Numerical Computations and Proofs

from Proof-Assistants to Aerospace Applications

Pierre Roux
ONERA, Toulouse

December 1st 2025

Defense for the HDR title issued by Toulouse INP, jury:

- Andrew Appel, Emeritus prof. Princeton, visiting Cornell (reviewer)
- Assia Mahboubi, DR INRIA Nantes (reviewer)
- David Monniaux, DR CNRS Grenoble (reviewer)
- Yves Bertot, DR INRIA Nice (examiner)
- Sylvie Boldo, DR INRIA Saclay (examiner)
- Emmanuel Grolleau, Prof. ENSMA (examiner)
- Didier Henrion, DR CNRS Toulouse (examiner)
- Philippe Queinnec, Prof. Toulouse INP (examiner)

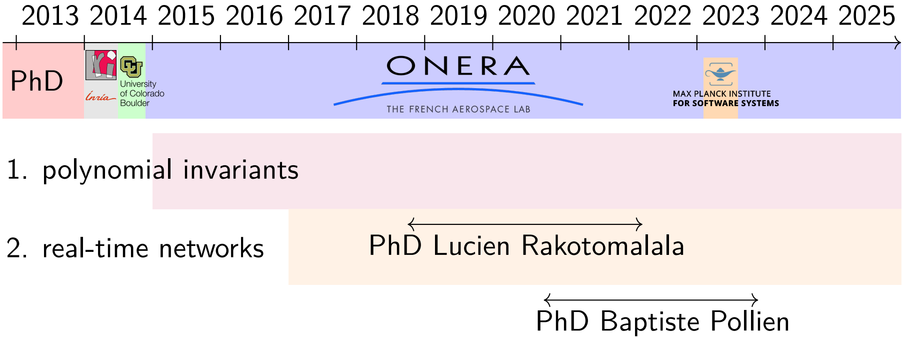
Overview


2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025
PhD
Inria
University
of Colorado
Boulder
ONERA
THE FRENCH AEROSPACE LAB
MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

Overview

1. polynomial invariants
2. real-time networks

Overview



- publications: conferences (24), journals (6)
- teaching: lecture and tutorials (cours, TD, TP) on programming (functional, imperative, OO), Rocq, process algebras and abstract interpretation (≃50 hours per year)
- projects: SEFA IKKY (DGAC, 2016-18), IREHDO2 (DGAC, 2016-18), TTE (CNES, 2016-17), Valencia (DGA, 2017-20, coordinator), RT-proofs (ANR-DFG, 2018-22, coordinator), Concorde (AID, 2020-23), Accord (DGAC, 2022-24)

Digital Flight Commands 1/2

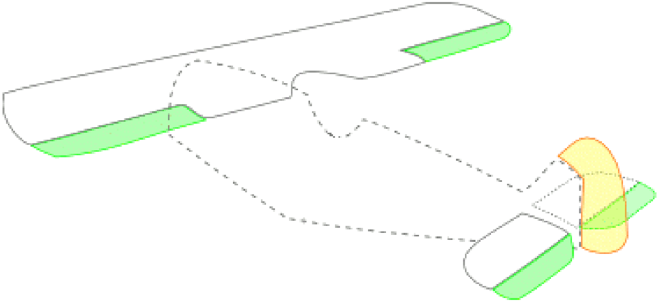


Image: Piotr Jaworski (GFDL)

Digital Flight Commands 1/2

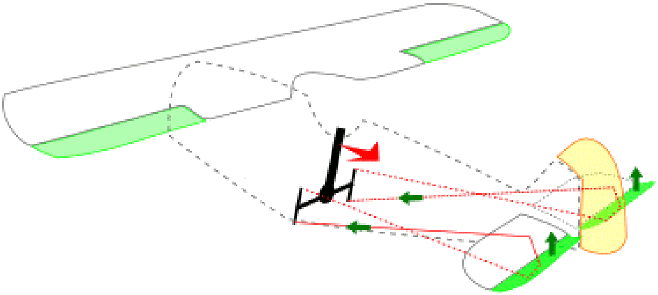


Image: Piotr Jaworski (GFDL)

Cables

Image: public domain

Digital Flight Commands 1/2

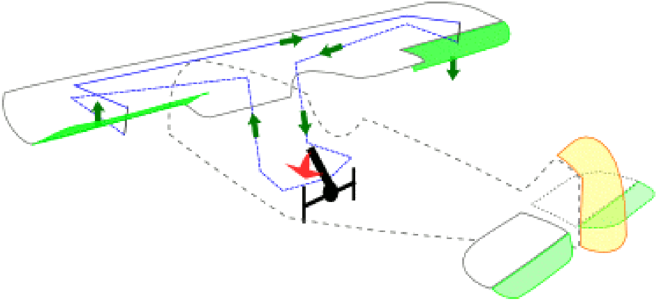


Image: Piotr Jaworski (GFDL)

Cables

Image: public domain

Digital Flight Commands 1/2

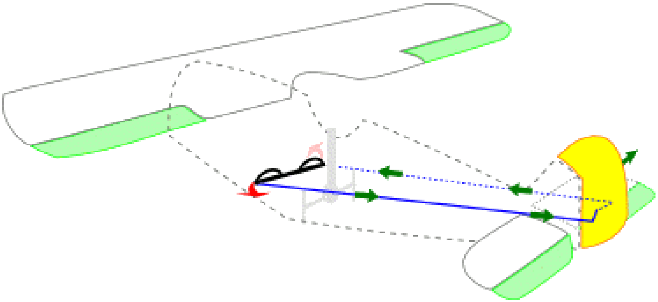


Image: Piotr Jaworski (GFDL)

Cables

Image: public domain

Digital Flight Commands 1/2

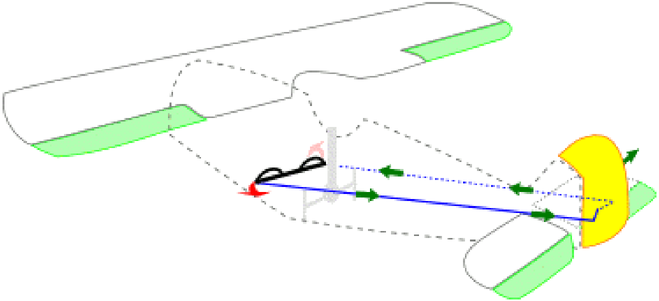


Image: Piotr Jaworski (GFDL)

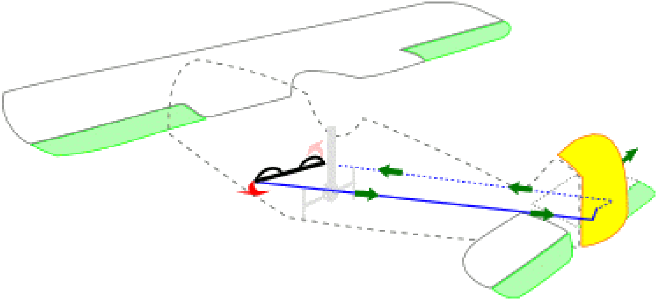
Cables

Image: public domain

Hydraulic Actuators

Image: Woodward

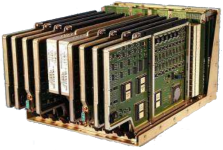
Digital Flight Commands 1/2



Cables

Hydraulic Actuators

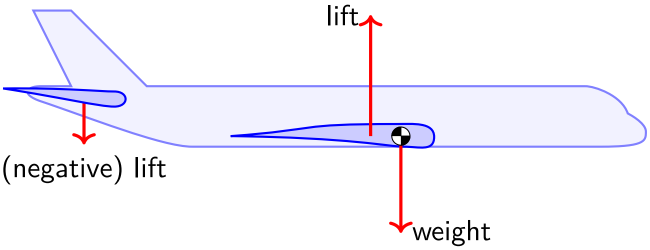
Computers



Digital Flight Commands 2/2

Digital Flight Commands

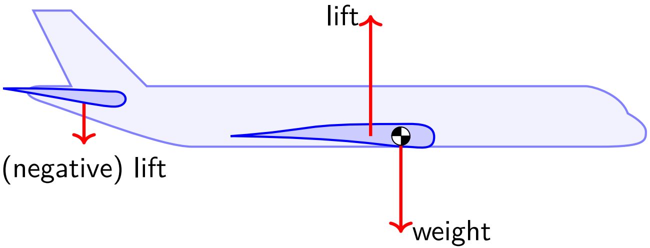
- Improve comfort
- Enable different aircrafts to feel similar (optimizing pilots training)
- Improve safety, by preventing dangerous attitude / efforts e.g., aircraft cannot stall
- Improve fuel efficiency by enabling smaller stability margins



Digital Flight Commands 2/2

Digital Flight Commands

- Improve comfort
- Enable different aircrafts to feel similar (optimizing pilots training)
- Improve safety, by preventing dangerous attitude / efforts e.g., aircraft cannot stall
- Improve fuel efficiency by enabling smaller stability margins



⇒ Those are critical systems

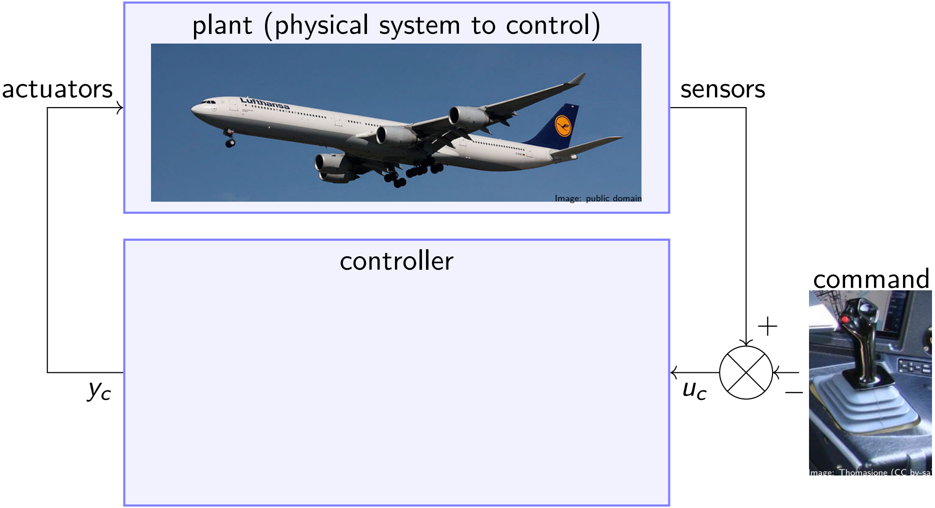
⇒ We want some guarantees on their correctness

