Efficient checking of positively invariant sets: implementation in Sage

Carybe Bégué

Ecole normale supérieure de Rennes

August 26, 2021

Positively invariant sets

Definition 1.1

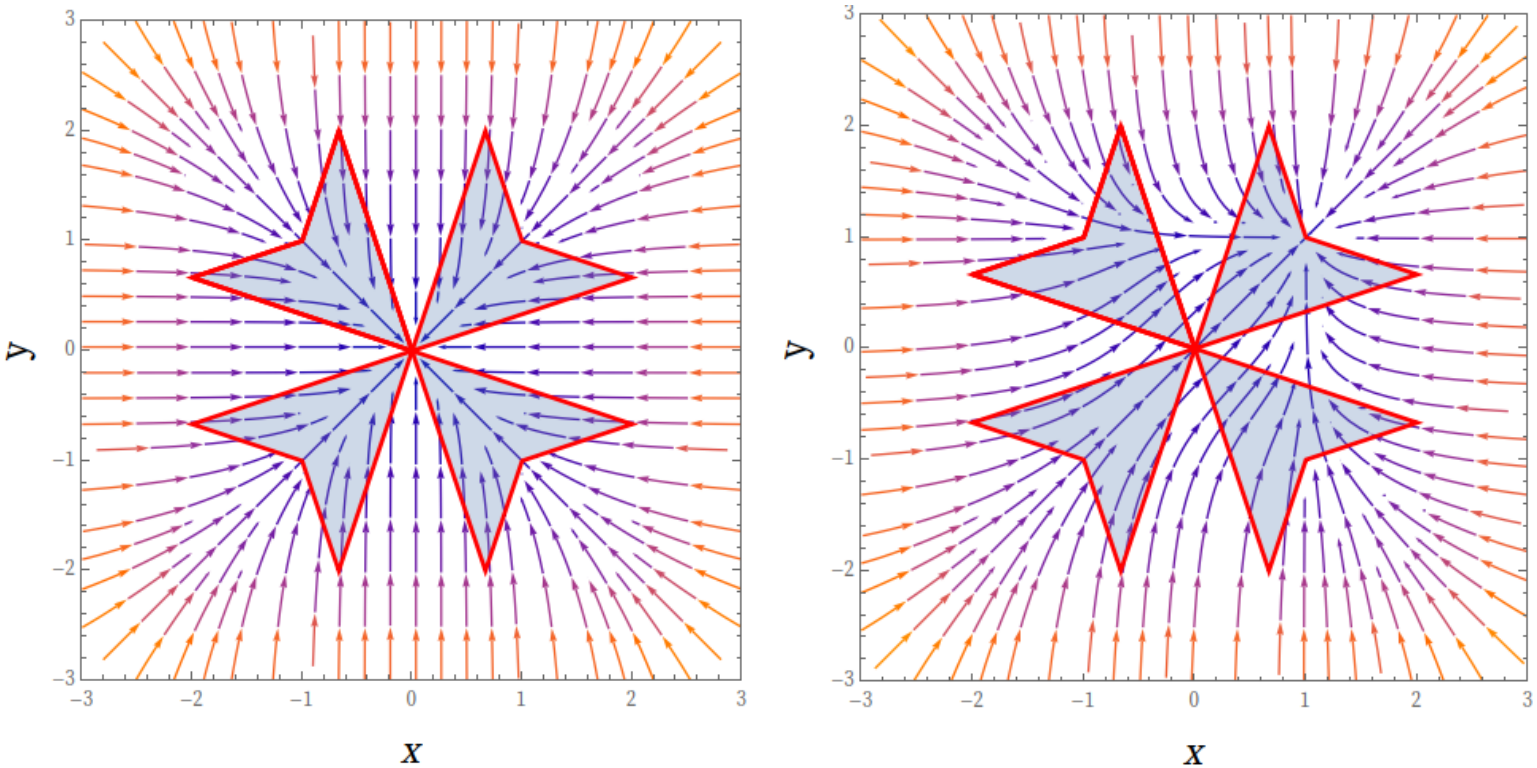
- Let $f = (f_1, \ldots, f_n)$ be a vector-valued continuous and polynomial functions from $\mathbb{R}^n$ to $\mathbb{R}$ and $\varphi(., y)$ the solution of $x' = f(y)$
- Let $S \subseteq \mathbb{R}^n$ be a subset defined by polynomial equalities and inequalities

Definition 1.2 (Positively invariant sets)

Given a set of ODE's equations f and a subset S of $\mathbb{R}^n$, S is positively invariant if and only if :

$$\forall x \in S, \forall t > 0, \varphi(t, x) \in S$$

Examples



(a) Positively invariant set (b) Not positively invariant set

Figure: Example of positively invariant sets

Positively invariant sets

2 algorithms (LZZ and ES) are created and developed in Mathematica to resolve the problem[1]:

- Given f and S, is S positively invariant?

