:

- Augmenting a calculus for predicate logic (as in saturation-based first-order theorem proving) with native reasoning mechanisms for theories (rather than axiomatizing them)
- Augmenting an SMT calculus for quantifier-free problems (e.g., CDCL(T)) with generic quantifier instantiation mechanisms (e.g., E-matching)

QSMA belongs to a third kind: it augments a quantifier-free blackbox reasoner R($\mathcal{T}$) for a single theory $\mathcal{T}$ + Booleans, yet it does not instantiate quantifiers.
QSMA embodies a lazy approach to quantifier elimination (QE), where

- theory-specific basic blocks of QE done by R($\mathcal{T}$), (R($\mathcal{T}$) does more than simply deciding $\mathcal{T}$-satisfiability);
- theory-generic orchestration done by QSMA proper.

Its implementation YicesQS came first, as an exercise examplifying Yices's functionalities for computing model-based over-approximations (MBO) and under-approximations (MBU), following the development of Yices's MCSAT solver. YicesQS first entered the SMT-comp in 2021 tentatively.
We were convinced that the approach was sound, complete and terminating.
The theory was proved last summer in joint work with M. P. Bonacina and Ch. Vauthier. The paper reports on this work and on the performance of YicesQS at the 2022 SMT comp.

Lazy QE and approximations

QE property: For any formula $\phi[\bar{z}]$ there is a $\mathcal{T}$-equivalent quantifier-free $A[\bar{z}]$.

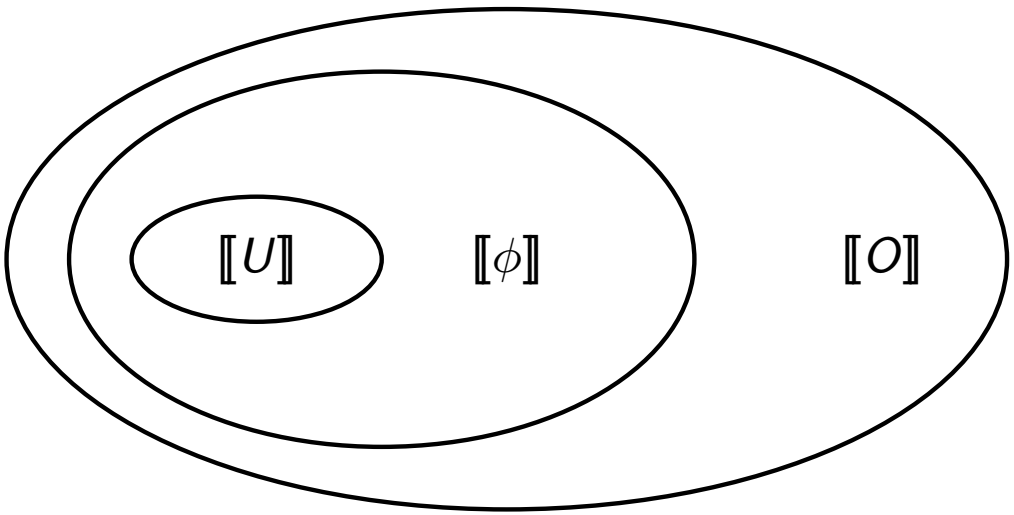
The exact characterization of a formula ϕ as a quantifier-free A is often extremely costly to compute. If the only reason to produce A is to decide whether ϕ is satisfiable, it may not be necessary to compute A exactly.

Lazy QE and approximations

QE property: For any formula $\phi[\bar{z}]$ there is a $\mathcal{T}$-equivalent quantifier-free $A[\bar{z}]$.

The exact characterization of a formula ϕ as a quantifier-free A is often extremely costly to compute. If the only reason to produce A is to decide whether ϕ is satisfiable, it may not be necessary to compute A exactly.

- An *over-approximation* of ϕ is a quantifier-free formula O with $[\![\phi]\!] \subseteq [\![O]\!]$.
- An *under-approximation* of ϕ is a quantifier-free formula U with $[\![U]\!] \subseteq [\![\phi]\!]$.



$[\![\phi]\!]$ denotes the semantics of a formula ϕ as a set of models satisfying ϕ.

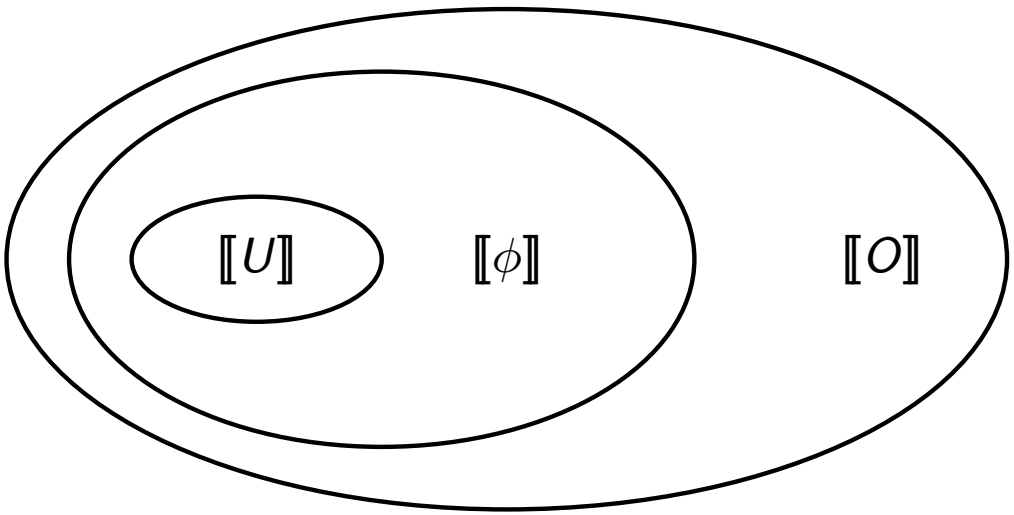
If O is unsat., then ϕ is unsat. If U is sat., then ϕ is sat.

Lazy QE and approximations

QE property: For any formula $\phi[\bar{z}]$ there is a $\mathcal{T}$-equivalent quantifier-free $A[\bar{z}]$.

The exact characterization of a formula ϕ as a quantifier-free A is often extremely costly to compute. If the only reason to produce A is to decide whether ϕ is satisfiable, it may not be necessary to compute A exactly.

- An *over-approximation* of ϕ is a quantifier-free formula O with $[\![\phi]\!] \subseteq [\![O]\!]$.
- An *under-approximation* of ϕ is a quantifier-free formula U with $[\![U]\!] \subseteq [\![\phi]\!]$.



$[\![\phi]\!]$ denotes the semantics of a formula ϕ as a set of models satisfying ϕ.

If O is unsat., then ϕ is unsat. If U is sat., then ϕ is sat.

The main idea in QSMA and YicesQS is to start with $U = \text{false}$ and $O = \text{true}$, and iteratively refine U and O until either U is sat or O is unsat.

