

# A Reflexive Tactic for Polynomial Positivity using Numerical Solvers and Floating-Point Computations

Érik Martin-Dorel<sup>1</sup>   Pierre Roux<sup>2</sup>

<sup>1</sup>IRIT, Université Paul Sabatier, Toulouse, France

<sup>2</sup>ONERA, Toulouse, France

Mercredi 31 Mai 2017

Journées FastRelax, Paris

# Références



P. Roux, “Formal proofs of rounding error bounds - with application to an automatic positive definiteness check,” *J. Autom. Reasoning*, vol. 57, no. 2, pp. 135–156, 2016.

# Références






P. Roux, “Formal proofs of rounding error bounds - with application to an automatic positive definiteness check,” *J. Autom. Reasoning*, vol. 57, no. 2, pp. 135–156, 2016.



P. Roux, Y. Voronin, and S. Sankaranarayanan, “Validating numerical semidefinite programming solvers for polynomial invariants,” in *SAS 2016, Edinburgh, UK, Proceedings*, pp. 424–446, 2016.

# Références

-  P. Roux, “Formal proofs of rounding error bounds - with application to an automatic positive definiteness check,” *J. Autom. Reasoning*, vol. 57, no. 2, pp. 135–156, 2016.
-  P. Roux, Y. Voronin, and S. Sankaranarayanan, “Validating numerical semidefinite programming solvers for polynomial invariants,” in *SAS 2016, Edinburgh, UK, Proceedings*, pp. 424–446, 2016.
-  E. Martin-Dorel and P. Roux, “A Reflexive Tactic for Polynomial Positivity using Numerical Solvers and Floating-Point Computations,” in *CPP 2017, Paris*, pp. 90–99, ACM, janvier 2017.

# Motivation

- Polynomial inequalities in the real field are decidable (Tarski)
- But exact algo. expensive

# Motivation

- Polynomial inequalities in the real field are decidable (Tarski)
  - But exact algo. expensive
- ⇒ Use incomplete numerical methods
- off-the-shelf optimization solvers
  - a posteriori validation with exact rational arithmetic:  
state of the art (simple but costly)

# Motivation

- Polynomial inequalities in the real field are decidable (Tarski)
  - But exact algo. expensive
- ⇒ Use incomplete numerical methods
- off-the-shelf optimization solvers
  - a posteriori validation with exact rational arithmetic:  
state of the art (simple but costly)
  - a posteriori validation with floating-point arithmetic  
(more efficient but non trivial)
- ⇒ We'd like formal proofs

# Motivation

- Polynomial inequalities in the real field are decidable (Tarski)
  - But exact algo. expensive
- ⇒ Use incomplete numerical methods
- off-the-shelf optimization solvers
  - a posteriori validation with exact rational arithmetic: state of the art (simple but costly)
  - a posteriori validation with floating-point arithmetic (more efficient but non trivial)
- ⇒ We'd like formal proofs

demo.v



# Agenda

- 1 Sum of Squares (SOS) Polynomials
- 2 Numerical Verification
- 3 Formalization & Reflexive Tactic
- 4 Benchmarks
- 5 Conclusion

# Agenda

- 1 Sum of Squares (SOS) Polynomials
- 2 Numerical Verification
- 3 Formalization & Reflexive Tactic
- 4 Benchmarks
- 5 Conclusion

# Sum of Squares (SOS) Polynomials

## Definition (SOS Polynomial)

A polynomial  $p$  is SOS if there are polynomials  $q_1, \dots, q_m$  s.t.

$$p = \sum_i q_i^2.$$

- If  $p$  SOS then  $p \geq 0$

# Sum of Squares (SOS) Polynomials

## Definition (SOS Polynomial)

A polynomial  $p$  is SOS if there are polynomials  $q_1, \dots, q_m$  s.t.

$$p = \sum_i q_i^2.$$

- If  $p$  SOS then  $p \geq 0$
- $p$  SOS iff there exist  $z := [1, x_0, x_1, x_0x_1, \dots, x_n^d]$  and  $Q \succeq 0$  (i.e., for all  $x, x^T Q x \geq 0$ ) s.t.

$$p = z^T Q z.$$

⇒ SOS can be encoded as semi-definite programming (SDP).

# SOS: Example

## Example

Is  $p(x, y) := 2x^4 + 2x^3y - x^2y^2 + 5y^4$  SOS ?

$$p(x, y) = \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}^T \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}$$

that is  $p(x, y) = q_{11}x^4 + 2q_{13}x^3y + 2q_{23}xy^3 + (2q_{12} + q_{33})x^2y^2 + q_{22}y^4$

# SOS: Example

## Example

Is  $p(x, y) := 2x^4 + 2x^3y - x^2y^2 + 5y^4$  SOS ?

$$p(x, y) = \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}^T \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}$$

that is  $p(x, y) = q_{11}x^4 + 2q_{13}x^3y + 2q_{23}xy^3 + (2q_{12} + q_{33})x^2y^2 + q_{22}y^4$

hence  $q_{11} = 2$ ,  $2q_{13} = 2$ ,  $2q_{23} = 0$ ,  $2q_{12} + q_{33} = -1$ ,  $q_{22} = 5$ .