$\Rightarrow$ It were tantamount to a problem of quantifiers elimination

[1] K. Ghorbal and A. Sogokon. "Characterizing Positively Invariant Sets: Inductive and Topological Methods". In: (2020).

Quantifiers elimination

Definition 2.1 (Quantifier elimination)

Let $p_1, \ldots, p_n$ be polynomials over $\mathbb{Q}[X_1, \ldots, X_m]$
Let $\bowtie_i \in \{<, >, =, \neq, \leq, \geq\}$
Let f a first order formula where variables are $p_i \bowtie_i 0$
We want to find a formula g such that :

- g only has free variables
- $f \equiv g$

For example:

$$\exists x \exists y (x + y = 5) \wedge (x < 3) \equiv True$$
$$\forall x \exists y (x + y = 5) \vee (x < 3) \equiv True$$
$$\forall x \forall y (x + y = 5) \wedge (x < 3) \equiv False$$

Sage implementation

- implementation in Sage
- branching 2 solvers: RedLog and QedCad with the same algorithm (CAD algorithm for cylindrical algebraic decomposition)
- QedCad do not function

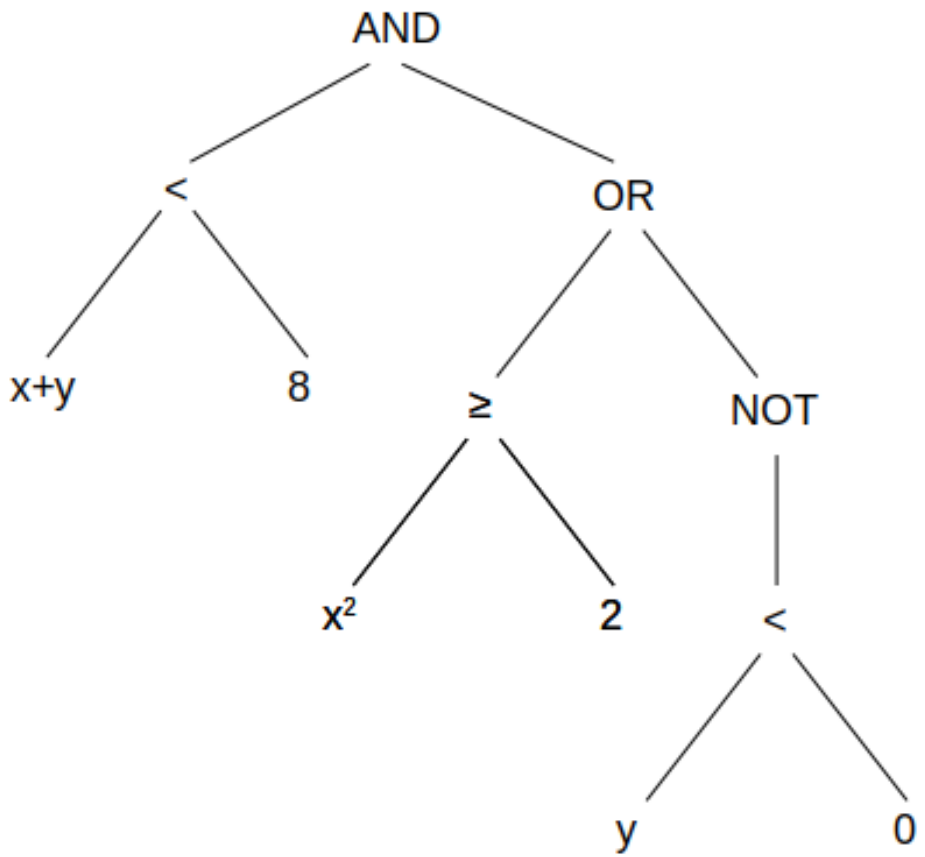
Sage implementation

- S and f declared with strings
- use the Python's parsing to translate the string into a logical tree
- leaf nodes are polynomials