Control Command Systems

plant (physical system to control)

command

Control Command Systems



Control Command Systems

plant (physical system to control)

actuators

sensors

controller

```
double x[3] = {0, 0, 0};
double nx[3];
double in;
while (1) {
  in = acquire_input();  // u_c
  nx[0] = 0.9379*x[0]-0.0381*x[1]-0.0414*x[2]+0.0237*in;
  nx[1] = -0.0404*x[0]+0.968*x[1]-0.0179*x[2]+0.0143*in;
  nx[2] = 0.0142*x[0]-0.0197*x[1]+0.9823*x[2]+0.0077*in;
  x[0] = nx[0]; x[1] = nx[1]; x[2] = nx[2];
  send_output(x);  // y_c
  wait_next_clock_tick();  // a tick every 10 ms
}
```

y_c

u_c

$+$

$-$

command

Control Command Systems

plant (physical system to control)

Image: public domain

actuators

sensors

controller

```
double x[3] = {0, 0, 0};
double nx[3];
double in;
while (1) {
  in = acquire_input();  // u_c
  nx[0] = 0.9379*x[0]-0.0381*x[1]-0.0414*x[2]+0.0237*in;
  nx[1] = -0.0404*x[0]+0.968*x[1]-0.0179*x[2]+0.0143*in;
  nx[2] = 0.0142*x[0]-0.0197*x[1]+0.9823*x[2]+0.0077*in;
  x[0] = nx[0]; x[1] = nx[1]; x[2] = nx[2];
  send_output(x);  // y_c
  wait_next_clock_tick();  // a tick every 10 ms
}
```

y_c

u_c

$+$

$-$

command

Image: Thomasione (CC by-sa)

Control Command Systems

plant (physical system to control)

Image: public domain

actuators

sensors

controller

```
double x[3] = {0, 0, 0};
double nx[3];
double in;
while (1) {
  in = acquire_input();  // u_c
  nx[0] = 0.9379*x[0]-0.0381*x[1]-0.0414*x[2]+0.0237*in;
  nx[1] = -0.0404*x[0]+0.968*x[1]-0.0179*x[2]+0.0143*in;
  nx[2] = 0.0142*x[0]-0.0197*x[1]+0.9823*x[2]+0.0077*in;
  x[0] = nx[0]; x[1] = nx[1]; x[2] = nx[2];
  send_output(x);  // y_c
  wait_next_clock_tick();  // a tick every 10 ms
}
```

y_c

u_c

$+$

$-$

command

Image: Thomasione (CC by-sa)

Control Command Systems

plant (physical system to control)

actuators

sensors

controller

```
double x[3] = {0, 0, 0};
double nx[3];
double in;
while (1) {
  in = acquire_input();  // u_c
  nx[0] = 0.9379*x[0]-0.0381*x[1]-0.0414*x[2]+0.0237*in;
  nx[1] = -0.0404*x[0]+0.968*x[1]-0.0179*x[2]+0.0143*in;
  nx[2] = 0.0142*x[0]-0.0197*x[1]+0.9823*x[2]+0.0077*in;
  x[0] = nx[0]; x[1] = nx[1]; x[2] = nx[2];
  send_output(x);  // y_c
  wait_next_clock_tick();  // a tick every 10 ms
}
```

y_c

u_c

$+$

$-$

command

Control Command Systems

plant (physical system to control)

Image: public domain

actuators

sensors

controller

```
double x[3] = {0, 0, 0};
double nx[3];
double in;
while (1) {
  in = acquire_input();  // u_c
  nx[0] = 0.9379*x[0]-0.0381*x[1]-0.0414*x[2]+0.0237*in;
  nx[1] = -0.0404*x[0]+0.968*x[1]-0.0179*x[2]+0.0143*in;
  nx[2] = 0.0142*x[0]-0.0197*x[1]+0.9823*x[2]+0.0077*in;
  x[0] = nx[0]; x[1] = nx[1]; x[2] = nx[2];
  send_output(x);  // y_c
  wait_next_clock_tick();  // a tick every 10 ms
}
```

y_c

u_c

$+$

$-$

command

Image: Thomasione (CC by-sa)

Control Command Systems

plant (physical system to control)

actuators

sensors

controller

```
double x[3] = {0, 0, 0};
double nx[3];
double in;
while (1) {
  in = acquire_input();  // u_c
  nx[0] = 0.9379*x[0]-0.0381*x[1]-0.0414*x[2]+0.0237*in;
  nx[1] = -0.0404*x[0]+0.968*x[1]-0.0179*x[2]+0.0143*in;
  nx[2] = 0.0142*x[0]-0.0197*x[1]+0.9823*x[2]+0.0077*in;
  x[0] = nx[0]; x[1] = nx[1]; x[2] = nx[2];
  send_output(x);  // y_c
  wait_next_clock_tick();  // a tick every 10 ms
}
```

y_c

u_c

$+$

$-$

command

- we want to prove all reachable states are safe
  e.g., no combination low velocity / high angle of attack (stall)

$\Rightarrow$ main tool: loop invariant

Verifying Polynomial Invariants

Verifying Real-Time Embedded Networks

Technical Expertise

Perspectives

Static Analysis

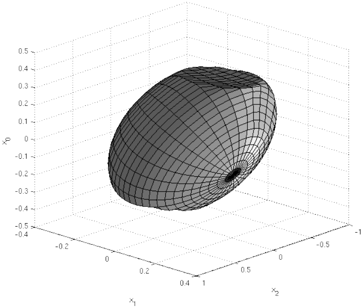
Static analyzers can infer loop invariants.

Mostly linear invariants:

- intervals
- polyhedra
- octogons
- zonotopes
- ...

Static Analysis

Static analyzers can infer loop invariants.

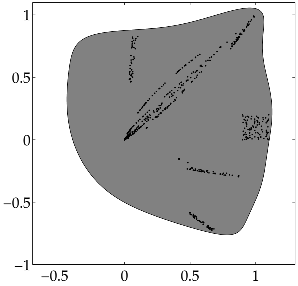


Mostly linear invariants:

- intervals
- polyhedra
- octogons
- zonotopes
- . . .

Quadratic / polynomial invariants are better suited for controllers:

- ellipsoids
- polynomial sublevel curves



Polynomial Invariants

In some paper, authors offer for

```
(x1, x2) ∈ [0.9, 1.1] × [0, 0.2]
while (1) {
  pre_x1 = x1; pre_x2 = x2;
  if (x1^2 + x2^2 <= 1) {
    x1 = pre_x1^2 + pre_x2^3;
    x2 = pre_x1^3 + pre_x2^2;
  } else {
    x1 = 0.5 * pre_x1^3 + 0.4 * pre_x2^2;
    x2 = -0.6 * pre_x1^2 + 0.3 * pre_x2^2; } }
```

the inductive invariant

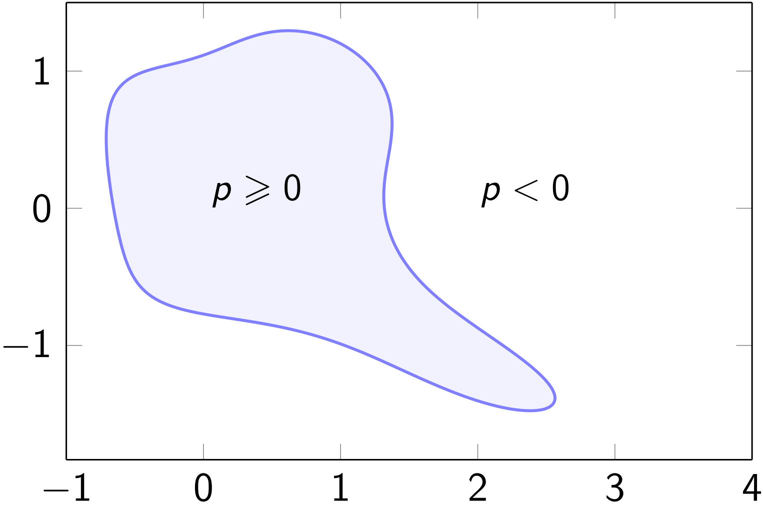
$2.510902467 + 0.0050x_1 + 0.0148x_2 - 3.0998x_1^2 + 0.8037x_2^3 + 3.0297x_1^3 - 2.5924x_2^2 - 1.5266x_1x_2 + 1.9133x_1^2x_2 + 1.8122x_1x_2^2 - 1.6042x_1^4 - 0.0512x_1^3x_2 + 4.4430x_1^2x_2^2 + 1.8926x_1x_2^3 - 0.5464x_2^4 + 0.2084x_1^5 - 0.5866x_1^4x_2 - 2.2410x_1^3x_2^2 - 1.5714x_1^2x_2^3 + 0.0890x_1x_2^4 + 0.9656x_2^5 - 0.0098x_1^6 + 0.0320x_1^5x_2 + 0.0232x_1^4x_2^2 - 0.2660x_1^3x_2^3 - 0.7746x_1^2x_2^4 - 0.9200x_1x_2^5 - 0.6411x_2^6 \geqslant 0.$

Should We Trust Such Results ?

- Some are correct (we'll prove it formally).

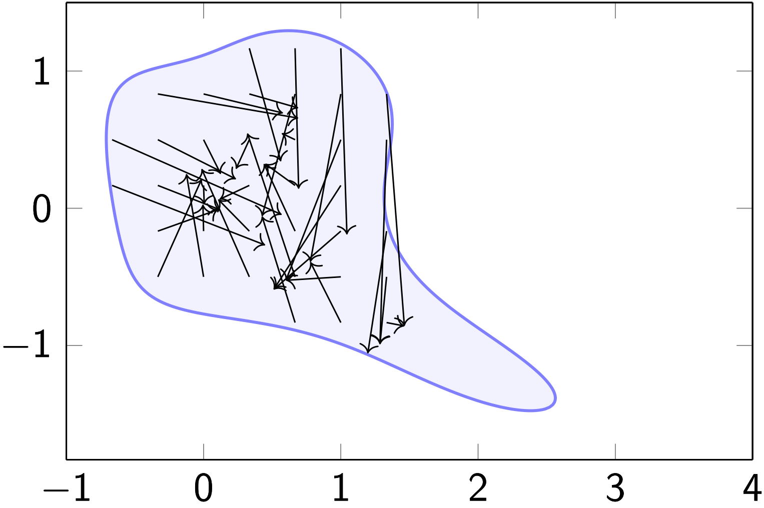
Should We Trust Such Results ?

- Some are correct (we'll prove it formally).
- Others   (previous degree 6 polynomial)



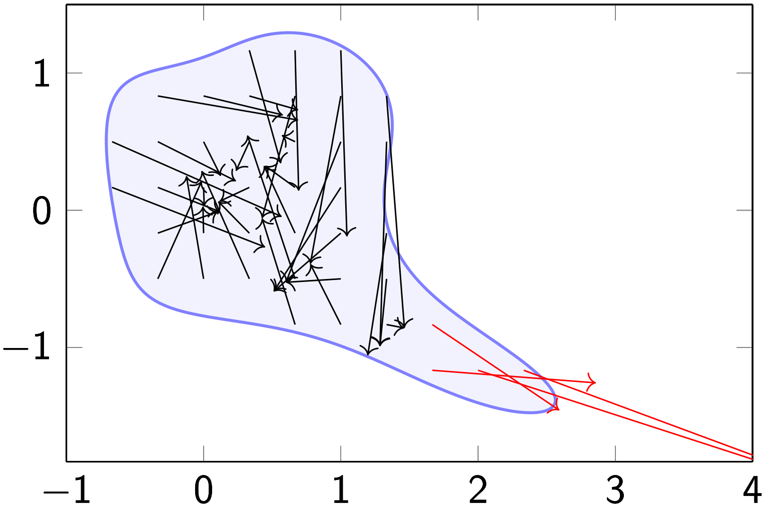
Should We Trust Such Results ?