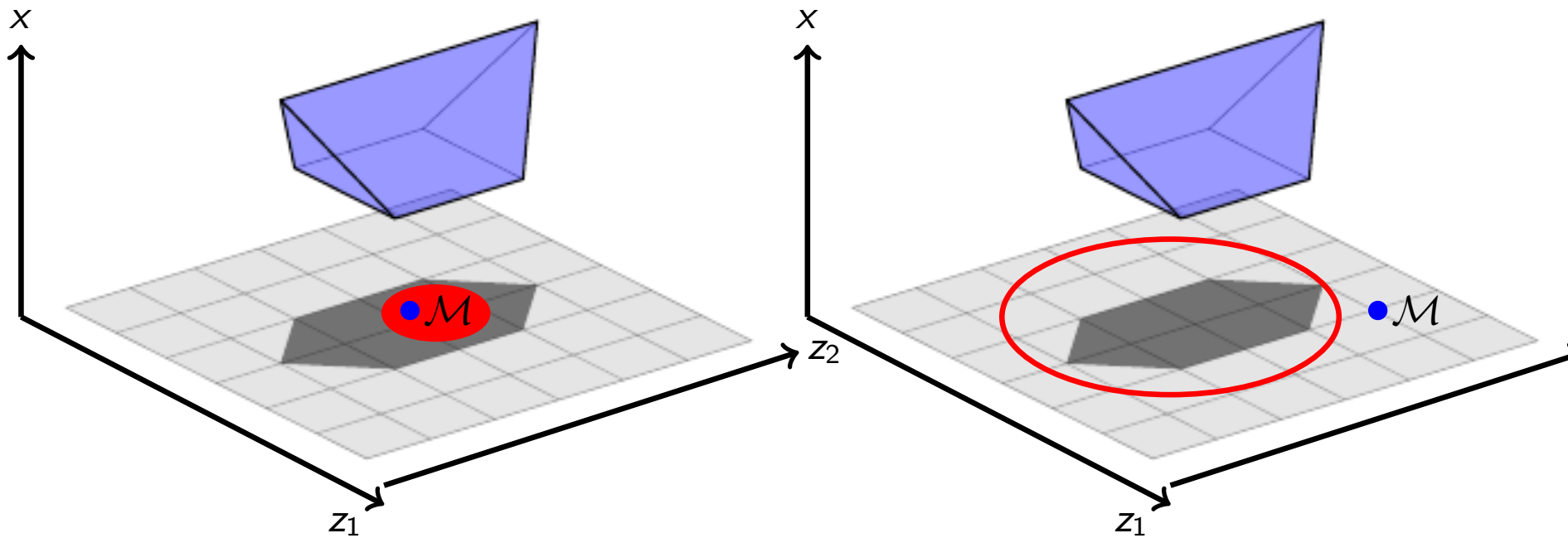
Worst case: you end up computing a quantifier-free formula A such that $[\![\phi]\!] = [\![A]\!]$. In practice, you hope the algorithm stops earlier than that.

Base case for QE, MBU, MBO

- **Quantifier elimination**:
  Given $\exists \bar{x}\ F[\bar{z}, \bar{x}]$ with quantifier-free $F[\bar{z}, \bar{x}]$, produce quantifier-free $A[\bar{z}]$ with $(\exists \bar{x}\ F[\bar{z}, \bar{x}]) \Leftrightarrow A[\bar{z}]$ $\mathcal{T}$-valid.

Base case for QE, MBU, MBO

- **Quantifier elimination**:
  Given $\exists \bar{x}\ F[\bar{z}, \bar{x}]$ with quantifier-free $F[\bar{z}, \bar{x}]$, produce quantifier-free $A[\bar{z}]$ with $(\exists \bar{x}\ F[\bar{z}, \bar{x}]) \Leftrightarrow A[\bar{z}]$ $\mathcal{T}$-valid.
- **Model-Based Under-approximations** $MBU(F[\bar{z}, \bar{x}], \bar{x}, \mathcal{M})$:
  Additionally given $\mathcal{M}$ satisfying $\exists \bar{x}\ F[\bar{z}, \bar{x}]$, produce quantifier-free $U[\bar{z}]$ satisfied by $\mathcal{M}$, with $U[\bar{z}] \Rightarrow (\exists \bar{x}\ F[\bar{z}, \bar{x}])$ $\mathcal{T}$-valid.
- **Model-Based Over-approximations** $MBO(F[\bar{z}, \bar{x}], \bar{x}, \mathcal{M})$:
  Additionally given $\mathcal{M}$ *not* satisfying $\exists \bar{x}\ F[\bar{z}, \bar{x}]$, produce quantifier-free $O[\bar{z}]$ *not* satisfied by $\mathcal{M}$, with $(\exists \bar{x}\ F[\bar{z}, \bar{x}]) \Rightarrow O[\bar{z}]$ $\mathcal{T}$-valid.



In **blue**: $F[z_1, z_2, x]$; its **grey shadow**: $\exists x\ F[z_1, z_2, x]$;
in **red**: the under-approximation $U[z_1, z_2]$ / the over-approximation $O[z_1, z_2]$.

Related work

Quantified reasoning techniques that more or less relate to lazy QE have been used in SMT-solving, for instance

- model-based projections (MBP) [KGC14, BJ15].
- counterexample-guided quantifier instantiation (CEGQI) (e.g, [Dut15, RKK17, NPR+21])

They mostly work with MBU. For instance in order to satisfy $\forall \bar{x}.F[\bar{z}, \bar{x}]$:

- run an SMT query on $\neg F[\bar{z}, \bar{x}]$
- if a model is found, exploit the model with MBU. In CEGQI: to produce an instantiation of $\forall \bar{x}.F[\bar{z}, \bar{x}]$.

In QSMA,

- we solve $\neg F[\bar{z}, \bar{x}]$ modulo the current assignment/model $\mathcal{M}$ for $\bar{z}$ (i.e. we solve $\neg F[\mathcal{M}(\bar{z}), \bar{x}]$ in order to defeat model $\mathcal{M}$, as in [Dut15]);
- we also exploit MBO in case a model is **not** found.

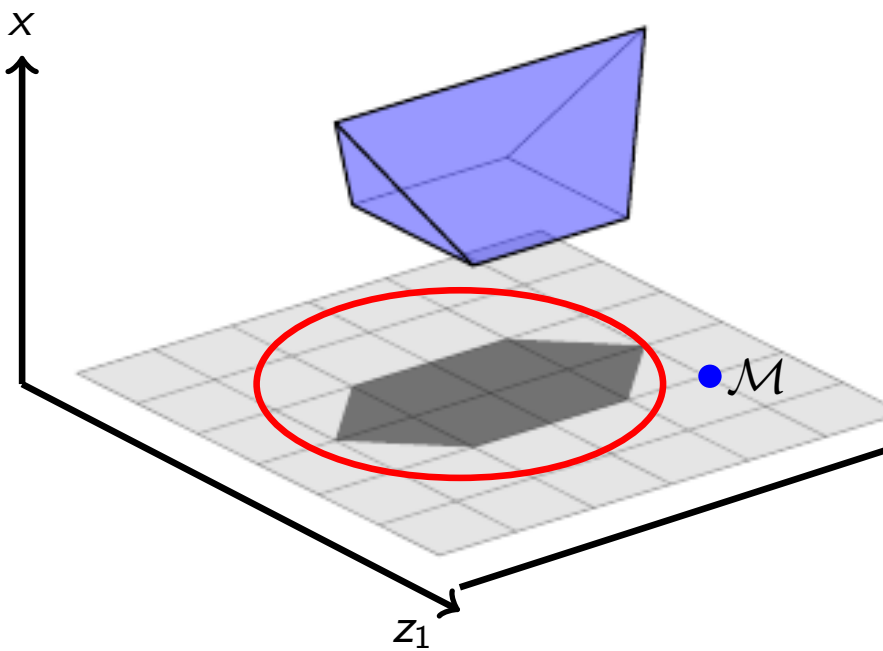
MBO is not unfamiliar

- **UNSAT cores are Boolean MBO**:
  Solving modulo assumptions $a_1, \ldots, a_n$ means
  solving some $\exists \bar{x}.F[a_1, \ldots, a_n, \bar{x}]$ modulo assignment $a_1 \mapsto \text{true}, \ldots, a_n \mapsto \text{true}$
  An unsat core $a_{\pi(1)}, \ldots, a_{\pi(k)}$ with $k \leq n$ corresponds to the
  over-approximation $\neg(a_{\pi(1)} \wedge \cdots \wedge a_{\pi(k)})$ of $\exists \bar{x}.F[a_1, \ldots, a_n, \bar{x}]$.
  In other words: unsat cores are MBO for Boolean assignments.
  Note: Unsat cores are used in [BJ15]
  MBO generalizes UNSAT cores to theory assignments.