# SOS: Example

## Example

Is  $p(x, y) := 2x^4 + 2x^3y - x^2y^2 + 5y^4$  SOS ?

$$p(x, y) = \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}^T \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}$$

that is  $p(x, y) = q_{11}x^4 + 2q_{13}x^3y + 2q_{23}xy^3 + (2q_{12} + q_{33})x^2y^2 + q_{22}y^4$

hence  $q_{11} = 2$ ,  $2q_{13} = 2$ ,  $2q_{23} = 0$ ,  $2q_{12} + q_{33} = -1$ ,  $q_{22} = 5$ .

For instance

$$Q = \begin{bmatrix} 2 & -3 & 1 \\ -3 & 5 & 0 \\ 1 & 0 & 5 \end{bmatrix} = L^T L \quad L = \frac{1}{\sqrt{2}} \begin{bmatrix} 2 & -3 & 1 \\ 0 & 1 & 3 \end{bmatrix}$$

hence  $p(x, y) = \frac{1}{2} (2x^2 - 3y^2 + xy)^2 + \frac{1}{2} (y^2 + 3xy)^2$ .

# Agenda

- 1 Sum of Squares (SOS) Polynomials
- 2 Numerical Verification**
- 3 Formalization & Reflexive Tactic
- 4 Benchmarks
- 5 Conclusion



# SOS: Using approximate SDP solvers

Result  $Q$  from SDP solver will only satisfy equality constraints up to some error  $\delta$

$$p = z^T Q z + z^T E z, \quad \forall i j, |E_{i,j}| \leq \delta.$$

# SOS: Using approximate SDP solvers

Result  $Q$  from SDP solver will only satisfy equality constraints up to some error  $\delta$

$$p = z^T Q z + z^T E z, \quad \forall i, j, |E_{i,j}| \leq \delta.$$

If  $Q + E \succeq 0$  then  $p = z^T (Q + E) z$  is SOS.

## SOS: Using approximate SDP solvers

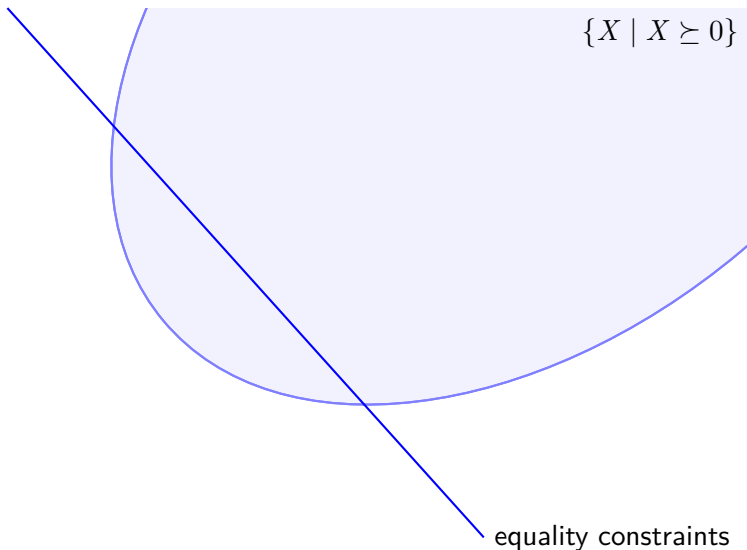
Result  $Q$  from SDP solver will only satisfy equality constraints up to some error  $\delta$

$$p = z^T Q z + z^T E z, \quad \forall i, j, |E_{i,j}| \leq \delta.$$

If  $Q + E \succeq 0$  then  $p = z^T (Q + E) z$  is SOS.

- Hence the validation method: given  $p \simeq z^T Q z$ 
    - ① Check that all monomials of  $p$  are in  $z z^T$ .
    - ② Bound difference  $\delta$  between coefficients of  $p$  and  $z^T Q z$ .
    - ③ If  $Q - s \delta I \succeq 0$  ( $s := \text{size of } Q$ ), then  $p$  is proved SOS.
  - 2 can be done with interval arithmetic and 3 with a Cholesky decomposition ( $\Theta(s^3)$  flops).
- ⇒ Efficient validation method using just floats.

