Verified reductions for optimization

Alexander Bentkamp[1,2], **Ramon Fernández Mir**[3], Jeremy Avigad[4]

[1]Heinrich-Heine-Universität Düsseldorf, [2]Chinese Academy of Sciences,
[3]University of Edinburgh, [4]Carnegie Mellon University

April 26, 2023

Motivation

Convex optimization problems are ubiquitous in engineering, industry and finance. Some applications include

- safety and stability analysis of control systems
- power control
- portfolio optimization
- electronic circuit design
- etc.

There is often a large gap between the original problem and the problem sent to the solver.

Proof assistants can help with bridging this gap reliably.

Convex optimization

Optimizing over $x \in \mathbb{R}^n$ with convex f_is and affine h_is, a convex optimization problem in *standard form* is

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, k \\ & h_i(x) = 0, \quad i = 1, \ldots, l. \end{array}$$

However, solvers want the problem in *conic form*:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in K, \end{array}$$

where K is a convex cone e.g. $\mathbb{R}^n_+$, $\mathcal{Q}^n$, $\mathcal{K}_{\text{exp}}$, etc.

Disciplined convex programming

A framework to systematically reduce problems to conic form.

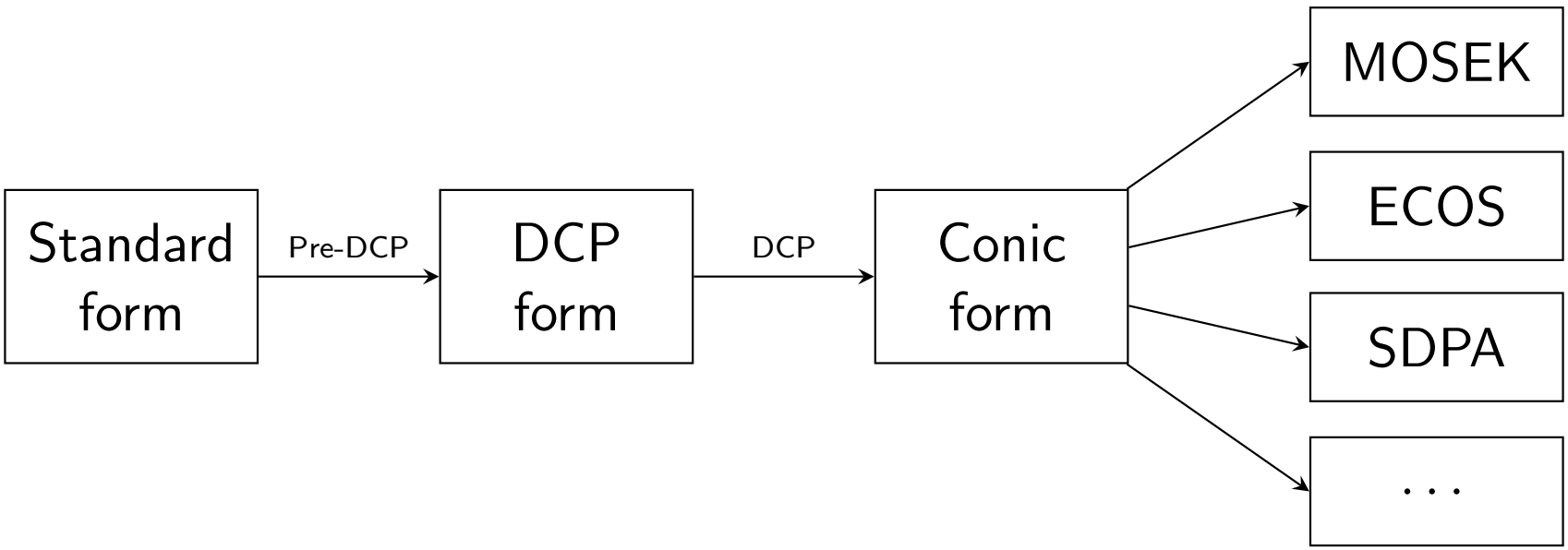
Key idea: replace expressions that are not in conic form with equivalent optimization problems in conic form.

The framework consists of

- a library of well-understood base functions called *atoms*
- optimization problems representing each atom called *graph implementations* e.g. $|x| \mapsto \min\{t \mid x \leq t \wedge -x \leq t\}$
- infrastructure to keep track of curvature, monotonicity, etc.

Disciplined convex programming

The current setup:



- Pre-DCP transformations are done by hand.
- DCP transformations are done using a modeling framework such as CVXPY, CVX, Convex.jl, etc.

Problem description

Issues:

- Transformations are not rigorous.
  - Pre-DCP transformations can introduce errors.
  - DCP transformations are hard-coded and syntactic, and side conditions are not checked in a principled way.
  - DCP frameworks rely on properties of the atoms that are assumed but not verified.
- Solvers cannot be fully trusted: interior-point methods are inherently approximate, some problems are ill-conditioned, etc.