MBO is not unfamiliar

- **UNSAT cores are Boolean MBO**:
  Solving modulo assumptions $a_1, \ldots, a_n$ means
  solving some $\exists \bar{x}.F[a_1, \ldots, a_n, \bar{x}]$ modulo assignment $a_1 \mapsto \text{true}, \ldots, a_n \mapsto \text{true}$
  An unsat core $a_{\pi(1)}, \ldots, a_{\pi(k)}$ with $k \leq n$ corresponds to the
  over-approximation $\neg(a_{\pi(1)} \wedge \cdots \wedge a_{\pi(k)})$ of $\exists \bar{x}.F[a_1, \ldots, a_n, \bar{x}]$.
  In other words: unsat cores are MBO for Boolean assignments.
  Note: Unsat cores are used in [BJ15]
  MBO generalizes UNSAT cores to theory assignments.

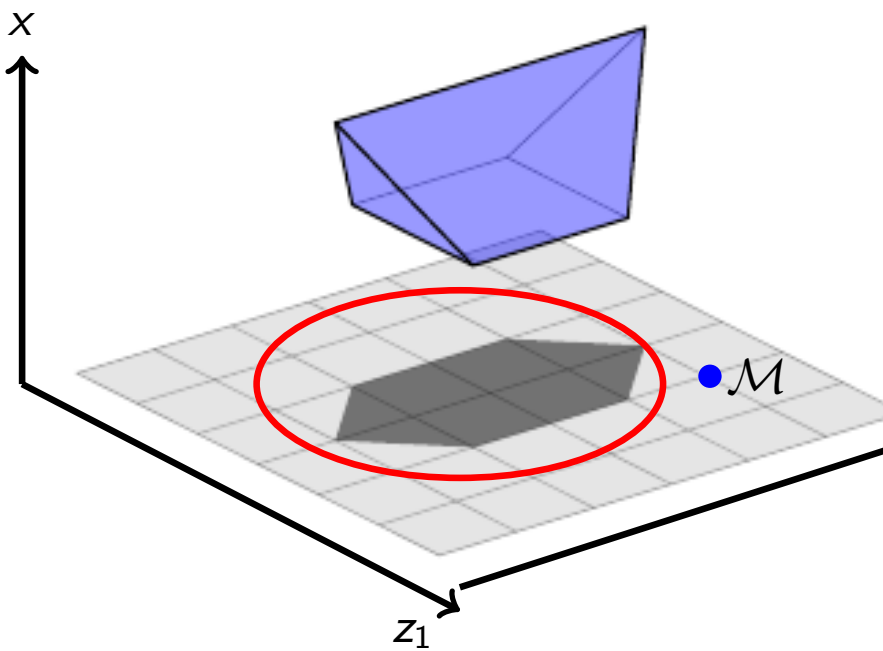
- **MBO relates to (reverse) interpolation**:
  An MBO separates a formula $\exists \bar{x}.F[\bar{z}, \bar{x}]$
  from a single model $\mathcal{M}$.
  Originally called *model interpolant*.
  A trivial loop with various models of a
  second formula $\exists \bar{x}'.F'[\bar{z}, \bar{x}']$, assuming its
  termination, produces Craig interpolants
  separating F and F' [JD21].
  Note: Interpolants are used in [KGC14].



MBO is not unfamiliar

- **UNSAT cores are Boolean MBO**:
  Solving modulo assumptions $a_1, \ldots, a_n$ means
  solving some $\exists \bar{x}.F[a_1, \ldots, a_n, \bar{x}]$ modulo assignment $a_1 \mapsto \text{true}, \ldots, a_n \mapsto \text{true}$
  An unsat core $a_{\pi(1)}, \ldots, a_{\pi(k)}$ with $k \leq n$ corresponds to the
  over-approximation $\neg(a_{\pi(1)} \wedge \cdots \wedge a_{\pi(k)})$ of $\exists \bar{x}.F[a_1, \ldots, a_n, \bar{x}]$.
  In other words: unsat cores are MBO for Boolean assignments.
  Note: Unsat cores are used in [BJ15]
  MBO generalizes UNSAT cores to theory assignments.

- **MBO relates to (reverse) interpolation**:
  An MBO separates a formula $\exists \bar{x}.F[\bar{z}, \bar{x}]$
  from a single model $\mathcal{M}$.
  Originally called *model interpolant*.
  A trivial loop with various models of a
  second formula $\exists \bar{x}'.F'[\bar{z}, \bar{x}']$, assuming its
  termination, produces Craig interpolants
  separating F and F' [JD21].
  Note: Interpolants are used in [KGC14].



- **MBO is the core mechanism of MCSAT**:
  Conflict explanations in MCSAT [dMJ13] are MBO for a single variable x and a conjunction of theory literals. MCSAT can be instrumented to produce MBO / model interpolants for arbitrary quantifier-free formulas [JD21].

2. The QSMA algorithm and its variant OptiQSMA

QSMA input and QSMA tree

QSMA does not prenexify or Skolemize the input formula.
It handles quantifiers wherever they occur in the input formula:

Arbitrarily quantified formula $\varphi[\bar{z}]$ can be expressed in the following grammar:

$$\varphi, \varphi_1, \ldots, \varphi_k ::= \exists \bar{x}.F[\bar{z}, \bar{x}, \bar{p}]\{p_i \leftarrow \varphi_i[\bar{z}, \bar{x}]\}_{i=1}^{k}.$$

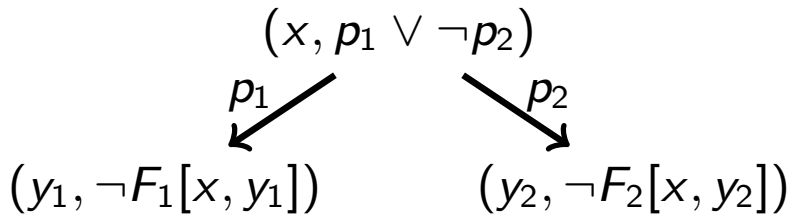
where subformulas with an existential quantifier at their heads are abstracted with proxy Boolean variables $\bar{p}$ and $F[\bar{z}, \bar{x}, \bar{p}]$ quantifier-free.

That grammar gives to the input formula an inductive structure, called the QSMA-tree, that is coarser-grained than the formula tree.
Base case when $\bar{p}$ has length 0 (no quantified subformulas).