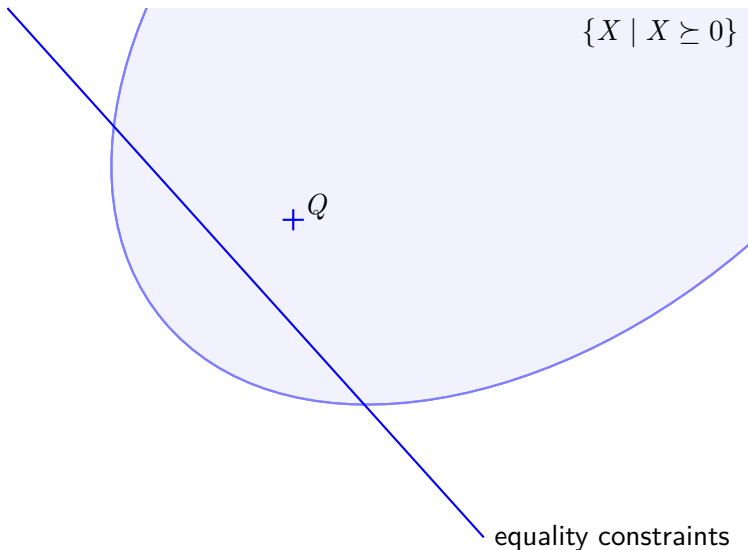
# Intuitively



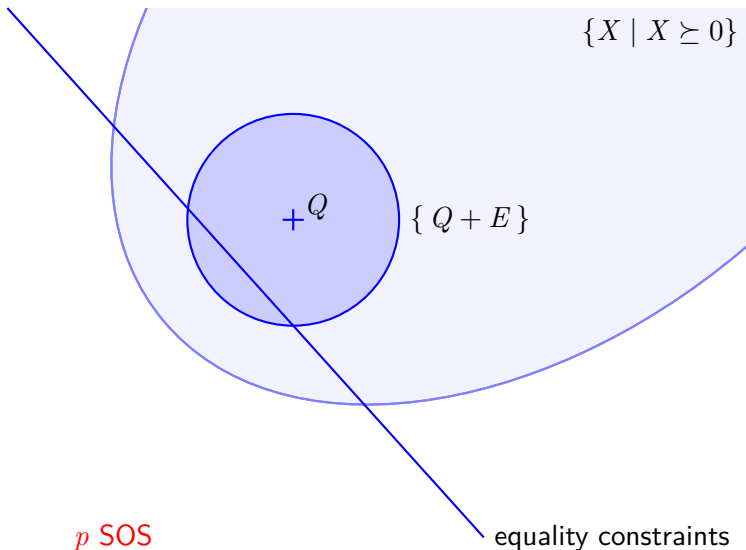
$$\{X \mid X \succeq 0\}$$

equality constraints

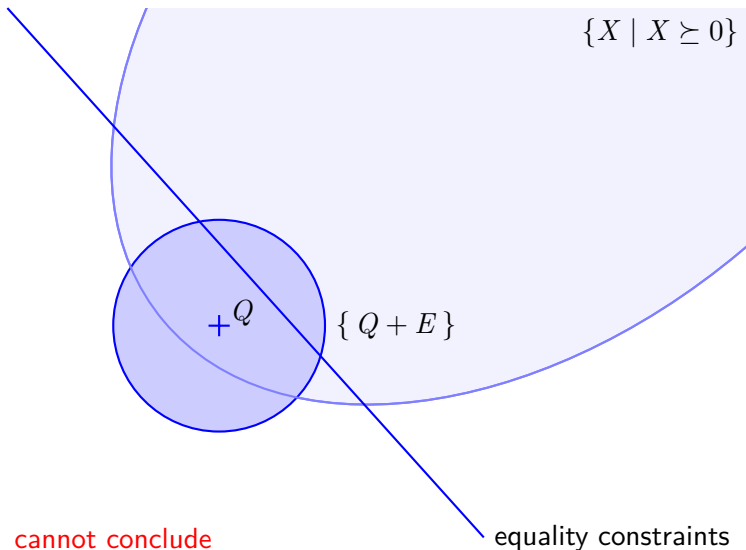
# Intuitively



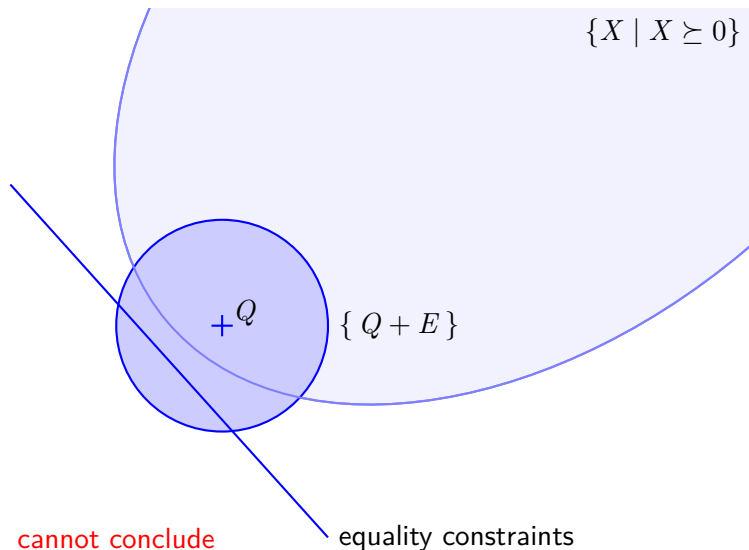
# Intuitively



# Intuitively



# Intuitively





# Cholesky Decomposition

- To prove that  $a \in \mathbb{R}$  is non negative, we can exhibit  $r$  such that  $a = r^2$  (typically  $r = \sqrt{a}$ ).