- Some are correct (we'll prove it formally).
- Others (previous degree 6 polynomial)



Should We Trust Such Results ?

- Some are correct (we'll prove it formally).
- Others aren't (previous degree 6 polynomial)



Sum of Squares (SOS) Polynomials

Invariant checking can be reduced to
proving some polynomial p non negative.

Definition (SOS Polynomial)

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

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

- If p SOS then $p \geqslant 0$

Sum of Squares (SOS) Polynomials

Invariant checking can be reduced to
proving some polynomial p non negative.

Definition (SOS Polynomial)

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

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

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

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

$\Rightarrow$ we have solvers, called SDP (Semi-Definite Programming)

SOS: Example

Example

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

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

SDP gives

$$Q = \begin{bmatrix} 2 & -3 & 1 \\ -3 & 5 & 0 \\ 1 & 0 & 5 \end{bmatrix}$$

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

SDP gives

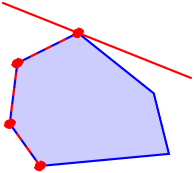
$$Q = \begin{bmatrix} 2 & -3 & 1 \\ -3 & 5 & 0 \\ 1 & 0 & 5 \end{bmatrix} = L^T L \qquad L = \frac{1}{\sqrt{2}} \begin{bmatrix} 2 & -3 & 1 \\ 0 & 1 & 3 \end{bmatrix}$$

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

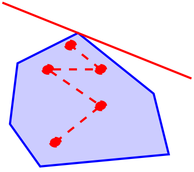
SDP Solvers Yield Approximate Solutions

- Linear programming

simplex: exact solution



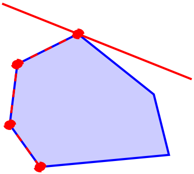
interior-point: approximate solution



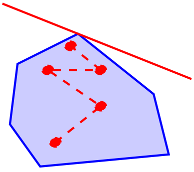
SDP Solvers Yield Approximate Solutions

- Linear programming

simplex: exact solution

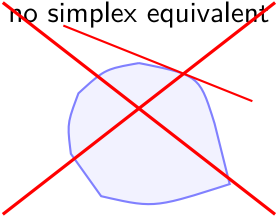


interior-point: approximate solution

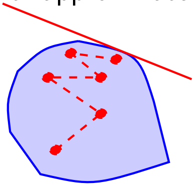


- Semidefinite programming

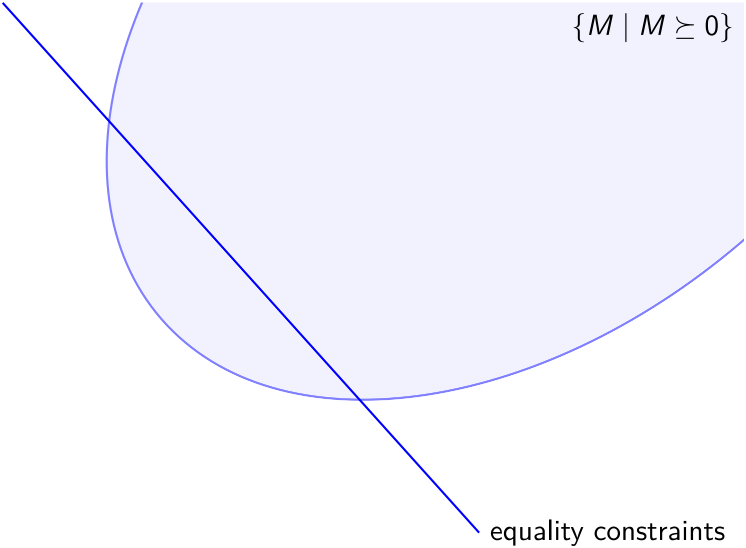
no simplex equivalent



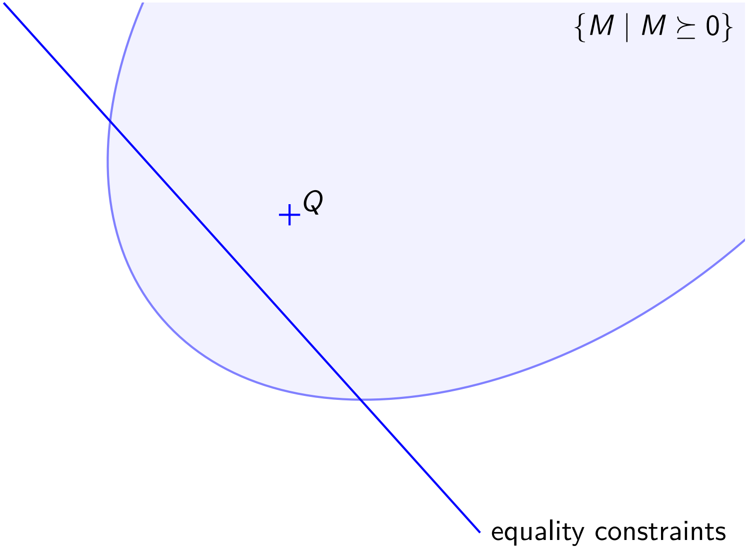
interior-point: approximate solution



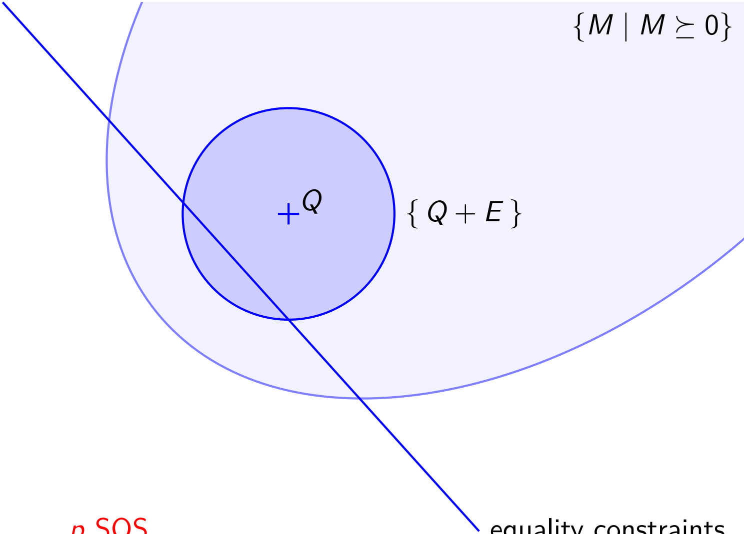
Intuitively



Intuitively



Intuitively



p SOS

equality constraints

SOS: Using Approximate SDP Solvers

Result Q from SDP solver will only satisfy equality constraints up to some error δ

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

SOS: Using Approximate SDP Solvers

Result Q from SDP solver will only satisfy equality constraints up to some error δ

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

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 δ

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

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

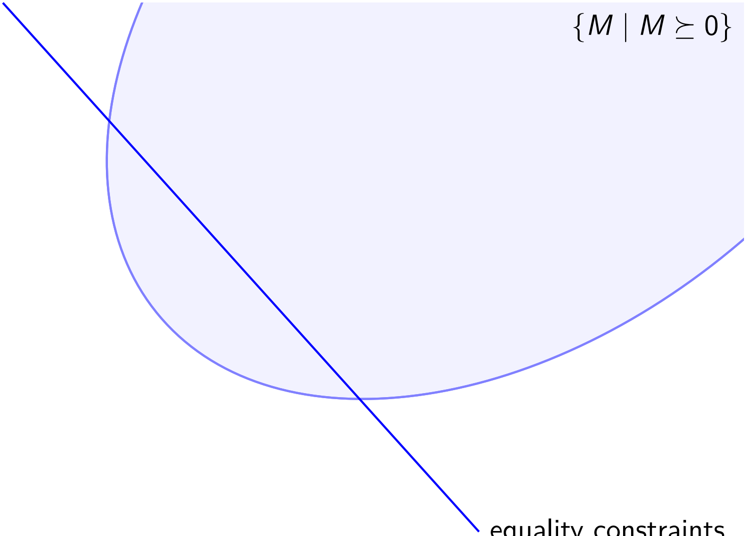
- Hence the validation method: given $p \simeq z^T Q z$
  1. Bound difference δ between coefficients of p and $z^T Q z$.
  2. If $Q - s \delta I \succeq 0$ (with $s :=$ size of Q), then p is proved SOS.
- 1 can be done with interval arithmetic and 2 with a Cholesky decomposition ($\Theta(s^3)$ flops).

$\Rightarrow$ Efficient validation method using just floats.

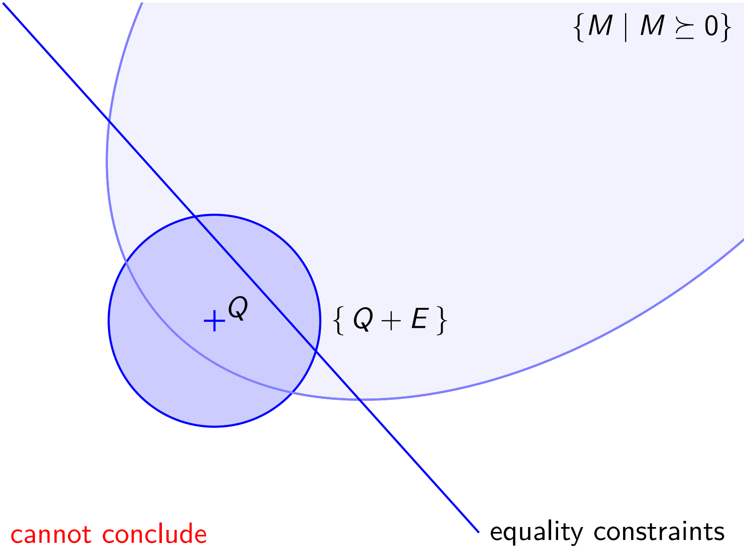
[SAS 2016, FMSD 2018]

Often won't work, needs some padding.