Example: formula φ of the form $\exists x.((\exists y_1.\neg F_1[x, y_1]) \vee (\forall y_2.F_2[x, y_2]))$, where F_1 and F_2 are quantifier-free.

QSMA sees φ as $\exists x.(p_1 \vee \neg p_2)$, where p_1 and p_2 are proxy Boolean variables for $\exists y_1.\neg F_1[x, y_1]$ and $\exists y_2.\neg F_2[x, y_2]$.

QSMA tree:



Notations and QSMA spec

- **Formula at node n.** Each node $n = (\bar{x}, F)$ of the tree T corresponds to a subformula $n.\psi = \exists \bar{x}.F[\ldots]$ of the input formula.
- **Recursivity**
  QSMA is a recursive algorithm that follows the QSMA tree T of the input formula $\varphi[\bar{z}]$. Initially, it takes T as input and an assignment $\mathcal{M}$ for $\bar{z}$.
  Recursively, it takes as input:
  a node n of T and an assignment $\mathcal{M}$ for $Rigid(n)$, where:
- **Rigid variables.** For a QSMA tree T corresponding to input formula $\varphi[\bar{z}]$:
  $Rigid(root(T)) = \{\bar{z}\}$
  $Rigid(b) = Rigid(n) \cup \{\bar{x}\}$ where $n = (\bar{x}, F)$ is b's parent
- **Under- and Over-approximations at each node.**
  QSMA keeps for each node n of T an under-approximation $n.U$ and an over-approximation $n.O$ of $n.\psi$, respectively initialized with false and true.
- **Spec for QSMA.**
  pre-condition: $\mathcal{M}$ assignment to $Rigid(n)$,
  and $\forall b \in T.\ [\![b.U]\!] \subseteq [\![b.\psi]\!] \subseteq [\![b.O]\!]$ (I)
  post-condition with return value rv:
  (I) and
  $\mathcal{M} \models (n.U \vee \neg n.O)$ and $(rv$ iff $\mathcal{M} \models n.U)$ and $(\neg rv$ iff $\mathcal{M} \models \neg n.O)$

QSMA algorithm

1: **function** SUBTREEISSOLVED(n, $\mathcal{M}$)
2: $\quad$ **if** $\mathcal{M} \models n.U$ **then**
3: $\quad\quad$ **return** *true*
4: $\quad$ **else if** $\mathcal{M} \models \neg n.O$ **then**
5: $\quad\quad$ **return** *false*
6: $\quad$ **while** true **do**
7: $\quad\quad$ $L \leftarrow n.F \wedge \bigwedge_{n \xrightarrow{p} b}((p \Rightarrow b.O) \wedge (\neg p \Rightarrow \neg b.U))$
8: $\quad\quad$ $\mathcal{M}' \leftarrow \mathsf{SMA}(L, \mathcal{M})$
9: $\quad\quad$ **if** $\mathcal{M}' = \mathit{nil}$ **then**
10: $\quad\quad\quad$ $n.O \leftarrow n.O \wedge \mathsf{MBO}(L, FV(L) \setminus \mathit{Rigid}(n), \mathcal{M})$
11: $\quad\quad\quad$ **return** *false*
12: $\quad\quad$ **else**
13: $\quad\quad\quad$ **if** SOLUTIONFORALLCHILDREN(n, $\mathcal{M}'$) **then**
14: $\quad\quad\quad\quad$ $L' \leftarrow n.F \wedge \bigwedge_{n \xrightarrow{p} b}((p \Rightarrow b.U) \wedge (\neg p \Rightarrow \neg b.O))$
15: $\quad\quad\quad\quad$ $n.U \leftarrow n.U \vee \mathsf{MBU}(L', FV(L') \setminus \mathit{Rigid}(n), \mathcal{M})$
16: $\quad\quad\quad\quad$ **return** *true*
17:
18: **function** SOLUTIONFORALLCHILDREN(n, $\mathcal{M}$)
19: $\quad$ **for** all b such that $n \xrightarrow{p} b$ **do**
20: $\quad\quad$ **if** $\mathcal{M}(p) \neq$ SUBTREEISSOLVED(b, $\mathcal{M}$) **then**
21: $\quad\quad\quad$ **return** *false*
22: $\quad$ **return** *true*

QSMA properties

- **QSMA is correct**

  If the pre-condition is satisfied before the call,
  then the post-condition is satisfied after the call.

  That implies that QSMA's output for (the QSMA tree corresponding to)
  an input formula $\varphi[\bar{z}]$ and an assignment $\mathcal{M}$ for $\bar{z}$ describes
  whether $\mathcal{M} \models \varphi[\bar{z}]$.

  For a satisfiability answer for $\varphi[\bar{z}]$,
  simply start QSMA on $\exists \bar{z}.\varphi[\bar{z}]$ and the empty assignment $\mathcal{M}$.

QSMA properties

- **QSMA is correct**

  If the pre-condition is satisfied before the call,
  then the post-condition is satisfied after the call.

  That implies that QSMA's output for (the QSMA tree corresponding to) an input formula $\varphi[\bar{z}]$ and an assignment $\mathcal{M}$ for $\bar{z}$ describes whether $\mathcal{M} \models \varphi[\bar{z}]$.

  For a satisfiability answer for $\varphi[\bar{z}]$,
  simply start QSMA on $\exists \bar{z}.\varphi[\bar{z}]$ and the empty assignment $\mathcal{M}$.
  Assumptions on the quantifier-free blackbox reasoner:

  - SMA$(L, \mathcal{M})$ produces an extension $\mathcal{M}'$ of assignment $\mathcal{M}$ satisfying L if there exists one, nil if not.
  - $MBU(L, \bar{x}, \mathcal{M})$ and $MBO(L, \bar{x}, \mathcal{M})$ satisfy the specs described earlier.

QSMA properties

- **QSMA is correct**

  If the pre-condition is satisfied before the call,
  then the post-condition is satisfied after the call.

  That implies that QSMA's output for (the QSMA tree corresponding to) an input formula $\varphi[\bar{z}]$ and an assignment $\mathcal{M}$ for $\bar{z}$ describes whether $\mathcal{M} \models \varphi[\bar{z}]$.

  For a satisfiability answer for $\varphi[\bar{z}]$,
  simply start QSMA on $\exists \bar{z}.\varphi[\bar{z}]$ and the empty assignment $\mathcal{M}$.
  Assumptions on the quantifier-free blackbox reasoner:
  - $\mathrm{SMA}(L, \mathcal{M})$ produces an extension $\mathcal{M}'$ of assignment $\mathcal{M}$ satisfying L if there exists one, nil if not.
  - $MBU(L, \bar{x}, \mathcal{M})$ and $MBO(L, \bar{x}, \mathcal{M})$ satisfy the specs described earlier.