# Cholesky Decomposition

- To prove that  $a \in \mathbb{R}$  is non negative, we can exhibit  $r$  such that  $a = r^2$  (typically  $r = \sqrt{a}$ ).
- To prove that a matrix  $A \in \mathbb{R}^{n \times n}$  is positive semi-definite we can similarly expose  $R$  such that  $A = R^T R$  (since  $x^T (R^T R) x = (Rx)^T (Rx) = \|Rx\|_2^2 \geq 0$ ).

# Cholesky Decomposition

- To prove that  $a \in \mathbb{R}$  is non negative, we can exhibit  $r$  such that  $a = r^2$  (typically  $r = \sqrt{a}$ ).
- To prove that a matrix  $A \in \mathbb{R}^{n \times n}$  is positive semi-definite we can similarly expose  $R$  such that  $A = R^T R$  (since  $x^T (R^T R) x = (Rx)^T (Rx) = \|Rx\|_2^2 \geq 0$ ).
- The Cholesky decomposition computes such a matrix  $R$ :

$R := 0$ ;

**for**  $j$  **from** 1 **to**  $n$  **do**

**for**  $i$  **from** 1 **to**  $j - 1$  **do**

$$R_{i,j} := \left( A_{i,j} - \sum_{k=1}^{i-1} R_{k,i} R_{k,j} \right) / R_{i,i}$$

**od**

$$R_{j,j} := \sqrt{M_{j,j} - \sum_{k=1}^{j-1} R_{k,j}^2};$$

**od**

# Cholesky Decomposition (end)

With rounding errors  $A \neq R^T R$ , Cholesky can succeed while  $A \not\geq 0$ .

## Cholesky Decomposition (end)

With rounding errors  $A \neq R^T R$ , Cholesky can succeed while  $A \not\geq 0$ .

But error is bounded and for some (tiny)  $c \in \mathbb{R}$ :  
if Cholesky succeeds on  $A$  then  $A + cI \succeq 0$ .

Hence:

### Theorem

If Cholesky succeeds on  $A - cI$  then  $A \succeq 0$

holds for any  $c \geq \frac{(s+1)\varepsilon}{1-(2s+2)\varepsilon} \text{tr}(A) + 4(s+1) \left( 2(s+2) + \max_i(A_{i,i}) \right) \eta$   
( $\varepsilon$  and  $\eta$  relative and absolute precision of floating-point format).

Proved in Coq (paper proof: 6 pages, Coq: 5.1 kloc)

# Agenda

- 1 Sum of Squares (SOS) Polynomials
- 2 Numerical Verification
- 3 Formalization & Reflexive Tactic**
- 4 Benchmarks
- 5 Conclusion

# Outline of the formalization

# Outline of the formalization

- 1 Effective multivariate polynomials
  - CoqEAL [Cano, Cohen, Dénès, Mörtberg, Rouhling, Siles]
  - ↪ uses SSReflect and MathComp [Gonthier et al.]
  - proof: SsrMultinomials [Strub]
  - implem.: FMapAVL from Coq stdlib
  - coefficients:  $\mathbb{Q}$  as bigQ from Coq stdlib



# Outline of the formalization

- 1 Effective multivariate polynomials
  - CoqEAL [Cano, Cohen, Dénès, Mörtberg, Rouhling, Siles]
  - ↪ uses SSReflect and MathComp [Gonthier et al.]
  - proof: SsrMultinomials [Strub]
  - implem.: FMapAVL from Coq stdlib
  - coefficients:  $\mathbb{Q}$  as bigQ from Coq stdlib
- 2 Effective check for positive definite matrices
  - CoqEAL
  - proof: previous work
  - implem.: lists of lists, CoqEAL
  - coefficients: floating-point numbers from CoqInterval [Melquiond]

# Outline of the formalization

- ① Effective multivariate polynomials
  - CoqEAL [Cano, Cohen, Dénès, Mörtberg, Rouhling, Siles]
  - ↪ uses SSReflect and MathComp [Gonthier et al.]
  - proof: SsrMultinomials [Strub]
  - implem.: FMapAVL from Coq stdlib
  - coefficients:  $\mathbb{Q}$  as bigQ from Coq stdlib
- ② Effective check for positive definite matrices
  - CoqEAL
  - proof: previous work
  - implem.: lists of lists, CoqEAL
  - coefficients: floating-point numbers from CoqInterval [Melquiond]
- ③ Reflexive tactic
  - OCaml code as a wrapper for SDP solvers
  - Some Ltac code

# Refinement proofs: overview of CoqEAL's methodology

- 1 Implement algorithms in a general way  
(polymorphic functions; type classes)

# Refinement proofs: overview of CoqEAL's methodology

- 1 Implement algorithms in a general way (polymorphic functions; type classes)
- 2 Specialize the algorithms with proof-oriented datatypes; correctness proof w.r.t a specification

$$\begin{array}{c} x : A \\ \downarrow g_A \\ g_A x \end{array}$$

# Refinement proofs: overview of CoqEAL's methodology

- 1 Implement algorithms in a general way  
(polymorphic functions; type classes)
- 2 Specialize the algorithms with proof-oriented datatypes;  
correctness proof w.r.t a specification  
(or w.r.t another algorithm  $\rightsquigarrow$  program refinement)

$$\begin{array}{ccc} x & = & x : A \\ \downarrow f & & \downarrow g_A \\ f x & = & g_A x \end{array}$$