Intuitively

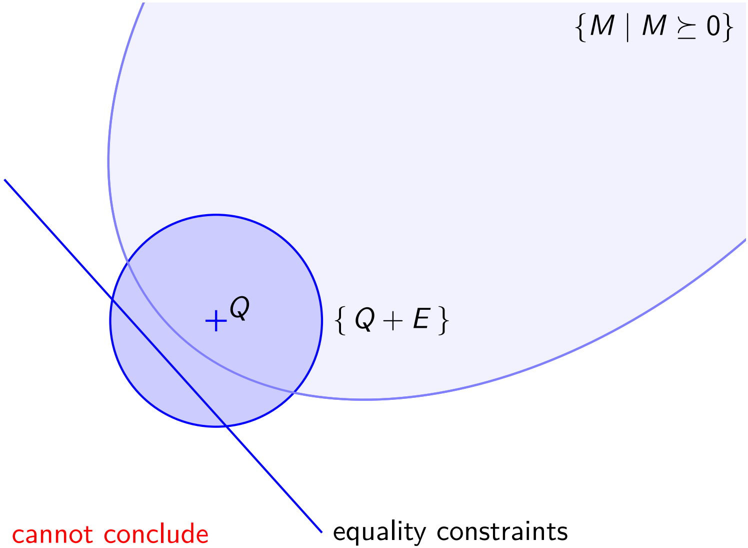


Intuitively

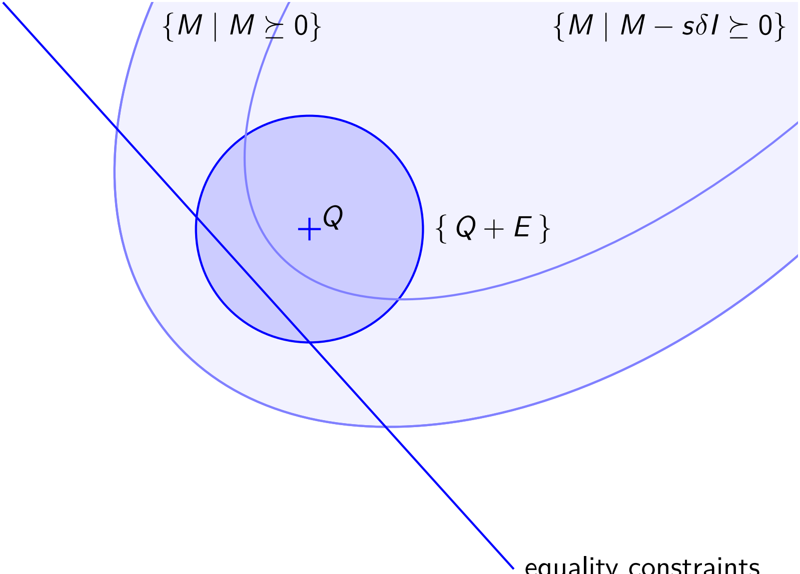


cannot conclude

Intuitively



Padding



$\{M \mid M \succeq 0\}$
$\{M \mid M - s\delta I \succeq 0\}$
Q
$\{\, Q + E \,\}$
equality constraints

Making it Work

- Instead of asking for $p = z^T Q z, Q \succeq 0$
  ask for $p = z^T Q z, Q - s\delta I \succeq 0$
- But isn't δ computed from the result Q? (distance p to $z^T Q z$)

Making it Work

- Instead of asking for $p = z^T Q z, Q \succeq 0$
  ask for $p = z^T Q z, Q - s\delta I \succeq 0$
- But isn't δ computed from the result Q? (distance p to $z^T Q z$)
- This distance is a stopping criterion of the interior-point algo.

$\Rightarrow$ overapproximation of δ

Making it Work

- Instead of asking for $p = z^T Q z, Q \succeq 0$
  ask for $p = z^T Q z, Q - s\delta I \succeq 0$
- But isn't δ computed from the result Q? (distance p to $z^T Q z$)
- This distance is a stopping criterion of the interior-point algo.

$\Rightarrow$ overapproximation of δ

Implemented in our OCaml library OSDP:

- simple interface to SOS programming,
- interfaces multiple SDP solvers (Csdp, Mosek, SDPA)
- under LGPL license
- available at `https://github.com/Embedded-SW-VnV/osdp`
  or `opam install osdp`

Using a Proof-Assistant

- We would like to prove our OCaml implementation

⇒ Use a proof-assistant

Using a Proof-Assistant

- We would like to prove our OCaml implementation

$\Rightarrow$ Use a proof-assistant

- Rocq (formerly Coq) features computation capabilities

Using a Proof-Assistant

- We would like to prove our OCaml implementation

⇒ Use a proof-assistant

- Rocq (formerly Coq) features computation capabilities

Our Rocq library ValidSDP (with Érik Martin-Dorel, UPS):

- automatic tactic for polynomial inequalities
- under LGPL license
- available at `https://github.com/validsdp/validsdp` or `opam install coq-validsdp`
- maintained since 2016, compatible with latest Rocq 9.1
- built on top of: OSDP, bignums, Flocq, CoqInterval, MathComp, multinomials, Analysis, CoqEAL

[CPP 2017]

ValidSDP, Example

```
From Ltac2 Require Import Ltac2.
From Stdlib Require Import Reals.
From ValidSDP Require Import validsdp.
Local Open Scope R_scope.

Let p x0 x1 x2 : R :=  (* A largish polynomial. *)
  2238448784199197/4503599627370496
  + -7081956584605647/72057594037927936 * x0
  + -5081574377800643/576460752303423488 * x2
  + 6018099001714223/18014398509481984 * x0^2
  + -30139342649847/1125899906842624 * x0 * x1
  + -541778131690975/9007199254740992 * x0^3
  + (* ... +78 lines *)

Lemma p_pos : forall x0 x1 x2 : R, p x0 x1 x2 >= 0.
Proof. intros x0 x1 x2; validsdp. (* 0.46 s *) Qed.
```

Proof by Reflection

Example

```
Inductive even : nat -> Prop :=
| Even0 : even 0
| EvenS : forall n, even n -> even (S (S n)).

Lemma even42 : even 42. Proof.
apply EvenS. apply EvenS. apply EvenS. (* ... x 21 *)
apply Even0. Qed.
```

Proof by Reflection

Example

```
Inductive even : nat -> Prop :=
| Even0 : even 0
| EvenS : forall n, even n -> even (S (S n)).

Lemma even42 : even 42. Proof.
apply EvenS. apply EvenS. apply EvenS. (* ... x 21 *)
apply Even0. Qed.

Fixpoint is_even n := match n with
| 0 => true | 1 => false | S (S n') => is_even n' end.

Lemma is_even_correct n : is_even n = true -> even n.

Lemma even42_refl : even 42. Proof.
apply is_even_correct. (* is_even 42 = true *)
compute. (* true = true *) exact eq_refl. Qed.
```

Proof by Reflection, with Witness

Example

```
Definition is_even_wit n w := Nat.eqb n (2 * w).

Lemma is_even_wit_correct w n :
  is_even_wit n w = true -> even n.

Lemma even42_refl_wit : even 42.
Proof. apply (is_even_wit_correct 21).
(* is_even_wit 42 21 = true *)
compute. (* true = true *) exact eq_refl. Qed.
```

Proof by Reflection, with Witness

Example

```
Definition is_even_wit n w := Nat.eqb n (2 * w).

Lemma is_even_wit_correct w n :
  is_even_wit n w = true -> even n.

Lemma even42_refl_wit : even 42.
Proof. apply (is_even_wit_correct 21).
(* is_even_wit 42 21 = true *)
compute. (* true = true *) exact eq_refl. Qed.
```

- for us, the witness is the SDP-computed matrix Q
- we use SDP solvers as untrusted oracles

Proof vs Computation

⇒ We need heavy computations (polynomials, matrices, floats)

- That we need to prove correct

Proof vs Computation

⇒ We need heavy computations (polynomials, matrices, floats)

- That we need to prove correct
- Multiple formalization of a same concept with trade-offs between, e.g., ease of proof and efficient computation
- Example

natural numbers	proof	computation
peano (`nat` in Rocq)	+	−−
binary (`N` in Rocq)	−	+
hardware (`bigN` in Rocq)	−−	++

⇒ Need a way to "switch" between formalizations

Parametricity

- Workflow

parametric program $\quad$ p(T, $+$, $*$)

Parametricity

- Workflow

parametric program $\quad p(T, +, *)$

$\downarrow$

$p(T_P, +_P, *_P)$

$\downarrow$ nice proof

correct $(p(T_P, +_P, *_P))$

Parametricity

- Workflow

parametric program $p(T, +, *)$

$p(T_P, +_P, *_P)$

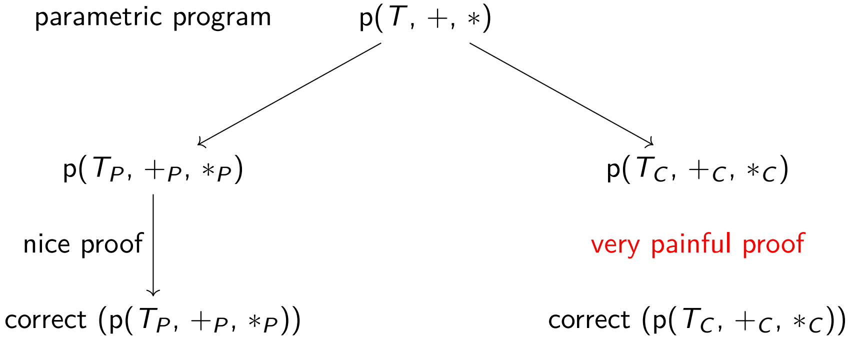
$p(T_C, +_C, *_C)$

nice proof

correct $(p(T_P, +_P, *_P))$

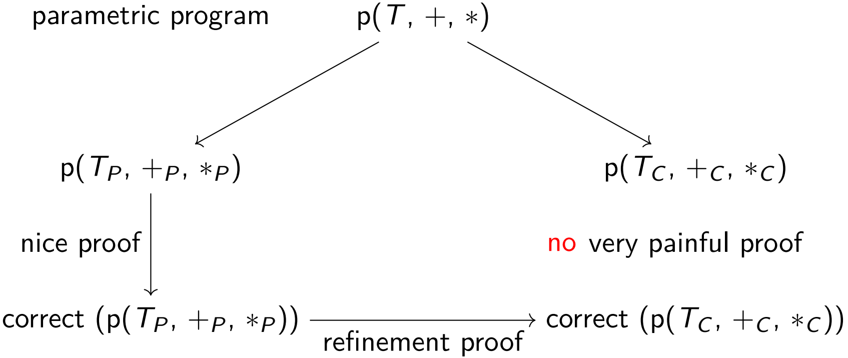
Parametricity

- Workflow



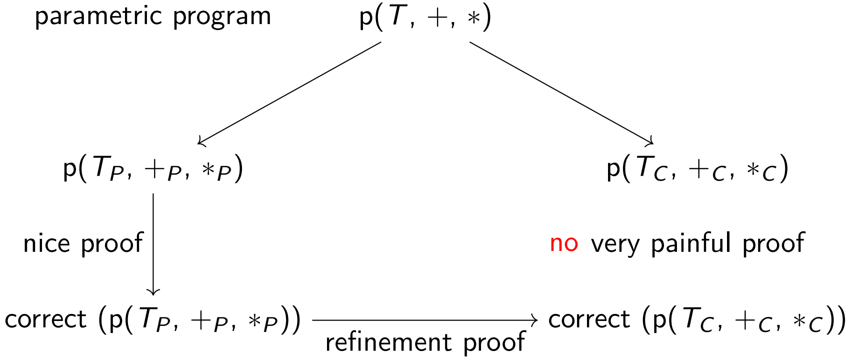
Parametricity

- Workflow



Parametricity

- Workflow



- CoqEAL largely automates refinement proofs
  based on paramcoq (now ported to Elpi `derive.param2`)