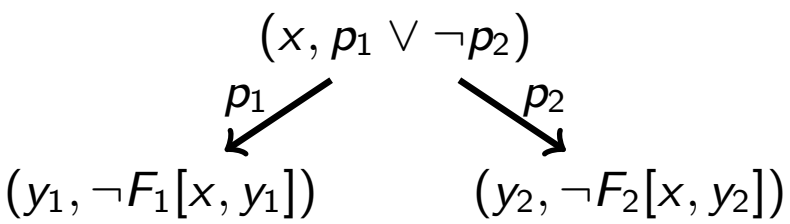
- **QSMA terminates**

  Assumptions on the quantifier-free blackbox reasoner:
  - SMA terminates on each call.
  - MBU and MBO terminate on each call and *have finite bases*:
    The sets
    $$\{\mathrm{MBU}(F[\bar{z}, \bar{x}], \bar{x}, \mathcal{M}) \mid \mathcal{M} \text{ assignment to } \bar{z} \text{ such that } \mathcal{M} \models \exists \bar{x}.F[\bar{z}, \bar{x}]\}$$
    and
    $$\{\mathrm{MBO}(F[\bar{z}, \bar{x}], \bar{x}, \mathcal{M}) \mid \mathcal{M} \text{ assignment to } \bar{z} \text{ such that } \mathcal{M} \not\models \exists \bar{x}.F[\bar{z}, \bar{x}]\}$$
    are finite for all quantifier-free formulas $F[\bar{z}, \bar{x}]$ and tuples $\bar{x}$.

Optimization OptiQSMA implemented in YicesQS

Idea: reduce the number of recursive calls to `subtreeIsSolved` by entrusting more work to each call to SMA.

Example:



After the first call to SMA giving an assignment $\mathcal{M}$ for x, p_1 and p_2, there will be a recursive call on the p_1 child trying to give an assignment to y_1 to satisfy $\neg F_1[x, y_1]$.
If $\mathcal{M}(p_1) = \text{true}$, we could have packaged that task into the first call to SMA, if instead of giving it formula $p_1 \vee \neg p_2$ we gave it $(p_1 \vee \neg p_2) \wedge (p_1 \Rightarrow \neg F_1[x, y_1])$, saving a recursive call on the p_1 child.

In general: instead of making recursive calls on every child of a node n, we skip those children (and their descendants!) for which the proxy variable is true, and look ahead for descendants b whose proxy variable is false. I.e. we look for

$$n \xrightarrow{p_1} n_1 \rightarrow \ldots \xrightarrow{p_q} n_q \xrightarrow{p} b$$

where $\mathcal{M}$ makes $p_1, \ldots, p_q$ true and makes p false.
Such descendants b are called *First Alternation Nodes* (FAN)
(and those $n_1, \ldots, n_q$ in between: *No Alternation Nodes* or NAN),
and only on FANs we make recursive calls.

QSMA properties and implementation

OptiQSMA is correct and terminating, under the same assumptions as QSMA.

OptiQSMA is implemented in an OCaml solver called YicesQS (for Quantified Satisfaction): `https://github.com/disteph/yicesQS`

It builds on top of SRI's Yices SMT-solver `https://yices.csl.sri.com/` for quantifier-free formulas. Yices offers a C API that includes
`check-with-model` (SMA),
`model-interpolant` (MBO),
and `generalize-model` (MBU).

Technically (and since last week),
these work for both SMT-solvers in Yices: CDCL(T) and MCSAT
(before last week: only Yices/MCSAT).

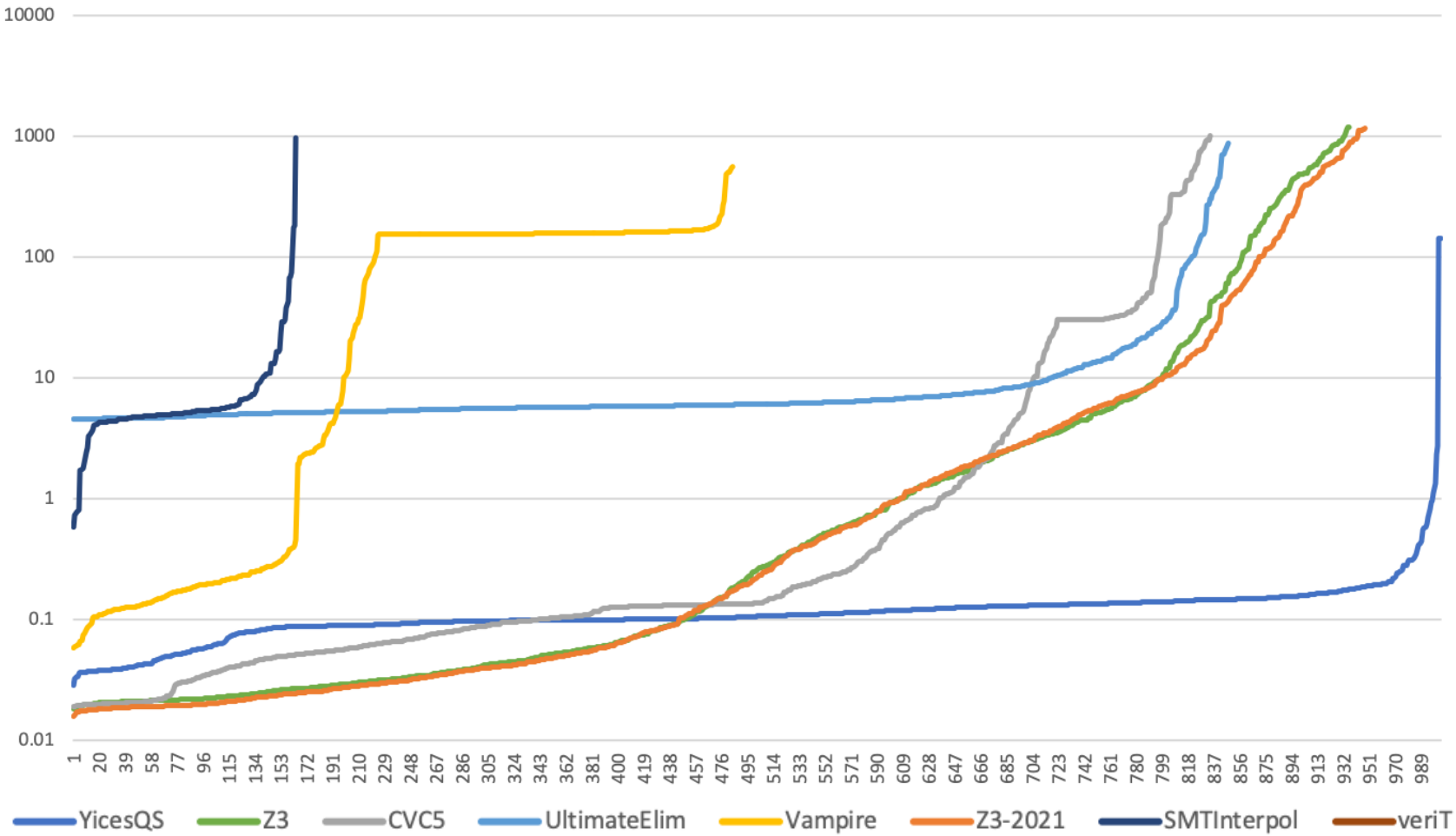
In practice for arithmetic, only Yices/MCSAT provides a useful MBO (based on CAD), and can answer SMA queries for non-linear problems.
MBU for (linear & non-linear) arithmetic is offered separately (based on CAD).

For bitvectors, Yices/CDCL(T) is usually faster than Yices/MCSAT for SMA.
(though at the 2022/2023 SMT-comp we used Yices/MCSAT)
Invertibility conditions [NPR+21] can be used for MBO and MBU.