Problem description

Issues:

- Transformations are not rigorous.
  - Pre-DCP transformations can introduce errors.
  - DCP transformations are hard-coded and syntactic, and side conditions are not checked in a principled way.
  - DCP frameworks rely on properties of the atoms that are assumed but not verified.
- Solvers cannot be fully trusted: interior-point methods are inherently approximate, some problems are ill-conditioned, etc.

Solution: `CvxLean`, a modeling framework written in Lean.

The Lean theorem prover

Lean 4 is a programming language and interactive theorem prover. Some features relevant to this project:

- access to a formal mathematics library (mathlib) containing $\sim$45k definitions and $\sim$111k theorems
- the interactive view where users see how the problem changes at every step
- extensible syntax to express problems in a natural way and create custom commands
- metaprogramming support to conveniently manipulate expressions and build tactics

CvxLean overview

The type of minimization problems

```
structure Minimization (D R : Type) :=
  (objFun : D → R)
  (constraints : D → Prop)
```

The type of solutions

```
structure Solution {D R : Type} [Preorder R] (p : Minimization D R) :=
  (point : D)
  (feasibility : p.constraints point)
  (optimality : ∀ y : FeasPoint p, p.objFun point ≤ p.objFun y.point)
```

CvxLean overview

A problem in CvxLean is defined as follows:

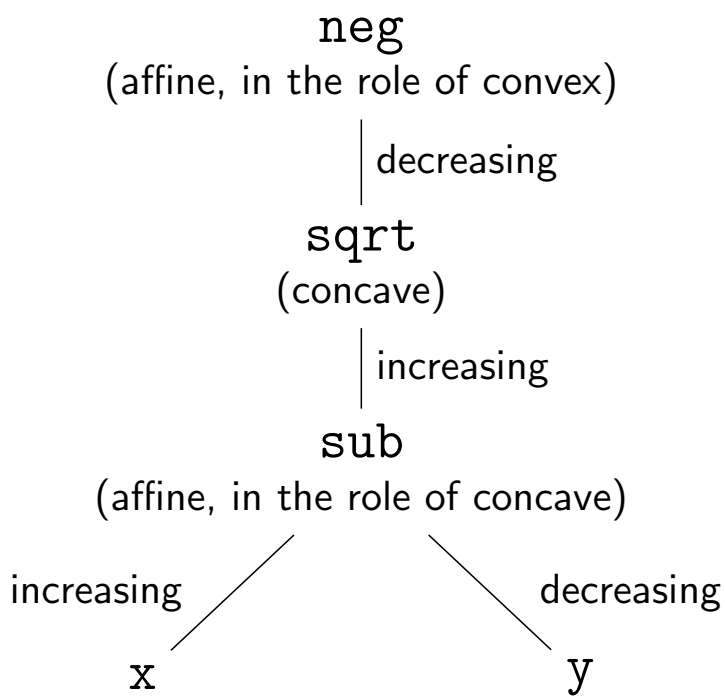
```
def prob :=
  optimization (x y : ℝ)
      minimize -sqrt (x - y)
      subject to
        c1 : y = 2*x - 3
        c2 : x^2 ≤ 2
        c3 : 0 ≤ x - y

#check prob -- Minimization (Real × Real) Real
```

Reduction to conic form

First, traverse the objective function and the constraints and build expression trees from our library of atoms satisfying

- curvature constraints e.g. the objective function is convex
- variable conditions e.g. the argument of $\sqrt{\cdot}$ is non-negative



Reduction to conic form

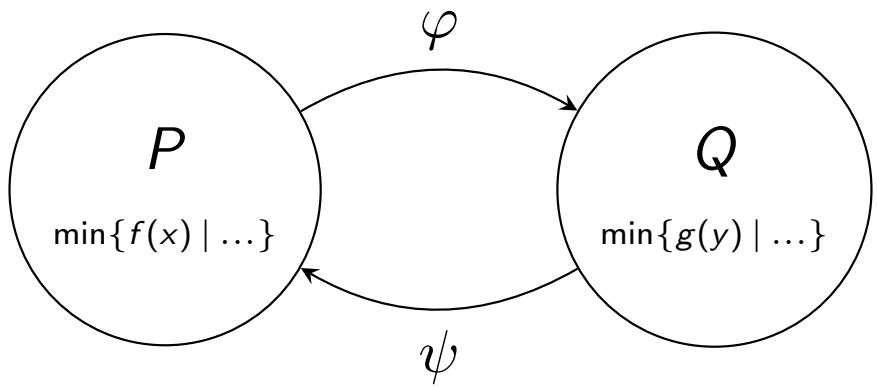
Next, iteratively replace atom applications by their graph implementations.