Instrumental in the making of ValidSDP [CPP 2017]

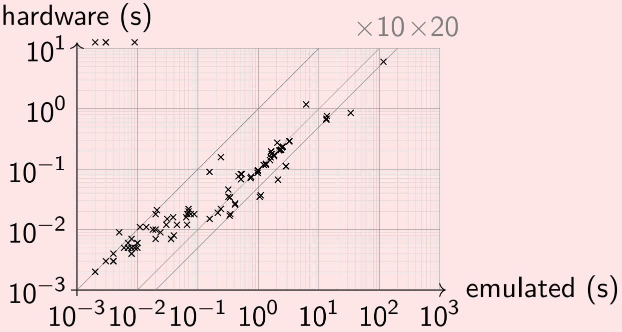
Hardware Floats in Rocq

- Our verification requires floating-point computations (Cholesky decomposition)
- Can be emulated with integers, but slow

Hardware Floats in Rocq

- Our verification requires floating-point computations (Cholesky decomposition)
- Can be emulated with integers, but slow

- add direct access to processor floats in Rocq
- CoqInterval benchmarks (with Guillaume Melquiond, INRIA):



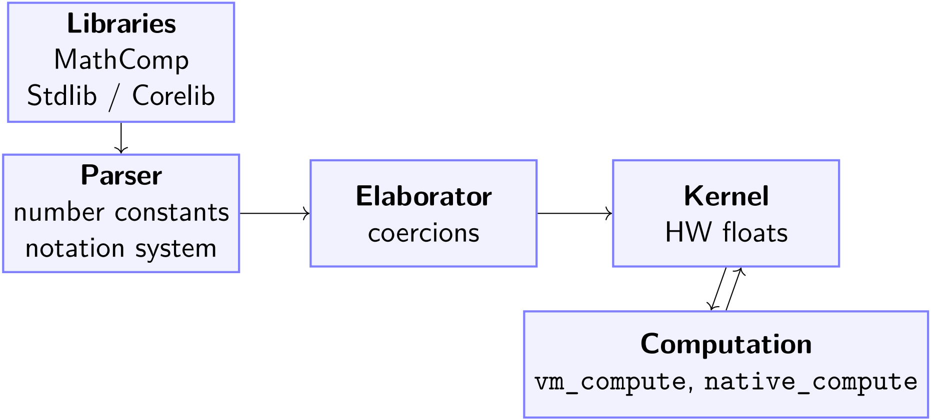
Hardware Floats in Rocq

- two order of magnitudes speedup on individual operators
- benched on Cholesky decomposition and CoqInterval
- floats spec. retrieved from Flocq library (bit precise)
- implem in each reduction engine (`compute`, `vm_compute`, `native_compute`)
- requires care to not break soundness (signed 0s, NaN payloads, OCaml unboxed float arrays,...)
- Started by Guillaume Bertholon during L3 internship, summer 2018 (coadvised with Érik Martin-Dorel, UPS)
- integrated in Coq 8.11 (released in Jan. 2020)

[ITP 2019, JAR 2023]

Contributing to Rocq

- Started in 2019 with hardware floats
- Core team member since 2023
- RM for 8.20 (June 2024, with Guillaume Melquiond, INRIA)
- 245 pull requests (PRs) authored / 200 PRs reviewed



Wrap up

- Polynomial invariants for controllers
- Nice inference techniques, but unsound

- Rigorous and cheap verification method
  OSDP (OCaml) [SAS 2016, FMSD 2018]
- Verified in Rocq (with Érik Martin-Dorel, UPS)
  ValidSDP [CPP 2017]
- Extensive use of parametricity techniques
- Hardware floats in Rocq [ITP 2019, JAR 2023]
- Core dev of Rocq

Wrap up

- Polynomial invariants for controllers
- Nice inference techniques, but unsound

- Rigorous and cheap verification method
  OSDP (OCaml) [SAS 2016, FMSD 2018]
- Verified in Rocq (with Érik Martin-Dorel, UPS)
  ValidSDP [CPP 2017]
- Extensive use of parametricity techniques
- Hardware floats in Rocq [ITP 2019, JAR 2023]
- Core dev of Rocq

Not seen today

- Double rounding, proofs in Rocq/Flocq [JFR 2014]
- Bounding floating-point rounding errors [JAR 2016]
- Integration into Alt-Ergo SMT solver (with Mohamed Iguernlala and Sylvain Conchon, INRIA) [TACAS 2018]

Verifying Polynomial Invariants

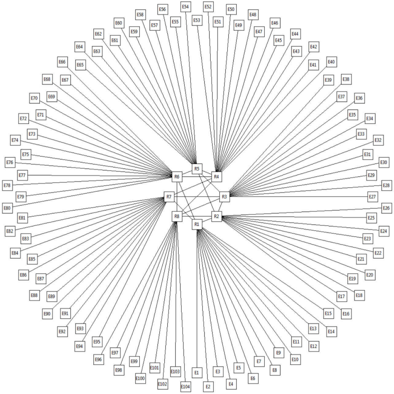
Verifying Real-Time Embedded Networks

Technical Expertise

Perspectives

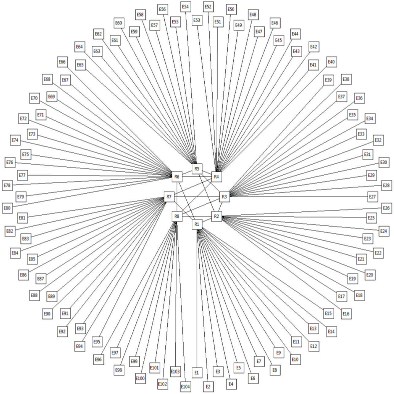
Embedded Networks

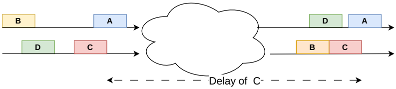
Many systems to connect in an aircraft $\Longrightarrow$ network



Embedded Networks

Many systems to connect in an aircraft $\Longrightarrow$ network





- Real-time constraints in avionics

$\Rightarrow$ Need bounds on network traversal times

- Also need to ensure absence of buffer overflows

Main Solutions for Real-Time Networks

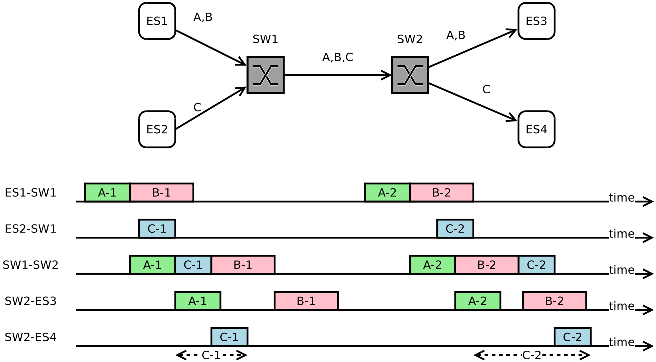
- Time triggered
  - Statically schedule all messages
  - Requires a global clock
  - Hard to establish and maintain
  - Applications in space industry (spacecrafts, launchers)

Main Solutions for Real-Time Networks

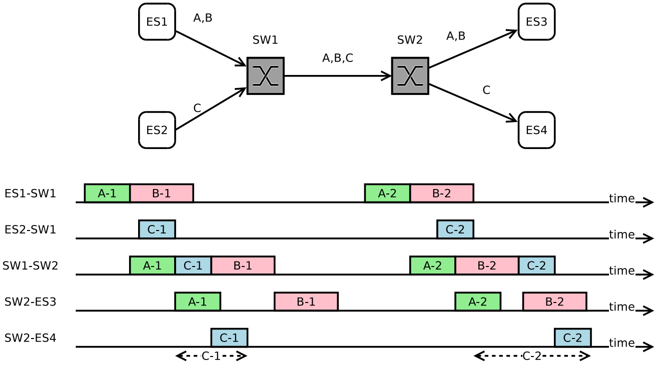
- Time triggered
  - Statically schedule all messages
  - Requires a global clock
  - Hard to establish and maintain
  - Applications in space industry (spacecrafts, launchers)

- Rate constrained
  - Limit emission rates of each node
  - And use a mathematical method to statically prove that everything arrives on time, without buffer overflow
  - No requirement for global clock
  - Used in all modern commercial aircraft

Delay computation challenge



Delay computation challenge

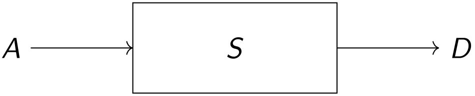


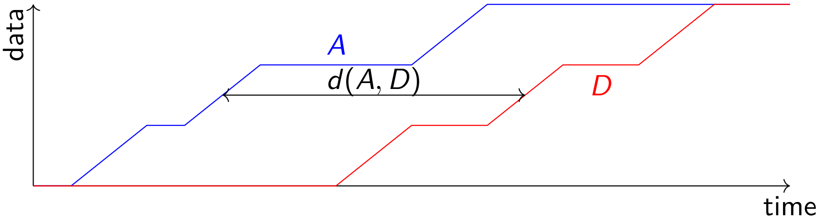
Computing the *exact* worst case

- an interleaving problem, with
- time, partially known arrival dates, complex scheduling, etc.
- NP-hard

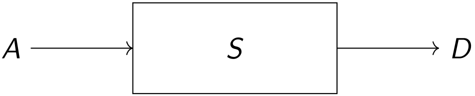
⇒ computation of upper bound

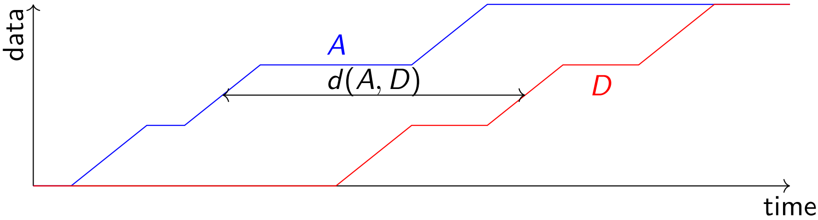
Network Calculus





Network Calculus

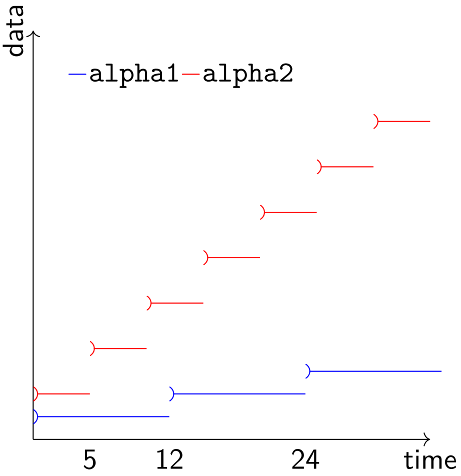




- Mathematical framework, based on
  - basic real analysis
  - tropical algebra (min-plus dioid of functions)
- Computations on piecewise-linear pseudo-periodic functions