3. Experimental results (2022 SMT-comp)

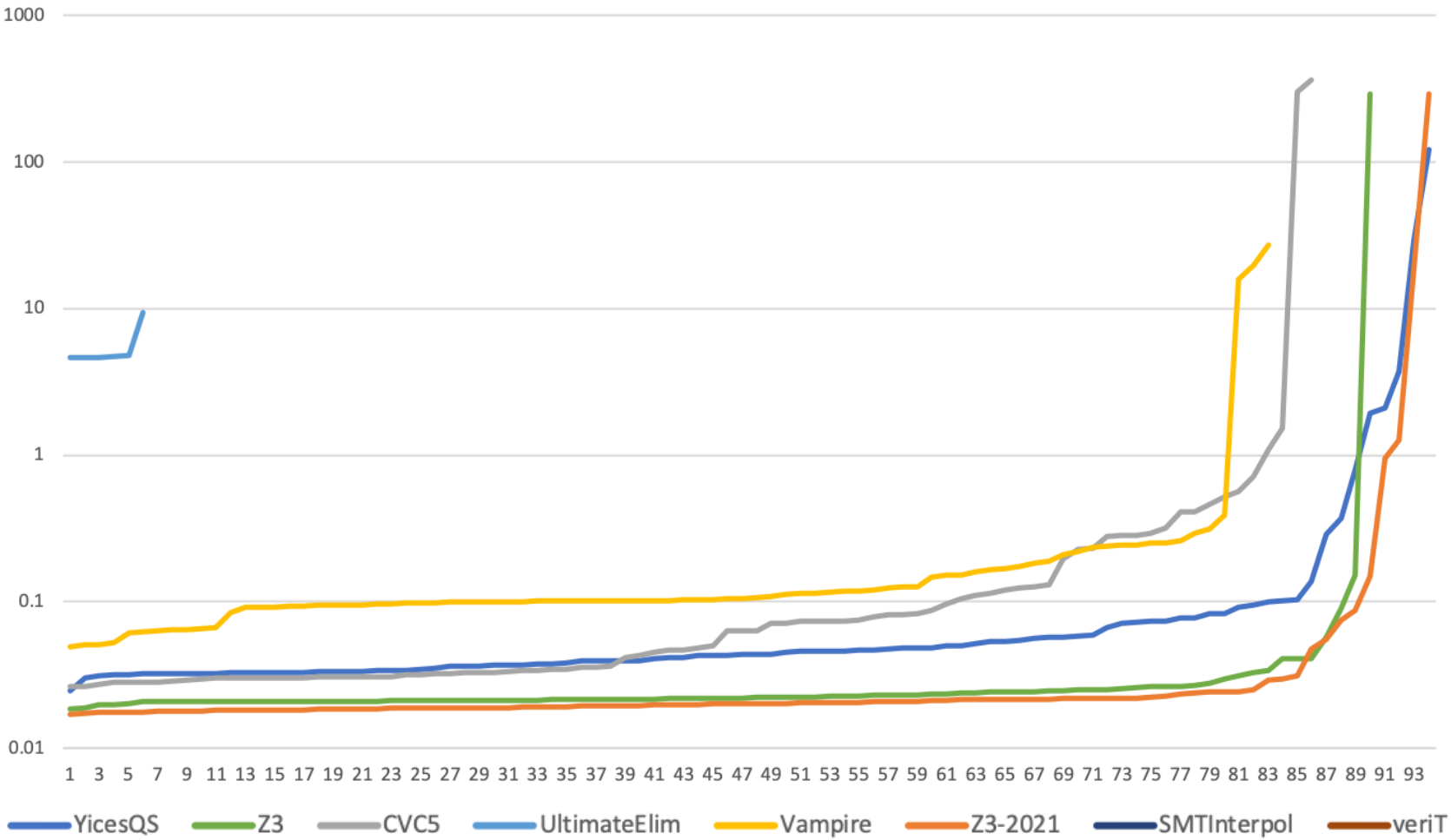
LRA

YicesQS	1003/1003	414s
Z3 2021	948/1003	41,068s
Z3	936/1003	41,240s
Ultim.Elim.	847/1003	16,136s
CVC5	834/1003	21,197s
Vampire	484/1003	45,326s
SMTInterpol	164/1003	2,584s



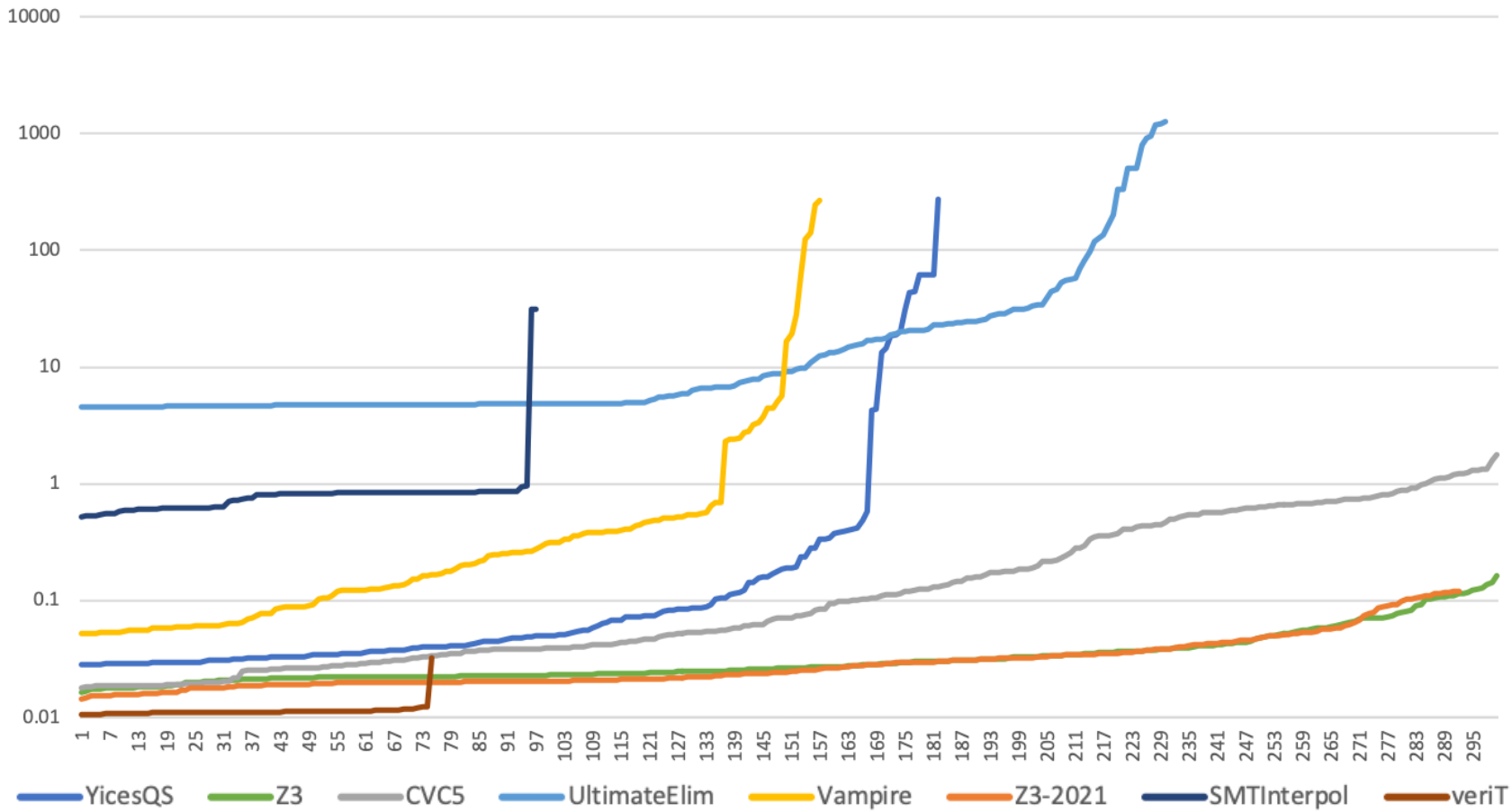
NRA

YicesQS	94/99	165s
Z3 2021	94/99	315s
Z3	90/99	294s
CVC5	86/99	672s
Vampire	83/99	73s
Ultim.Elim.	6/99	33s