# Refinement proofs: overview of CoqEAL's methodology

- 1 Implement algorithms in a general way  
(polymorphic functions; type classes)
- 2 Specialize the algorithms with proof-oriented datatypes;  
correctness proof w.r.t a specification  
(or w.r.t another algorithm  $\rightsquigarrow$  program refinement)
- 3 Specialize the algorithms with effective datatypes;

$$\begin{array}{ccc} x & = & x : A \\ \downarrow f & & \downarrow g_A \\ f x & = & g_A x \end{array} \qquad \begin{array}{c} c : C \\ \downarrow g_C \\ g_C c \end{array}$$

# Refinement proofs: overview of CoqEAL's methodology

- 1 Implement algorithms in a general way  
(polymorphic functions; type classes)
- 2 Specialize the algorithms with proof-oriented datatypes;  
correctness proof w.r.t a specification  
(or w.r.t another algorithm  $\rightsquigarrow$  program refinement)
- 3 Specialize the algorithms with effective datatypes;  
correctness proof w.r.t proof-oriented version ( $\rightsquigarrow$  data refinement)

$$\begin{array}{ccccc}
 x & = & x : A & \xleftrightarrow{\text{refines}} & c : C \\
 \downarrow f & & \downarrow g_A & & \downarrow g_C \\
 f x & = & g_A x & \xleftrightarrow{\text{refines}} & g_C c
 \end{array}$$

# Effective multivariate polynomials

- Implemented in a modular way:

```
Definition seqmultinom := list N.
```

```
Module MultinomOrd <: OrderedType.
```

```
  Definition t := seqmultinom. (*...*) End MultinomOrd.
```

```
Module FMapMultiply (M : Sfun MultinomOrd).
```

```
  Definition effmpoly := M.t. (*...*) End FMapMultiply.
```

```
Module M := FMapAVL.Make MultinomOrd.
```

```
Module PolyAVL := FMapMultiply M.
```



# Effective multivariate polynomials

- Implemented in a modular way:

```
Definition seqmultinom := list N.
```

```
Module MultinomOrd <: OrderedType.
```

```
  Definition t := seqmultinom. (*...*) End MultinomOrd.
```

```
Module FMapMultiply (M : Sfun MultinomOrd).
```

```
  Definition effmpoly := M.t. (*...*) End FMapMultiply.
```

```
Module M := FMapAVL.Make MultinomOrd.
```

```
Module PolyAVL := FMapMultiply M.
```

- Main refinement predicates:

Rseqmultinom :  $\forall (n : \text{nat}), \text{multinom } n \rightarrow \text{seqmultinom} \rightarrow \text{Type}$

Reffmpoly :  $\forall (T : \text{ringType}) (n : \text{nat}), \text{mpoly } n T \rightarrow \text{effmpoly } T \rightarrow \text{Type}$

ReffmpolyC :  $\forall (A : \text{ringType}) (C : \text{Type}), (A \rightarrow C \rightarrow \text{Type}) \rightarrow$

$\forall (n : \text{nat}), \text{mpoly } n A \rightarrow \text{effmpoly } C \rightarrow \text{Type}$

# Effective multivariate polynomials

- Implemented in a modular way:

**Definition** seqmultinom := list N.

**Module** MultinomOrd <: OrderedType.

**Definition** t := seqmultinom. (\*...\*) **End** MultinomOrd.

**Module** FMapMultiply (M : Sfun MultinomOrd).

**Definition** effmpoly := M.t. (\*...\*) **End** FMapMultiply.

**Module** M := FMapAVL.Make MultinomOrd.

**Module** PolyAVL := FMapMultiply M.

- Main refinement predicates:

**Rseqmultinom** :  $\forall (n : \text{nat}), \text{multinom } n \rightarrow \text{seqmultinom} \rightarrow \text{Type}$

**Reffmpoly** :  $\forall (T : \text{ringType}) (n : \text{nat}), \text{mpoly } n T \rightarrow \text{effmpoly } T \rightarrow \text{Type}$

**ReffmpolyC** :  $\forall (A : \text{ringType}) (C : \text{Type}), (A \rightarrow C \rightarrow \text{Type}) \rightarrow$

$\forall (n : \text{nat}), \text{mpoly } n A \rightarrow \text{effmpoly } C \rightarrow \text{Type}$

- Proof-oriented** type for coefficients: needs a ringType structure; instantiated with MathComp's rat. **Effective** counterpart: bigQ.