For example, we can replace a subexpression $\sqrt{x - y}$ by a new variable t and add the constraint $t^2 \leq x - y$.

```
optimization (x y t.0 t.1 : ℝ)
  minimize -t.0
  subject to
    c1' : zeroCone (2*x - 3 - y)              -- 2*x - 3 - y = 0
    c2' : posOrthCone (2 - t.1)               -- 2 - t.1 ≥ 0
    c4' : rotatedSoCone (x - y) 0.5 ![t.0]    -- x - y ≥ t.0^2
    c5' : rotatedSoCone t.1 0.5 ![x]          -- t.1 ≥ x^2
```

Verifying the reduction

Problems P and Q are *strongly equivalent* if we have



- x feasible in $P \Rightarrow \varphi(x)$ feasible in Q and $g(\varphi(x)) \leq f(x)$
- y feasible in $Q \Rightarrow \psi(y)$ feasible in P and $f(\psi(y)) \leq g(y)$

This guarantees that φ and ψ map the solutions. It is enough to capture all the transformations used in CVXPY, for instance.

Verifying the reduction

`CvxLean` builds (φ, ψ) so problems before and after each reduction step are strongly equivalent.

The idea is that every eliminable atom includes proofs of key properties that can be combined to prove the overall equivalence.

Following the example of `sqrt`, we need to prove

- solution correctness: `∀ a, 0 ≤ a → sqrt a = sqrt a`
- solution feasibility: `∀ a, 0 ≤ a → (sqrt a)^2 ≤ a`
- optimality: `∀ v a a', a ≤ a' → v^2 ≤ a → v ≤ sqrt a'`
- condition elimination: `∀ v a a', a ≤ a' → v^2 ≤ a → 0 ≤ a'`

Adding atoms

The declaration of the atom `sqrt` looks as follows:

```
declare_atom sqrt [concave] (x : ℝ)+ : sqrt x :=
  vconditions (cond : 0 ≤ x)
  implementationVars (t : ℝ)
  implementationObjective (t)
  implementationConstraints (c1 : rotatedSoCone x 0.5 ![t])
  solution (t := Real.sqrt x)
  solutionEqualsAtom by ...
  feasibility (c1 : by ...)
  optimality by ...
  vconditionElimination (cond : by ...)
```

The ellipses are filled by the formal proofs of the four conditions.

More complex atoms

Provided that $A \succeq 0$, the graph implementation of the $log(det(A))$ is the following optimization problem over $t \in \mathbb{R}^n$ and $Y \in \mathbb{R}^{n \times n}$:

$$\begin{aligned} \text{maximize} \quad & \sum_i t_i \\ \text{subject to} \quad & (t, 1, y) \in \mathcal{K}^n_{\text{exp}} \\ & \begin{pmatrix} D & Z \\ Z^T & A \end{pmatrix} \succeq 0, \end{aligned}$$

where $y_i = Y_{ii}$, $D = \text{Diag}(Y)$ and $Z = \text{UpperTriangular}(Y)$.

The proofs of this atom required some extra work such as results about LDL decompositions and Schur complements.

This was crucial to verify the reduction for covariance estimation for Gaussian variables.

The `solve` command

```
solve prob

#print prob.reduced    -- shows the reduced problem

#eval prob.status      -- "PRIMAL_AND_DUAL_FEASIBLE"
#eval prob.value       -- 2.101003
#eval prob.solution    -- (-1.414214, -5.828427)
```

The `solve` command reduces the problem, sends it to MOSEK, and reconstructs the point in the original domain.

User-defined reductions

CvxLean

```
def prob2 :
  optimization (x y : ℝ)
    maximize x + y
    subject to
      h : (exp x) * (exp y) ≤ 10

solve prob2
```

CVXPY

```
x = cp.Variable(1)
y = cp.Variable(1)

objFun = cp.Maximize(x + y)
constr = [cp.exp(x) * cp.exp(y) <= 10]

prob2 = cp.Problem(objFun, constr)
prob2.solve()
```

Both fail because the constraint is not DCP...

But, clearly, we just need to note that $e^x e^y = e^{x+y}$.

User-defined reductions

CvxLean

```
reduction red/prob2 :
  optimization (x y : ℝ)
    maximize x + y
    subject to
      h : (exp x) * (exp y) ≤ 10 := by
    conv_constr =>
      rw [←Real.exp_add]

solve prob2
-- (2.302585, 0.000000)
```

CVXPY

```
x = cp.Variable(1)
y = cp.Variable(1)

objFun = cp.Maximize(x + y)
-- next line edited by hand
constr = [cp.exp(x + y) <= 10]

prob2 = cp.Problem(objFun, constr)
prob2.solve()
-- [2.30258509] [0.]
```

With the `reduction` command, you can transform the problem using lemmas and tactics from mathlib. In this case:

```
theorem exp_add : exp (x + y) = exp x * exp y := by ...
```

Conclusion and future steps

1. Encode more and more practical examples.
2. Go beyond DCP, for instance, be able to support geometric or quasiconvex programming.
3. Work on automation to discharge side conditions.
4. Generate proofs of rigorous bounds.

The code is publicly available at
`https://github.com/verified-optimization/CvxLean`.