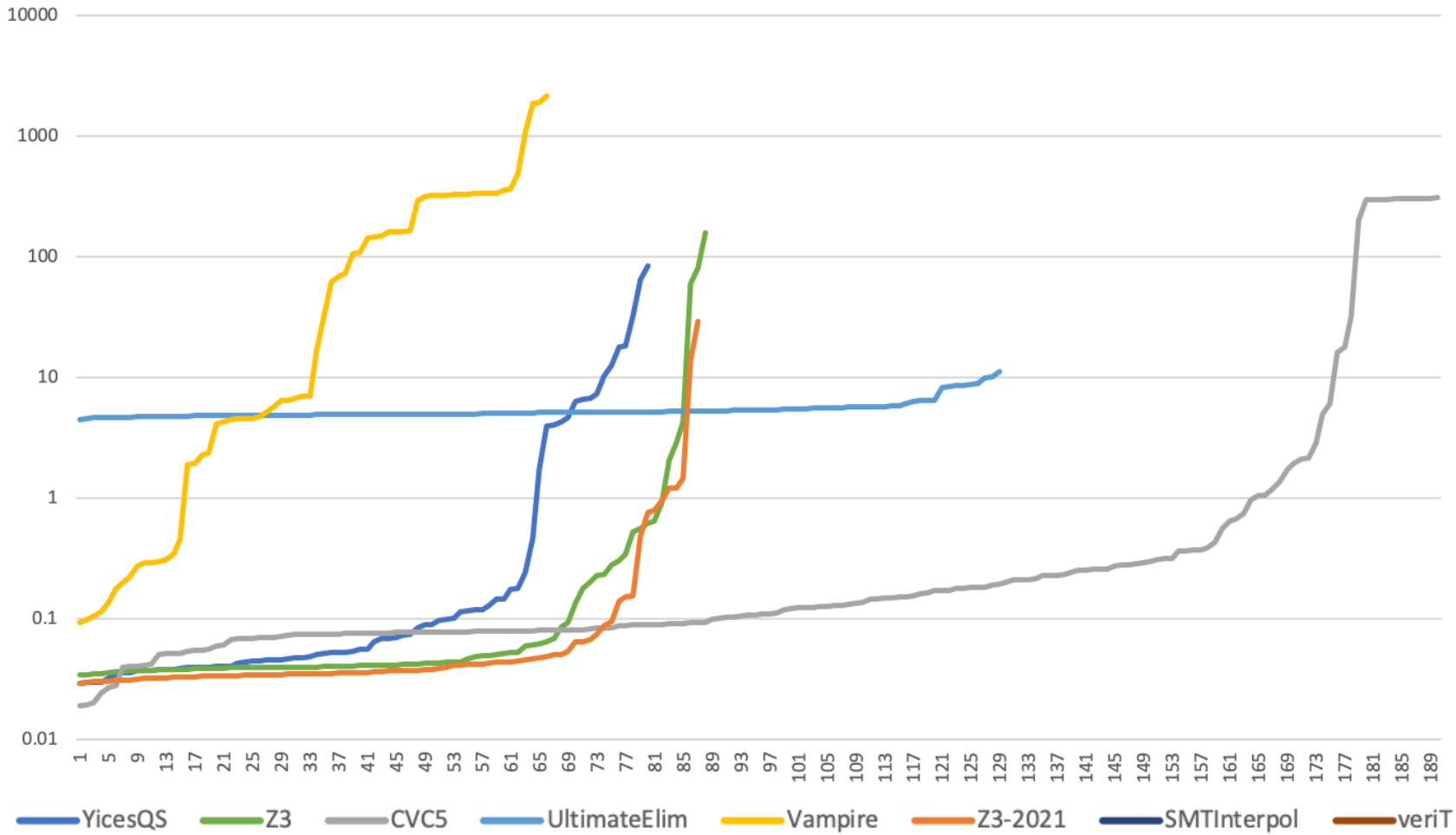
LIA

Z3	300/300	11s
CVC5	300/300	78s
Z3 2021	292/300	10s
Ultim.Elim.	230/300	11,789s
YicesQS	182/300	750s
Vampire	157/300	985s
SMTInterpol	97/300	134s
VeriT	75/300	1s