# Positive definiteness check for floating-point matrices

**Definition** posdef (n : nat) (A : 'M[R]\_n) :=  
 $\forall (x : 'cV[R]_n), x \neq 0 \rightarrow 0 < x^T \times A \times x. (* \text{pointwise } < *)$

# Positive definiteness check for floating-point matrices

**Definition** posdef (n : nat) (A : 'M[R]\_n) :=  
 $\forall (x : 'cV[R]_n), x \neq 0 \rightarrow 0 < x^T \times A \times x.$  (\* pointwise < \*)

posdef\_check :  $\forall (mx : \text{Type} \rightarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{Type})$   
 (T : Type) (n : nat),  
 (\*...type classes...→ \*)  
 mx T n n  $\rightarrow$  bool

# Positive definiteness check for floating-point matrices

**Definition** posdef (n : nat) (A : 'M[R]\_n) :=  
 $\forall (x : 'cV[R]_n), x \neq 0 \rightarrow 0 < x^T \times A \times x.$  (\* pointwise < \*)

posdef\_check :  $\forall (mx : \text{Type} \rightarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{Type})$   
 (T : Type) (n : nat),  
 (\*...type classes...→ \*)  
 mx T n n  $\rightarrow$  bool

- Correctness: use formal proof of Cholesky algo over  $\mathbb{R}$  (previous work)

# Positive definiteness check for floating-point matrices

**Definition** posdef (n : nat) (A : 'M[R]\_n) :=  
 $\forall (x : 'cV[R]_n), x \neq 0 \rightarrow 0 < x^T \times A \times x.$  (\* pointwise < \*)

posdef\_check :  $\forall (mx : \text{Type} \rightarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{Type})$   
 (T : Type) (n : nat),  
 (\*...type classes...→ \*)  
 mx T n n → bool

- Correctness: use formal proof of Cholesky algo over  $\mathbb{R}$  (previous work)
- Refinement 1: refine dependently-typed matrices with list-based ones

# Positive definiteness check for floating-point matrices

**Definition** posdef (n : nat) (A : 'M[R]\_n) :=  
 $\forall (x : 'cV[R]_n), x \neq 0 \rightarrow 0 < x^T \times A \times x.$  (\* pointwise < \*)

posdef\_check :  $\forall (mx : \text{Type} \rightarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{Type})$   
 (T : Type) (n : nat),  
 (\*...type classes...→ \*)  
 mx T n n  $\rightarrow$  bool

- Correctness: use formal proof of Cholesky algo over  $\mathbb{R}$  (previous work)
- Refinement 1: refine dependently-typed matrices with list-based ones
- Refinement 2: refine real coefficients with floating-point ones:

# Positive definiteness check for floating-point matrices

**Definition** posdef (n : nat) (A : 'M[R]\_n) :=  
 $\forall (x : 'cV[R]_n), x \neq 0 \rightarrow 0 < x^T \times A \times x.$  (\* pointwise < \*)

posdef\_check :  $\forall (mx : \text{Type} \rightarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{Type})$   
 (T : Type) (n : nat),  
 (\*...type classes...→ \*)  
 mx T n n  $\rightarrow$  bool

- Correctness: use formal proof of Cholesky algo over  $\mathbb{R}$  (previous work)
- Refinement 1: refine dependently-typed matrices with list-based ones
- Refinement 2: refine real coefficients with floating-point ones:
  - Rely on Float\_infnan\_spec



# Positive definiteness check for floating-point matrices

**Definition** posdef (n : nat) (A : 'M[R]\_n) :=  
 $\forall (x : 'cV[R]_n), x \neq 0 \rightarrow 0 < x^T \times A \times x.$  (\* pointwise < \*)

posdef\_check :  $\forall (mx : \text{Type} \rightarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{Type})$   
 (T : Type) (n : nat),  
 (\*...type classes...→ \*)  
 mx T n n → bool

- Correctness: use formal proof of Cholesky algo over  $\mathbb{R}$  (previous work)
- Refinement 1: refine dependently-typed matrices with list-based ones
- Refinement 2: refine real coefficients with floating-point ones:
  - Rely on `Float_infnan_spec`
  - = formalization of the floating-point “standard model” (previous work)

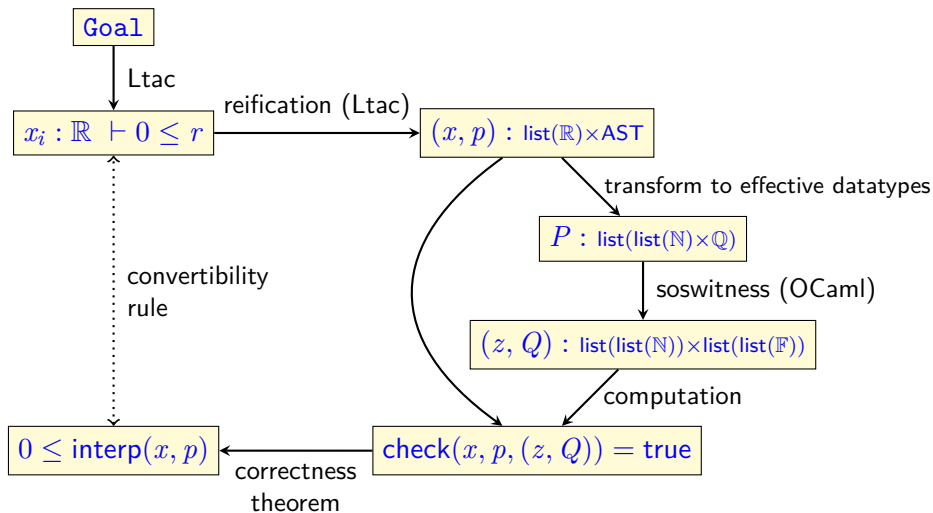
# Positive definiteness check for floating-point matrices