Network Calculus, Computations

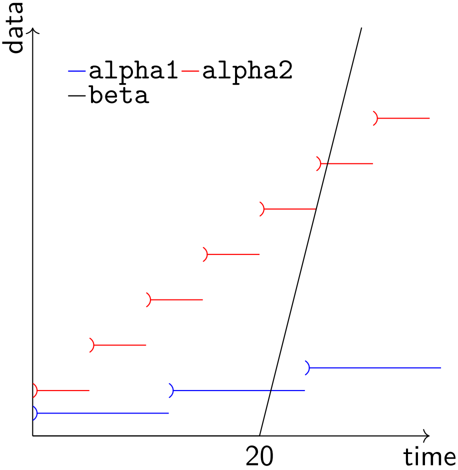
```
// input curves
alpha1 := stair(0, 12, 1)
alpha2 := stair(0, 5, 2)
```



On https://www.realtimeatwork.com/minplus-playground

Network Calculus, Computations

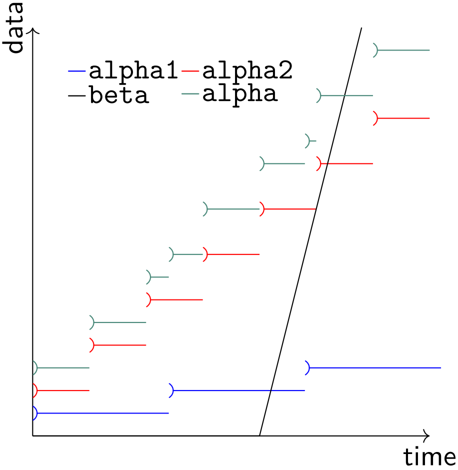
```
// input curves
alpha1 := stair(0, 12, 1)
alpha2 := stair(0, 5, 2)
beta := ratency(2, 20)
```



On https://www.realtimeatwork.com/minplus-playground

Network Calculus, Computations

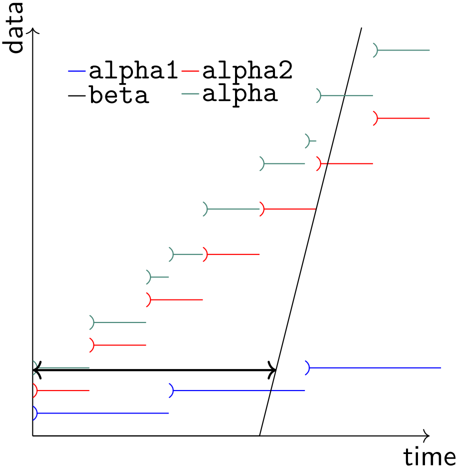
```
// input curves
alpha1 := stair(0, 12, 1)
alpha2 := stair(0, 5, 2)
beta := ratency(2, 20)
// computation
alpha := alpha1 + alpha2
```



On https://www.realtimeatwork.com/minplus-playground

Network Calculus, Computations

```
// input curves
alpha1 := stair(0, 12, 1)
alpha2 := stair(0, 5, 2)
beta := ratency(2, 20)
// computation
alpha := alpha1 + alpha2
d := hdev(alpha, beta)
» d = 43/2
```

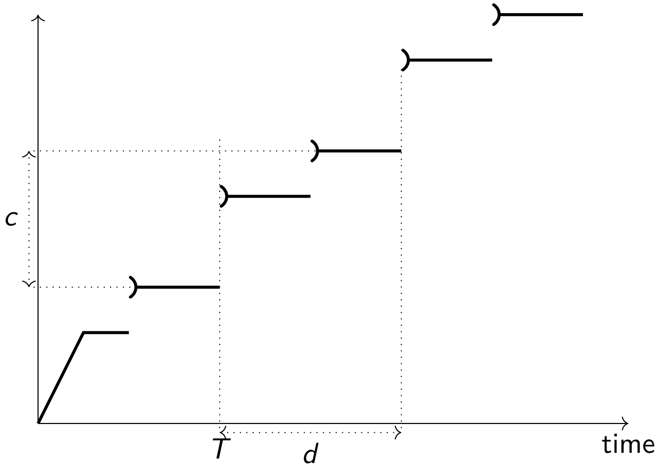


On https://www.realtimeatwork.com/minplus-playground

Encoding Functions

Network Calculus tools use functions that are:

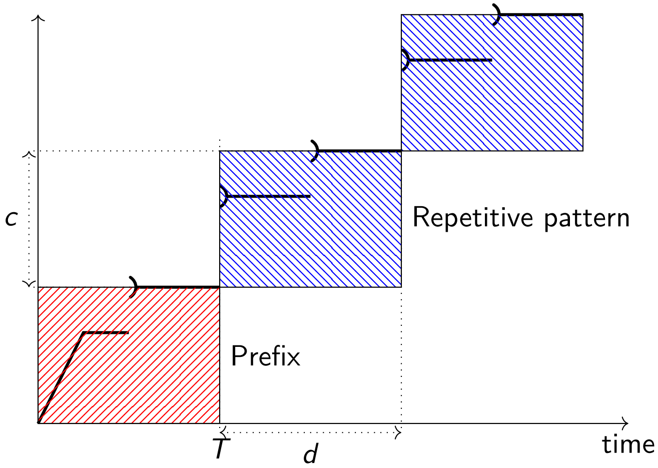
- piecewise affine: list of segments
- ultimately pseudo-periodic: a prefix and a repetitive pattern



Encoding Functions

Network Calculus tools use functions that are:

- piecewise affine: list of segments
- ultimately pseudo-periodic: a prefix and a repetitive pattern



Formal Proofs

Question: Are the theorems corrects?

- Maths are quite basic but can be tricky (discontinuities, ...)

⇒ Rocq formalization of the theory and representative theorems
`https://gitlab.mpi-sws.org/proux/nc-coq`
using: MathComp, MathComp Analysis, Hierarchy Builder
[Junior Workshop 2019]

Formal Proofs

Question: Are the theorems corrects?

- Maths are quite basic but can be tricky (discontinuities, ...)

⇒ Rocq formalization of the theory and representative theorems `https://gitlab.mpi-sws.org/proux/nc-coq` using: MathComp, MathComp Analysis, Hierarchy Builder [Junior Workshop 2019]

Question: Are the computations correct?

- Computation algorithms with many subcases

⇒ Rocq verified implementation `https://gitlab.mpi-sws.org/proux/nc-coq/-/tree/master/minerve` also using: bignums, CoqEAL (again) [NFM 2021]

Formal Proofs

Question: Are the theorems corrects?

- Maths are quite basic but can be tricky (discontinuities, ...)

⇒ Rocq formalization of the theory and representative theorems
`https://gitlab.mpi-sws.org/proux/nc-coq`
using: MathComp, MathComp Analysis, Hierarchy Builder
[Junior Workshop 2019]

Question: Are the computations correct?

- Computation algorithms with many subcases

⇒ Rocq verified implementation `https://gitlab.mpi-sws.org/proux/nc-coq/-/tree/master/minerve` also
using: bignums, CoqEAL (again) [NFM 2021]

- Tested on representative use cases. [ERTS 2022]

Lucien Rakotomalala's PhD (co-advised with Marc Boyer, ONERA)

Contributing to MathComp

- main library for mathematical developments in Rocq (inspired Mathlib in Lean)
- offers extensible algebraic structures (used for tropical algebra in our network calculus work)
- contributed to Hierarchy-Builder port
- RM for 2.0 (2023, with Reynald Affeldt, AIST, Japan)
- 200 PRs authored / 100 PRs reviewed

Contributing to MathComp

- main library for mathematical developments in Rocq (inspired Mathlib in Lean)
- offers extensible algebraic structures (used for tropical algebra in our network calculus work)
- contributed to Hierarchy-Builder port
- RM for 2.0 (2023, with Reynald Affeldt, AIST, Japan)
- 200 PRs authored / 100 PRs reviewed
- also contribute to Analysis
  - extended reals $\overline{\mathbb{R}} = \mathbb{R} \cup \{ \pm\infty \}$, used for network calculus
  - non negative numbers $\rightarrow$ basic interval arithmetic [ITP 2025]
  - a bit of probability [RTSS 2023]
  - 100 PRs authored / 100 PRs reviewed
- other contributions/maintenance: mathcomp-algebra-tactics, CoqEAL, paramcoq, bignums, coq-nix-toolbox

Wrap up

- Network Calculus: a framework to verify real-time networks

- Rocq proofs of main mathematical theorems
- Rocq automatic verification of computations [NFM 2021]
- Tested on representative use cases [ERTS 2022]
- Developer of MathComp

Wrap up

- Network Calculus: a framework to verify real-time networks

- Rocq proofs of main mathematical theorems
- Rocq automatic verification of computations [NFM 2021]
- Tested on representative use cases [ERTS 2022]
- Developer of MathComp

Not seen today

- Rocq proof of a novel Network Calculus result [ECRTS 2021]
- Link between Response Time Analysis (RTA) and Network Calculus (NC) [ECRTS 2022]
- 6 months visit at MPI-SWS (group of Björn Brandenburg) working on the Prosa library and Rocq proofs on probability [RTSS 2023]
- Study of the P4 language to specify switches [ERTS 2024]

Technical Expertise, Example

- Ariane 6 launcher uses TTEthernet technology for its embedded network
- Synchronized global clock

Technical Expertise, Example

- Ariane 6 launcher
  uses TTEthernet technology
  for its embedded network
- Synchronized global clock
- In 2017, bibliography review
  of model checking literature
  for CNES launcher division
- Discovered some missing cases
  with larger drift bounds
  in some failure scenarios

Image: ESA (CC-BY-SA)

Technical Expertise, Example

- Help Airbus to design and configure a real-time embedded network
- Verify real-time constraints are met
- Work started by Marc Boyer a few years ago
- Joined in 2024

Verifying Polynomial Invariants

Verifying Real-Time Embedded Networks

Technical Expertise

Perspectives

Perspectives

- Towards proving functional properties of *actual flight software* requires close collaboration with control theorists / engineers

Perspectives

- Towards proving functional properties of *actual flight software* requires close collaboration with control theorists / engineers
- Improving Rocq *elaborator*
  e.g., to write `n * x` rather than `n%:R * x`

Perspectives

- Towards proving functional properties of *actual flight software* requires close collaboration with control theorists / engineers
- Improving Rocq *elaborator*
  e.g., to write `n * x` rather than `n%:R * x`
- *Verified interval arithmetic*
  verify literature results on linear algebra
  bridging gap between computer algebra systems (CAS)
  and numerical frameworks (Octave, Matlab, Scilab)

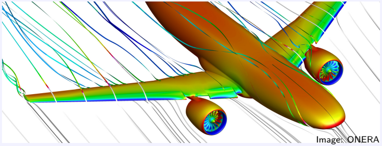
Perspectives

- Towards proving functional properties of *actual flight software* requires close collaboration with control theorists / engineers
- Improving Rocq *elaborator*
  e.g., to write `n * x` rather than `n%:R * x`
- *Verified interval arithmetic*
  verify literature results on linear algebra
  bridging gap between computer algebra systems (CAS)
  and numerical frameworks (Octave, Matlab, Scilab)
- *Verifying real-time network and systems*
  bridging gaps between models,
  mathematical theorems and computations

Design Process

physics, applied maths

model



control theory

controller

$$\dot{x} = Ax + Bu$$
$$y = Cx$$

computer science

program

```
double x[3] = {0, 0, 0}; double nx[3]; double in;
while (1) {
  in = acquire_input();
  nx[0] = 0.9379*x[0]-0.0381*x[1]-0.0414*x[2]+0.0237*in;
  ...  }
```

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 $x^T \left(R^T R\right) x = (Rx)^T (Rx) = \|Rx\|_2^2 \geqslant 0$).