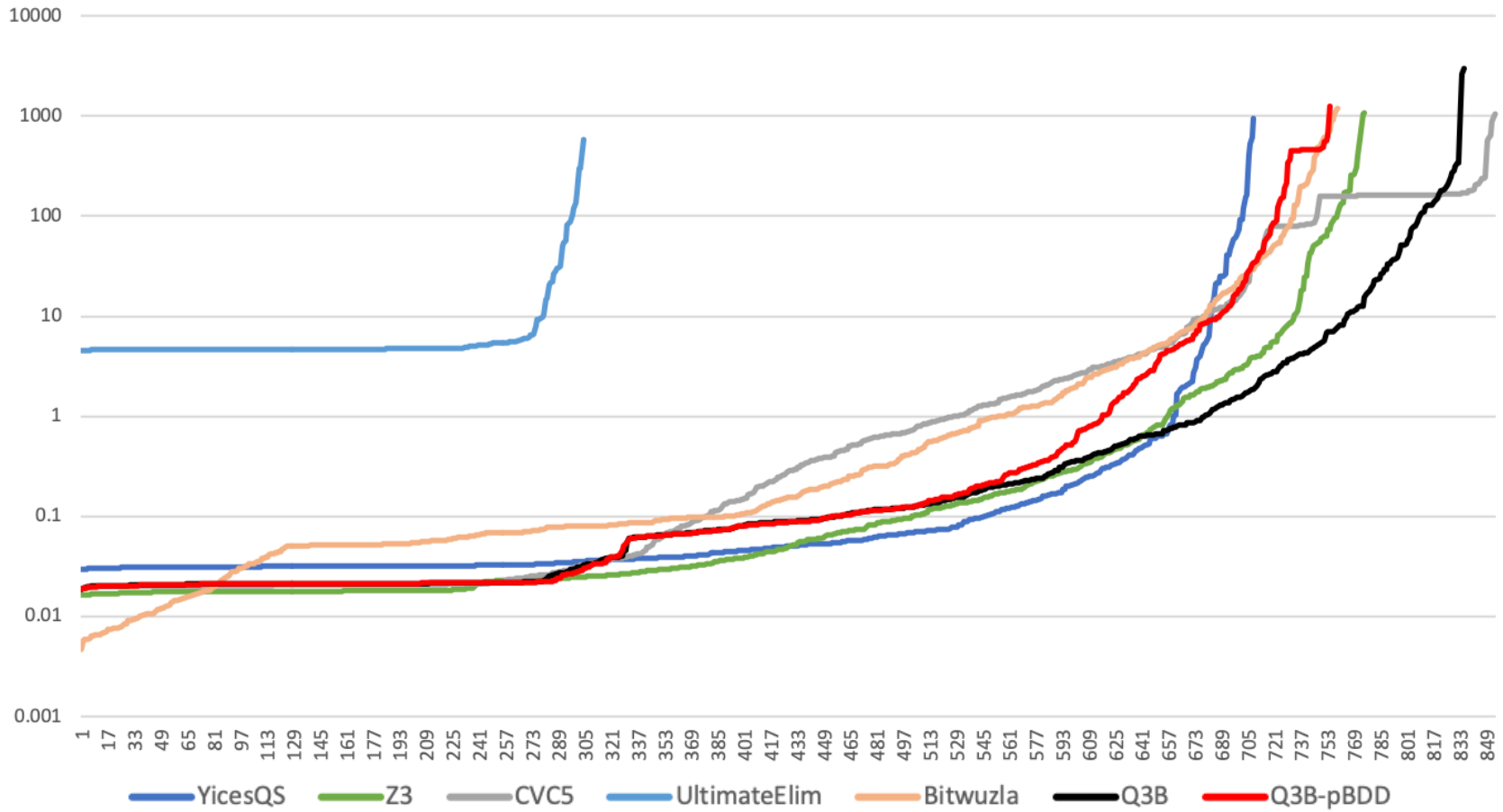
NIA

CVC5	190/208	3,642s
Ultim.Elim.	129/208	701s
Z3	88/208	317s
Z3 2021	87/208	53s
YicesQS	80/208	290s
Vampire	66/208	13,744s



BV

CVC5	854/970	25,584s
Q3B	835/970	13,510s
Z3	775/970	7,712s
Bitwuzla	759/970	15,572s
Q3B-pBDD	754/970	15,553s
YicesQS	708/970	3,862s
Ultim.Elim.	304/970	4,204s



4. Conclusion

Conclusion

At 2022 SMT-comp in arithmetic overall (LRA, NRA, LIA, NIA),
YicesQS wins the 24s timeout mode and wins the satisfiable benchmarks.
At 2023 SMT-comp: same YicesQS as in 2022.
From what I can see on Starexec, results look similar to 2022.

- **BV**:
  - SMA:
    eager to see how bitblasting (Yices/CDCL(T)) will improve performance.
  - MBO and MBU:
    Invertibility conditions target equivalences
    $$(\exists x.l[\bar{z}, x]) \Leftrightarrow F[\bar{z}]$$
    for a single BV literal l with a single occurrence of x.
    MBO and MBU would benefit from literal conjunctions & multi-occurrences of x.
    On the other hand, they don't need equivalence ($\Leftrightarrow$) but only implication
    ($\Rightarrow$ and $\Leftarrow$, respectively).
- **YicesQS on integers**:
  Not much we can do without improving how MCSAT deals with integers for the quantifier-free queries SMA and MBO. We could also imagine using ints-as-bitvectors.
- **Theory combination**:
  Hard to imagine how approaches related to quantifier-elimination can generalize to theory combinations. One of our projects is to integrate quantifiers in the CDSAT framework for theory combination, possibly by importing some ingredients/ideas from QSMA that can still be re-used in presence of multiple theories, starting with EUF.

Questions?

A satisfiability algorithm for a slightly more general question

"Given a formula $A[\bar{z}, \bar{x}]$ and a model $\mathcal{M}_{\bar{z}}$ on $\bar{z}$, produce either

- SAT($U[\bar{z}]$), with $U[\bar{z}]$ under-approx. of $\exists \bar{x}\ A[\bar{z}, \bar{x}]$ satisfied by $\mathcal{M}_{\bar{z}}$; or
- UNSAT($O[\bar{z}]$), with $O[\bar{z}]$ over-approx. of $\exists \bar{x}\ A[\bar{z}, \bar{x}]$ not satisfied by $\mathcal{M}_{\bar{z}}$."

(i.e. $\bar{z}$'s values are imposed, $\bar{x}$ are existentially quantified: values are up to us).

This generalizes the standard satisfiability question:
"Given a formula $A[\bar{x}]$, produce either

- SAT, if $\exists \bar{x}\ A[\bar{x}]$ is satisfied by the empty model (does not assign any value to any variable); or
- UNSAT, if not."

If you have an algorithm to solve the more general problem,
apply it on the empty model $\mathcal{M}$ and $A[\bar{x}]$ ($\bar{z}$ is empty) and inspect the result:

- UNSAT(O): return UNSAT
- SAT(U): return SAT

OptiQSMA

```
1: function optiSubtreeIsSolved(n, 𝓜)
2:     while true do
3:         L ← LF(n) ∧ ⋀_{n→⁺b}(¬b.p ⇒ ¬b.U)
4:         𝓜′ ← SMA(L, 𝓜)
5:         if 𝓜′ = nil then
6:             return UNSAT(MBO(L, FV(L) \ Rigid(n), 𝓜))
7:         else
8:             reasons ← ⊤
9:             if solutionForallDescendants(n, 𝓜′, reasons) then
10:                L′ ← LF(n) ∧ reasons
11:                return SAT(MBU(L′, FV(L′) \ Rigid(n), 𝓜))
12: function solutionForallDescendants(n, 𝓜, reasons)
13:     for all b ∈ FAN(n, 𝓜) do
14:         ans ← optiSubtreeIsSolved(b, 𝓜)
15:         if ans = SAT(U) then
16:             b.U ← b.U ∨ U
17:             return false
18:         else if ans = UNSAT(O) then
19:             reasons ← reasons ∧ (¬b.p ⇒ ¬O)
20:     for all b ∈ NAN(n, 𝓜) do
21:         reasons ← reasons ∧ b.p
22:     return true
```


N. Bjorner and M. Janota.
Playing with quantified satisfaction.
In M. Davis, A. Fehnker, A. McIver, and A. Voronkov, editors, *Proc. of the the 20th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'15)*, volume 9450 of *LNCS*. Springer-Verlag, 2015.


L. M. de Moura and D. Jovanovic.
A model-constructing satisfiability calculus.
In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *Proc. of the 14th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'13)*, volume 7737 of *LNCS*, pages 1–12. Springer-Verlag, 2013.


B. Dutertre.
Solving exists/forall problems with Yices.
In *Proc. SMT-13*, 2015.


D. Jovanovic and B. Dutertre.
Interpolation and model checking for nonlinear arithmetic.
In A. Silva and K. R. M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021,*

Proceedings, Part II, volume 12760 of *Lecture Notes in Computer Science*, pages 266–288. Springer, 2021.

A. Komuravelli, A. Gurfinkel, and S. Chaki.
Smt-based model checking for recursive programs.
In A. Biere and R. Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 17–34. Springer, 2014.

A. Niemetz, M. Preiner, A. Reynolds, C. W. Barrett, and C. Tinelli.
On solving quantified bit-vector constraints using invertibility conditions.
Formal Methods Syst. Des., 57(1):87–115, 2021.

A. Reynolds, T. King, and V. Kuncak.
Solving quantified linear arithmetic by counterexample-guided instantiation.
Formal Methods Syst. Des., 51(3):500–532, 2017.