Example

$$\text{"}x + y < 8 \text{ and } (x^2 \geq 2 \text{ or not}(y < 0))\text{"}$$

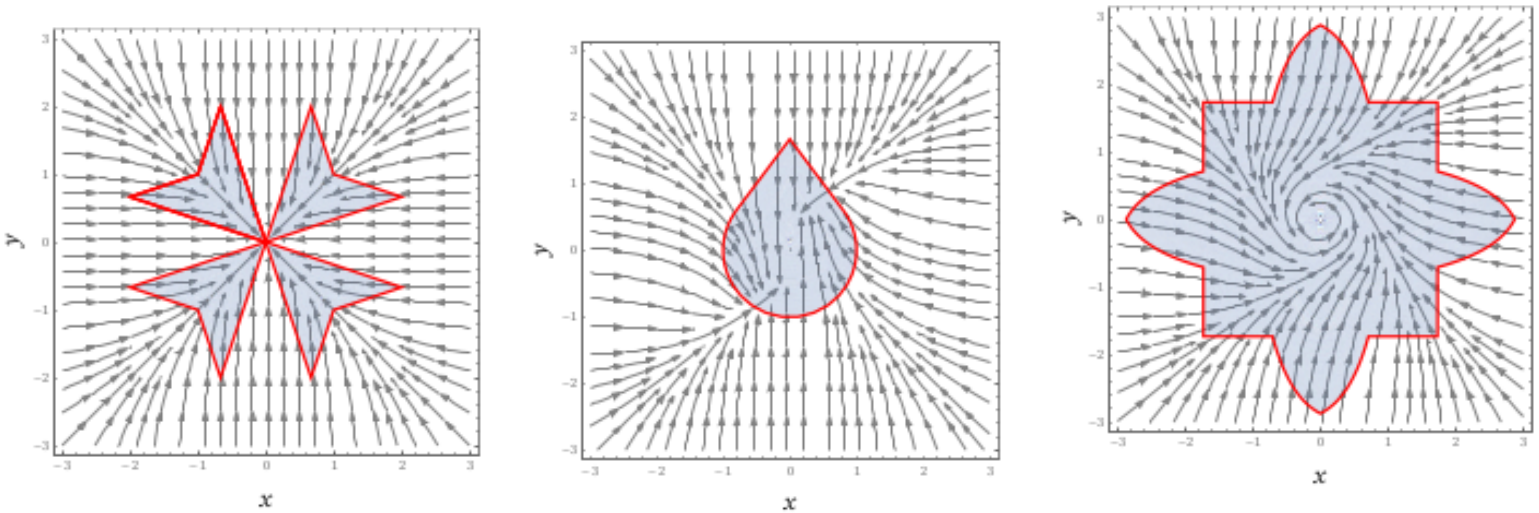
became:



Branching solvers

```
if quant=="ex":
        sin = "rlset r$\nphi:=ex("+vars+","+f+");
                                    \nrlqe phi;\nbye;"
elif quant=="all":
        sin = "rlset r$\nphi:=all("+vars+","+f+");
                                    \nrlqe phi;\nbye;"
process = Popen(['redpsl'], universal_newlines=True)
result = process.communicate(input=sin)
process.terminate()
```

3 benchmarks



(a) Maltese Cross (yes)

(b) Droplet (no)

(c) Non linear shape (yes)

Figure: Example of benchmarks

Execution with Reduce (RedLog)

In Mathematica, ES problem is really more efficient than LZZ problem but not with our new implementation in Sage.

	Sage		Mathematica	
	LZZ	ES	LZZ	ES
droplet	0.04s	0.3s	/	0.3s
malteseCross	0.1s	1.2s	/	164s
Nonlinshape	/	0.3s	30min	7s

Efficiency result

We test the Sage code over 30 benchmarks in 2 and 3 dimensions.

	LZZ mean time	ES mean time
2D benchs	0.04s	0.3s
3D benchs	0.009s	0.03s

Some benchmarks' execution is too long for LZZ and for ES.

Conclusion

- An open source implementation.
- More efficient than the Mathematica implementation over some benchmarks but most benchmarks' execution is too long.
- On average, LZZ is ten times faster than ES.

To do:

- Using a new solver Msolve (not yet developed).
- Understand with LZZ is faster than ES in Sage.