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 $x^T \left(R^T R\right) x = (Rx)^T (Rx) = \|Rx\|_2^2 \geqslant 0$).
- 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
- If it succeeds (no $\sqrt{}$ of negative or div. by 0) then $A \succeq 0$.

Cholesky Decomposition (end)

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

Cholesky Decomposition (end)

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

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

Hence:

Theorem

If floating-point Cholesky succeeds on $A - c\, I$ then $A \succeq 0$

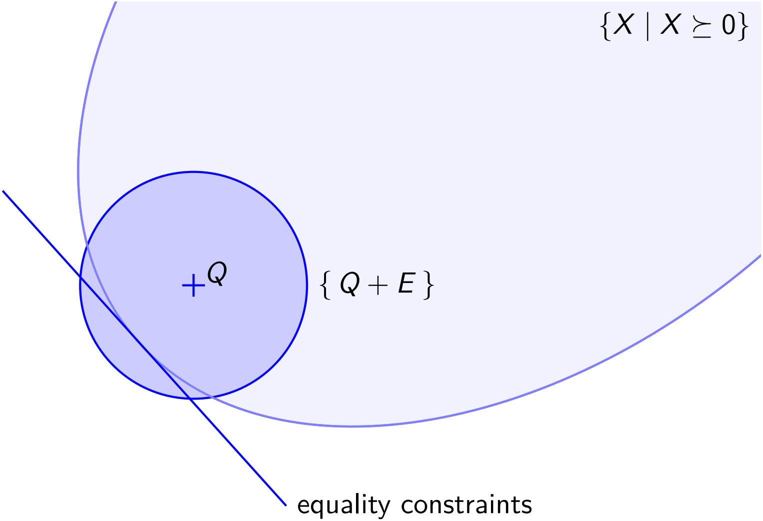
holds for any $c \geqslant \frac{(s+1)\varepsilon}{1-(s+1)\varepsilon} \mathrm{tr}(A) + 4s\left(2(s+1) + \max_i(A_{i,i})\right)\eta$

(ε and η relative and absolute precision of floating-point format).

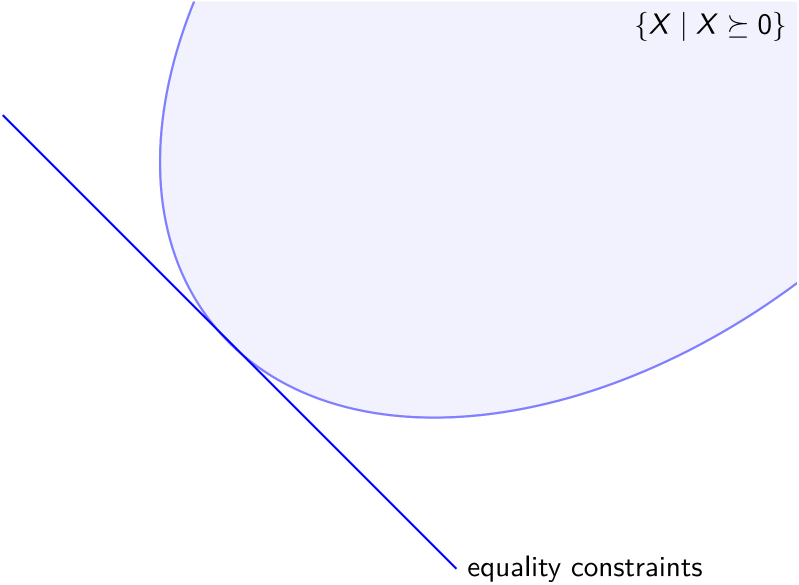
Proved in Rocq (paper proof: 6 pages, Rocq: 5.1 kloc)

Incompleteness: Empty Interior SDP Problems

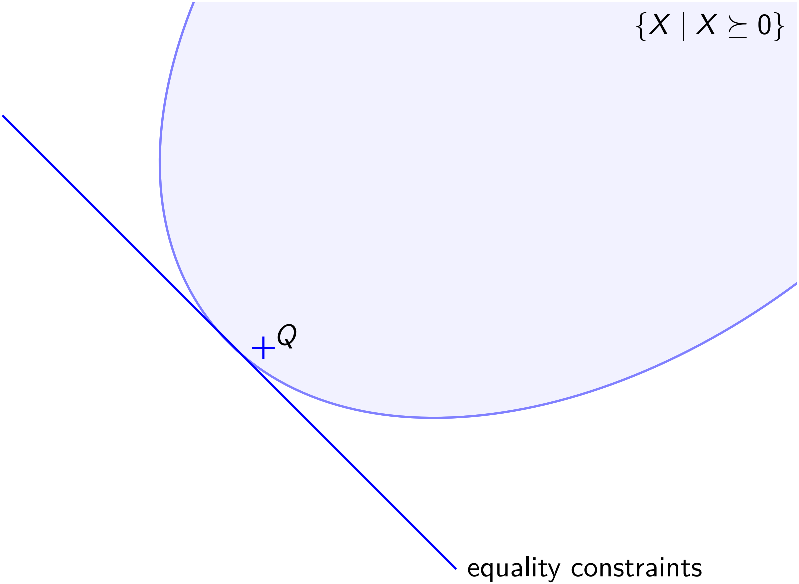
If the interior of the feasibility set of the problem is empty
(i.e., no feasible Q s.t. every Q' in a small neighborhood is feasible)
pure numerical methods won't work.



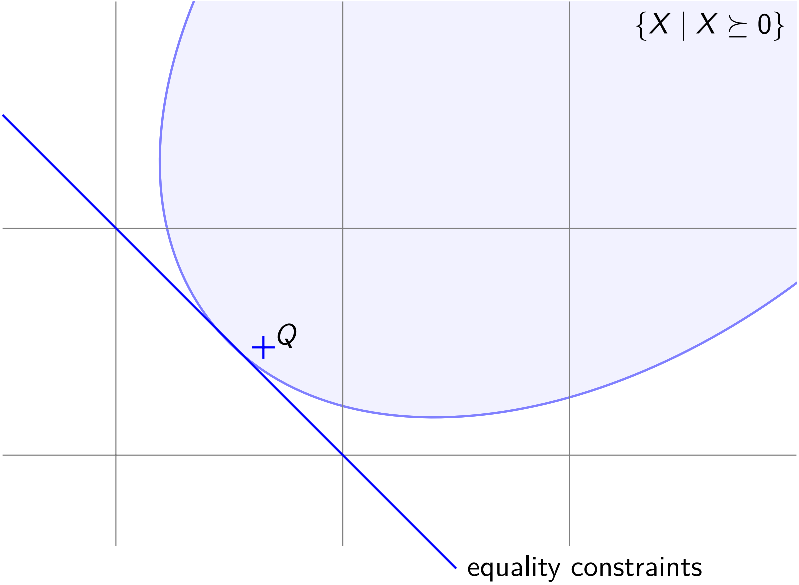
Intuitively, Rounding to an Exact Solution



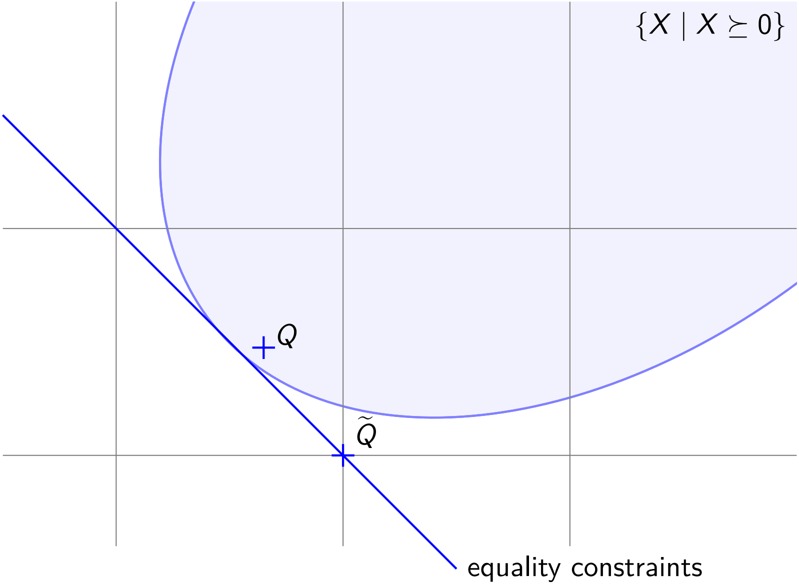
Intuitively, Rounding to an Exact Solution



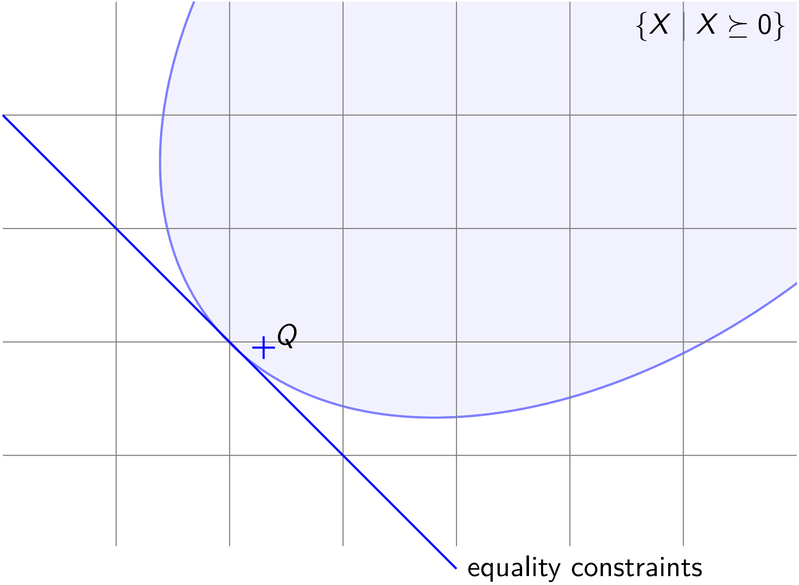
Intuitively, Rounding to an Exact Solution