**Definition** posdef (n : nat) (A : 'M[R]\_n) :=  
 $\forall (x : 'cV[R]_n), x \neq 0 \rightarrow 0 < x^T \times A \times x$ . (\* pointwise < \*)

posdef\_check :  $\forall (mx : \text{Type} \rightarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{Type})$   
 (T : Type) (n : nat),  
 (\*...type classes...→ \*)  
 mx T n n → bool

- Correctness: use formal proof of Cholesky algo over  $\mathbb{R}$  (previous work)
- Refinement 1: refine dependently-typed matrices with list-based ones
- Refinement 2: refine real coefficients with floating-point ones:
  - Rely on `Float_infnan_spec`
  - = formalization of the floating-point “standard model” (previous work)
  - instantiated with `CoqInterval`'s floating-point implementation, restricted to 53 bits.

# The validsdp tactic (1/3) – the big picture



## The validsdp tactic (2/3) – OCaml code

- Rely on the OSDP lib. (OCaml interface for off-the-shelf SDP solvers)
- Implement a Coq plugin (the `ValidSDP.soswitness` OCaml module provides a `soswitness` tactic that consists of a wrapper for OSDP)

OSDP library: 6.2 kloc of OCaml code + 1.2 kloc of C code.

`ValidSDP.soswitness` plugin: 0.3 kloc of OCaml code.

# The validsdp tactic (3/3) – correctness theorem

**Theorem soscheck\_eff\_wrapup\_correct :**

$$\begin{aligned} &\forall (x : \text{list } R) (p : \text{p\_abstr\_poly}) \\ &\quad (zQ : \text{list } (\text{list } N) * \text{list } (\text{list } (\text{s\_float } \text{bigZ } \text{bigZ}))), \\ &\text{soscheck\_eff\_wrapup } x \text{ } p \text{ } zQ = \text{true} \rightarrow \\ &(0 \leq \text{interp\_p\_abstr\_poly } x \text{ } p)\%R. \end{aligned}$$

Coq: 2.0 kloc for the main tactic and proofs + 6.5 kloc of refinement proofs  
(Cholesky: 3.0 kloc; FP arith: 1.3 kloc; multipoly: 2.2 kloc)

## Goals accepted by validsdp-0.3

- Inequalities between polynomial expressions
  - over real variables
  - with rational coefficients
  - built from  $+$ ,  $-$ ,  $\times$  and constant exponents

# Goals accepted by latest validsdp

Goals of the form

$x_1, x_2, \dots : \mathbb{R}$

=====

$g_1 \ x_1 \ x_2 \ \dots \ \begin{array}{l} \diagup \\ \diagdown \\ \diagup \\ \diagdown \end{array} \ h_1 \ x_1 \ x_2 \ \dots \rightarrow$

$g_2 \ x_1 \ x_2 \ \dots \ \begin{array}{l} \diagup \\ \diagdown \\ \diagup \\ \diagdown \end{array} \ h_2 \ x_1 \ x_2 \ \dots \rightarrow$

$\dots \dots \dots \dots \dots \dots \dots \rightarrow$

$p \ x_1 \ x_2 \ \dots \ \begin{array}{l} \diagup \\ \diagdown \\ \diagup \\ \diagdown \\ \diagup \\ \diagdown \end{array} \ q \ x_1 \ x_2 \ \dots$

assuming the polynomials  $g_i, h_i, p, q$  have been unfolded.

# Agenda

- 1 Sum of Squares (SOS) Polynomials
- 2 Numerical Verification
- 3 Formalization & Reflexive Tactic
- 4 Benchmarks**
- 5 Conclusion



# Benchmarks (1/3)

- In our CPP'2017 paper : OSDP 0.4.5 & ValidSDP 0.3
- In this presentation : OSDP 0.5.2 & ValidSDP version d60c663
- In both configurations: Coq 8.5.2, MathComp 1.6, Flocq 2.5.1, Coquelicot 2.1.1, CoqInterval 3.1.0, and dev. version of other libs
- Setup:
  - A desktop PC under Debian GNU/Linux Jessie
  - Core i5-4460S CPU clocked at 2.9 GHz
  - All timings are total elapsed time (in seconds)
  - Timeout of 900s

## Benchmarks (2/3)

OSDP (not verified)  
MonniauxC11 (not verified)  
NLCertify (not verified)  
ValidSDP  
PVS/Bernstein  
NLCertify  
HOL Light/Taylor

Problem	$n$	$d$	OSDP (not verified)	MonniauxC11 (not verified)	NLCertify (not verified)	ValidSDP	PVS/Bernstein	NLCertify	HOL Light/Taylor
adaptativeLV	4	4	<b>0.75</b>	2.67	1.12	5.16	14.93	<b>2.61</b>	12.31
butcher	6	4	1.58	—	<b>1.05</b>	9.40	48.44	<b>8.36</b>	15.62
caprasse	4	4	<b>0.41</b>	1.82	0.88	<b>5.19</b>	25.89	2.63	17.68
heart	8	4	<b>3.18</b>	268.75	—	<b>16.67</b>	131.13	—	26.15
magnetism	7	2	<b>1.11</b>	2.04	1.64	<b>5.18</b>	245.52	14.50	16.07
reaction	3	2	<b>0.81</b>	1.56	0.24	4.33	11.48	<b>1.96</b>	12.41
schwefel	3	4	<b>0.95</b>	2.45	2.76	<b>3.70</b>	14.72	56.13	17.46
fs260	6	4	<b>1.25</b>	—	—	<b>5.99</b>	—	—	—
fs461	6	4	<b>0.70</b>	11.18	0.87	<b>5.18</b>	621.06	7.46	22.70
fs491	6	4	<b>0.54</b>	21.81	—	<b>5.38</b>	—	—	—
fs745	6	4	0.98	11.74	<b>0.94</b>	<b>5.55</b>	623.17	6.90	22.48
fs752	6	2	<b>0.35</b>	1.81	0.90	<b>3.80</b>	54.52	7.88	13.34
fs8	6	2	<b>0.43</b>	1.53	1.48	<b>3.93</b>	52.63	6.62	13.40
fs859	6	8	—	—	—	—	—	—	—
fs860	6	4	<b>1.21</b>	10.53	1.11	<b>6.08</b>	73.65	7.34	14.28
fs861	6	4	<b>1.09</b>	10.48	1.20	<b>5.15</b>	69.74	7.87	14.28
fs862	6	4	1.27	79.25	<b>1.25</b>	<b>5.37</b>	73.54	7.58	14.14
fs863	6	2	<b>0.94</b>	1.50	—	<b>3.85</b>	—	—	13.85
fs864	6	2	<b>0.56</b>	2.05	—	<b>4.05</b>	—	—	13.28
fs865	6	2	<b>0.76</b>	2.11	—	<b>3.68</b>	—	—	13.76
fs867	6	2	<b>0.21</b>	2.09	1.74	<b>4.22</b>	—	8.04	—

## Benchmarks (3/3)

OSDP (not verified)  
 MonniauxC11 (not verified)  
 NLCertify (not verified)  
 ValidSDP  
 PVS/Bernstein  
 NLCertify  
 HOL Light/  
 Taylor

Problem	$n$	$d$	OSDP (not verified)	MonniauxC11 (not verified)	NLCertify (not verified)	ValidSDP	PVS/Bernstein	NLCertify	HOL Light/ Taylor
fs868	6	4	<b>0.94</b>	—	—	<b>6.05</b>	—	—	—
fs884	6	4	—	—	—	—	—	—	—
fs890	6	4	—	<b>7.78</b>	—	—	—	—	—
ex4_d4	2	12	—	—	—	—	—	—	—
ex4_d6	2	18	—	—	—	—	—	—	—
ex4_d8	2	24	<b>16.99</b>	—	—	<b>82.89</b>	—	—	—
ex4_d10	2	30	—	—	—	—	—	—	—
ex5_d4	3	8	<b>1.67</b>	—	—	<b>13.63</b>	—	—	—
ex5_d6	3	12	<b>16.10</b>	—	—	<b>66.82</b>	—	—	—
ex5_d8	3	16	<b>203.06</b>	—	—	<b>353.70</b>	—	—	—
ex5_d10	3	20	—	—	—	—	—	—	—
ex6_d4	4	8	<b>16.82</b>	—	—	<b>44.99</b>	—	—	—
ex6_d6	4	12	—	—	—	—	—	—	—
ex7_d4	2	12	—	—	—	—	—	—	—
ex7_d6	2	18	<b>1.50</b>	—	—	<b>26.78</b>	—	—	—
ex7_d8	2	24	<b>15.38</b>	—	—	<b>83.47</b>	—	—	—
ex7_d10	2	30	—	—	—	—	—	—	—
ex8_d4	2	8	<b>0.87</b>	15.72	—	<b>7.52</b>	—	—	—
ex8_d6	2	12	—	—	—	—	—	—	—
ex8_d8	2	16	—	—	—	—	—	—	—
ex8_d10	2	20	—	—	—	—	—	—	—

# Agenda

- 1 Sum of Squares (SOS) Polynomials
- 2 Numerical Verification
- 3 Formalization & Reflexive Tactic
- 4 Benchmarks
- 5 Conclusion

# Conclusion

- Context: formal proof of multivariate polynomial positivity
- A Coq reflexive tactic
  - Input: polynomial ineq. goals with real variables and rational constants
  - Use off-the-shelf SDP solvers as untrusted oracles
  - Numerical approach with formal floating-point arithmetic
  - Algorithm involving matrices (Cholesky)

# Discussion and open questions

- Bottleneck 1:  
emulate floating-point arithmetic in Coq (1000x overhead)
- Bottleneck 2: we currently use `vm_compute`.  
`native_compute` issue with large terms (witnesses for large problems)
- Coq 8.6 comes with LtacProf  $\rightsquigarrow$  feasibility of a 'vm\_compute profiler'?
- Possible extension of ValidSDP: add support of elementary functions

# Thank you!

## Questions



<https://sourcesup.renater.fr/validsdp/>