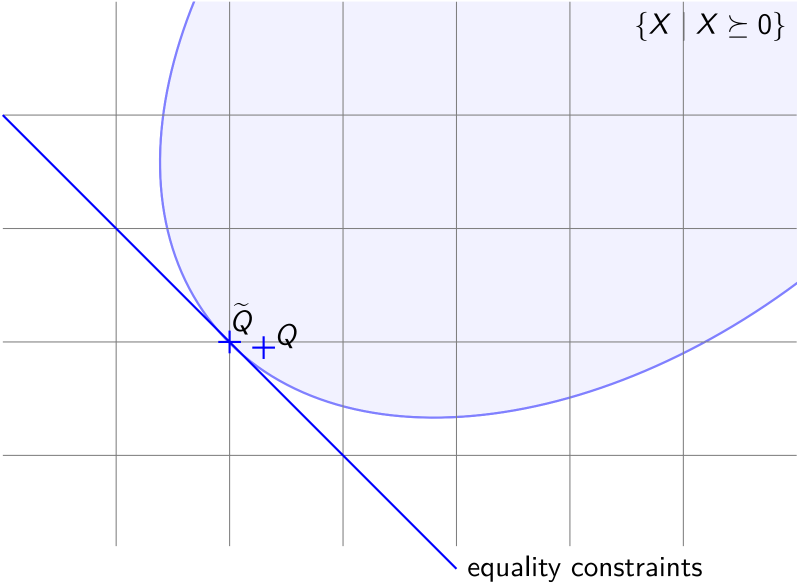
Intuitively, Rounding to an Exact Solution



Intuitively, Rounding to an Exact Solution

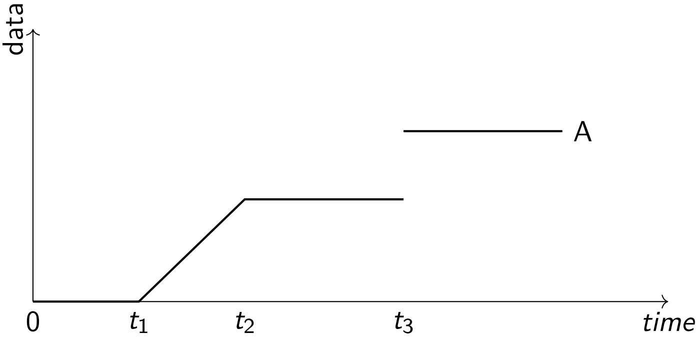


Intuitively, Rounding to an Exact Solution



Network Calculus, Arrival Model

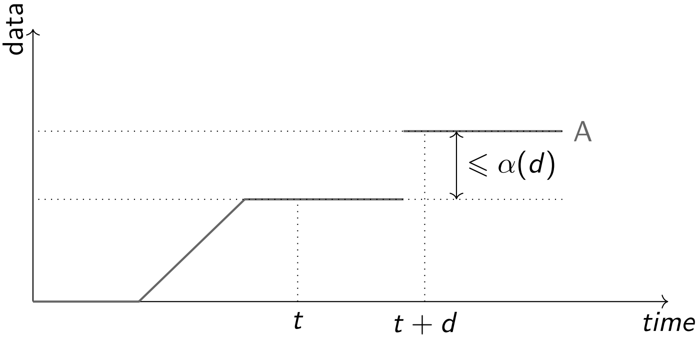
- Arrival A:
  - the cumulative amount of data received up to time t,
  - at some observation point in the network.



Network Calculus, Arrival Model

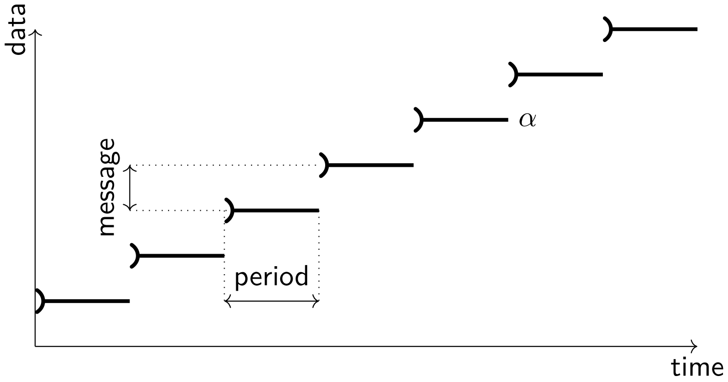
- Arrival A: some real behavior
  - the cumulative amount of data received up to time t,
  - at some observation point in the network.
- Arrival curve α: upper bound on all behaviors

$$\forall t, d \in \mathbb{R}^+, A(t+d) - A(t) \leqslant \alpha(d)$$



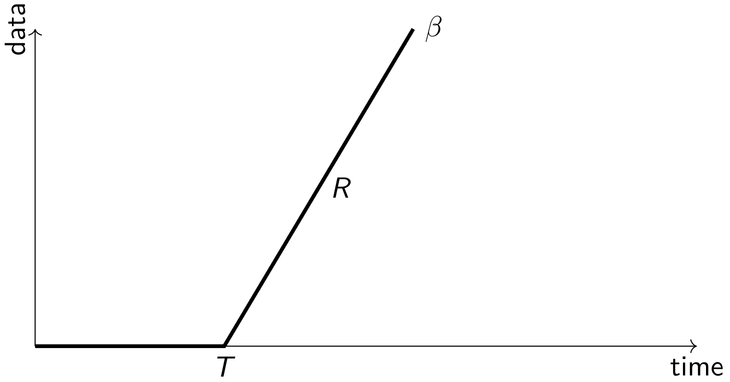
Network Calculus Arrival Curve

In the case of periodic messages with fixed size.



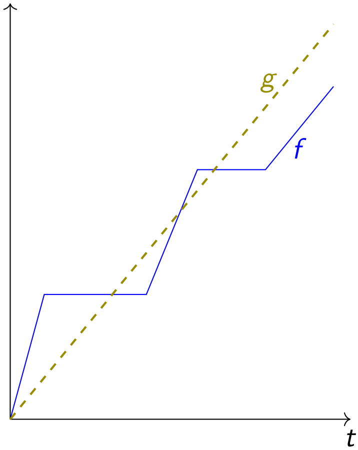
Network Calculus Service Model

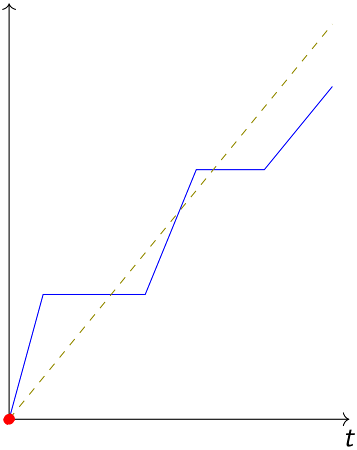
- Service curve β: a lower bound on the output
- Example: Rate-latency service
  - Server treats at least R bits per second,
  - after at most a latency T.



Convolution

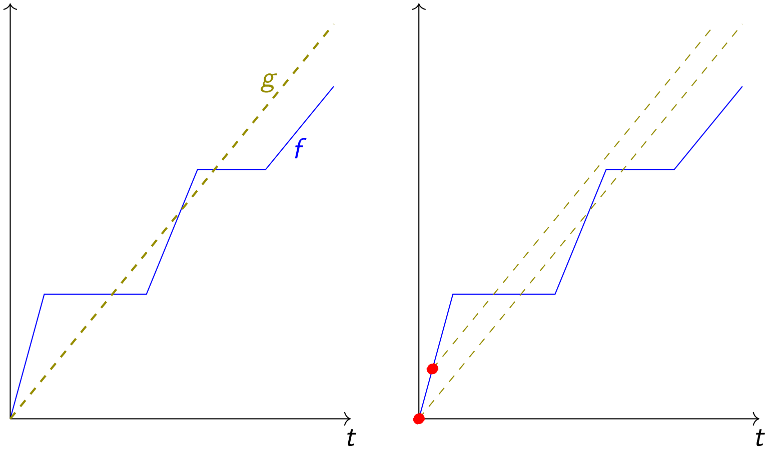
Defined as: $\forall t,\ (f * g)(t) = \inf_{0 \leqslant s \leqslant t} f(t-s) + g(s)$.





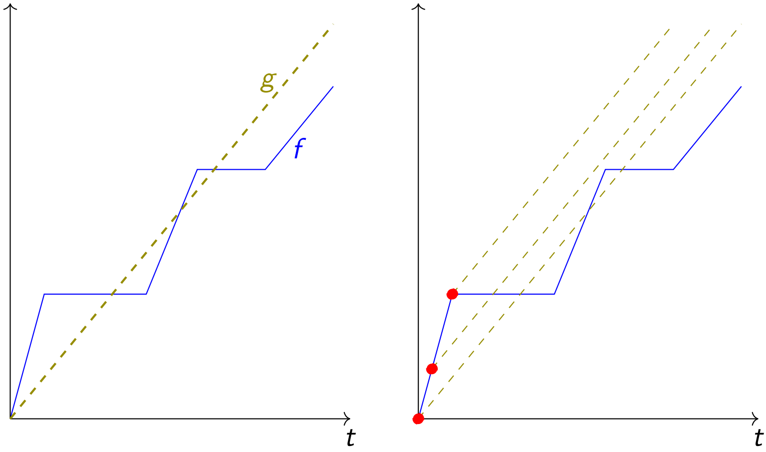
Convolution

Defined as: $\forall t,\ (f * g)(t) = \inf_{0 \leqslant s \leqslant t} f(t-s) + g(s)$.



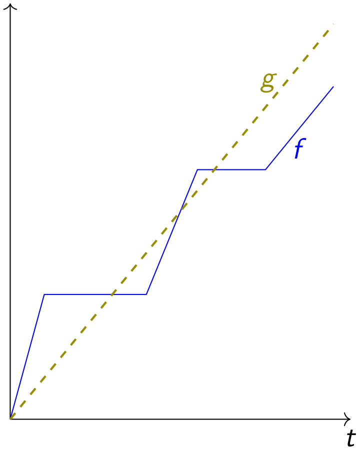
Convolution

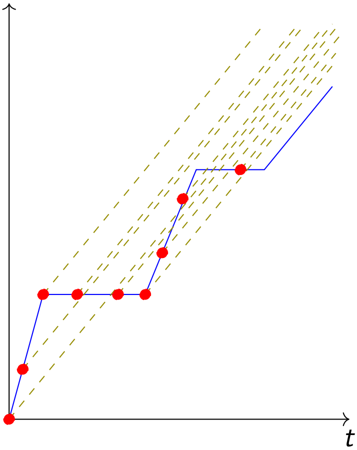
Defined as: $\forall t,\ (f * g)(t) = \inf_{0 \leqslant s \leqslant t} f(t-s) + g(s).$



Convolution

Defined as: $\forall t,\ (f * g)(t) = \inf_{0 \leqslant s \leqslant t} f(t-s) + g(s).$





Convolution

Defined as: $\forall t,\ (f * g)(t) = \inf_{0 \leqslant s \leqslant t} f(t-s) + g(s).$