Verified Optimization
(work in progress)

Alexander Bentkamp
Vrije Universiteit Amsterdam
Department of Computer Science
De Boelelaan 1111,
1081 HV Amsterdam, Netherlands
bentkamp@gmail.com

Jeremy Avigad
Carnegie Mellon University
Department of Philosophy
Baker Hall 161,
Pittsburgh, PA 15213
avigad@cmu.edu

Abstract

Optimization is used extensively in engineering, industry, and finance, and various methods are used to transform problems to the point where they are amenable to solution by numerical methods. We describe progress towards developing a framework, based on the Lean interactive proof assistant, for designing and applying such reductions in reliable and flexible ways.

1 Introduction

Interactive proof assistants are used to verify complex mathematical claims with respect to the primitives and rules of a formal axiomatic foundation. Formalization yields a high degree of certainty in the correctness of such claims, but it places a very high burden on practitioners, and for many purposes it is a higher standard than users may want or need. The project we describe here is motivated by the observation that interactive theorem provers can offer a wider range of benefits to applied mathematicians. Sometimes even just a formal specification of a complex problem or model is helpful, since it provides clarity and precision that can serve as a touchstone for informal reasoning and algorithmic implementation. Formal representation in a theorem prover can also serve as a gateway to the use of external tools like computer algebra systems, numeric computation packages, and automated reasoning systems, providing a basis for coordinating and interpreting the results. And verification itself is not an all-or-nothing affair; every working mathematician and scientist has to balance pragmatic constraints against the goal of ensuring that their results are as reliable as possible, and they should have the flexibility of deciding where verification effort matters the most. Proof assistants need to become a help rather than a hindrance before they are ready to enter the mainstream.

Optimization problems and constraint satisfaction problems are now ubiquitous in engineering, industry, and finance. These address the problem of finding an element of $\mathbb{R}^n$ satisfying a finite set of constraints or determining that the constraints are unsatisfiable; the problem of bounding the value of an objective function over the domain defined by such a set of constraints; and the problem of finding a value of the domain that maximizes (or minimizes) the value of the objective function. Linear programming, revolutionized by Dantzig's introduction of the simplex algorithm in 1947, deals with the case in which the constraints and objective function are linear. The development of interior point methods in the 1980s allows for the efficient solution of problems defined by convex constraints and objective functions, which gives rise to the field of convex programming [BV14].

There are a number of ways in which formal verification can be used to improve the reliability of such methods. Checking the correctness of a solution to a satisfaction problem is easy in principle: one simply plugs the result

into the constraints and checks that they hold. Verifying the correctness of a bounding problem or optimization problem is often almost as easy, in principle, since the results are often underwritten by the existence of suitable *certificates* that are output by the optimization tools. In practice, these tasks are made more difficult by the fact that floating point calculation can introduce numerical errors that bear on the correctness of the solution.

Here we focus on a different part of the process, namely, that of manipulating a problem and reducing it to a form where optimization software can be applied. Mathematical models are often complex, and practitioners rely on heuristics and expertise to put problems into forms that admit computational solutions. Such transformations are hard to automate, and manual transformation is error-prone. Our goal is to show proof assistants can be put to good use towards finding and verifying these transformations, and to develop tools to support the process.

In Section 2, we describe a general formal framework for reasoning about problems and reductions between them. In Section 3, to illustrate the method, we describe a class of transformations that form the basis for *disciplined convex programming*, a component of the popular CVX package [GB14]. Even though these transformations are relatively straightforward, we argue that a proof assistant provides a natural setting to carry them out in a verified way. In Section 4, we discuss more substantial problem transformations and reductions.

Our current work is spread between two versions of the Lean system [dMKA+15]. The development of Lean 3 has mostly stabilized; its library, mathlib [Mat20], comprises around $600\,000$ lines of code and covers substantial portions of algebra, linear algebra, topology, measure theory, and analysis. Lean 4 is currently under development as a performant dependently typed programming language; it is not backward compatible with Lean 3, and does not have a substantial library yet. We intend to make use of Lean 4's support for user extensible syntax, as described below, and we plan to move the full development to Lean 4 as soon as its library will support it.

2   Optimization Problems and Reductions

The general structure of an optimization problem is as follows:

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

We express maximization problems by negating the objective function. We assume that the objective function objFun is defined over the data type Domain and takes values in the real numbers. (It is often useful to allow values in the extended real numbers, and we have not ruled out adopting this option instead.) The domain is often $\mathbb{R}^n$ or a space of matrices, but it can also be something more exotic, like a space of functions. A *feasible point* is an element of the domain satisfying the constraints. A *solution* to the minimization problem is a feasible point x such that for every feasible point y the value of the objective function at x is smaller than or equal to the value at y.

Feasibility and bounding problems can also be expressed in these terms. If the objective function is constant (for example, the constant zero function), a solution to the optimization problem is simply a feasible point. And given a domain, an objective function, and constraints, a value b is a (strict) bound on the value of the objective function over the domain if and only if the feasibility problem obtained by adding the inequality objFun x $\le$ b to the constraints has no solution.

If p and q are problems, a *reduction* from p to q is simply a function mapping any solution to q to a solution to p. The existence of such a reduction means that to solve p it suffices to solve q. If p is a feasibility problem, it means that the feasibility of q implies the feasibility of p, and, conversely, that the infeasibility of p implies the infeasibility of q. With this framework in place, we can now easily describe what we are after: we are looking for a system that helps a user reduce a problem p to a problem q that can be solved by an external solver. (For a bounding problem q, the goal is instead to find a reduction to q from an infeasible problem p.) At the same time, we wish to verify the correctness of the reduction, either automatically or with user interaction. This will ensure that the results from the external solver really address the problem that the user is interested in solving.

There are at least three advantages to performing such transformations in a proof assistant. First, it offers strong guarantees that the results are correct and have the intended meaning. Second, it means that users can perform the transformations interactively or partially, and thus introspect and explore the results of individual transformation steps. Finally, users can benefit from the ambient mathematical library, including a database of functions and their properties.

The formulation described above is good for reasoning about problems in general, but it is not as good for reasoning about *particular* problems. The optimization function in our representation of a minimization problem

is a unary function and the constraints are given by a unary predicate, but we commonly think of these in terms of multiple variables, along these lines:

```
minimization
  !vars x y z
  !objective x + y + z
  !constraints
    x + 3 * y > 5,
    z < 10
```

We have implemented exactly this syntax using Lean 4's flexible mechanisms for macro expansion [UdM20], so that it represents an expression of the `Minimization` type presented above. We use exclamation marks because the keywords we choose become parser tokens in any Lean file that imports our library; for example, with the syntax above, the tokens `minimization`, `!vars`, `!objectives`, and `!constraints` can no longer be used as variable names or identifiers. Conversely, Lean's *delaborator* makes it possible to pretty-print suitably-described problems in the form above. We intend to use tactics written in Lean's powerful metaprogramming language and a supporting library to facilitate the interactive construction of reductions, using the means above to mediate between internal representations and user-facing syntax.

3 Disciplined Convex Programming

Disciplined convex programming (DCP) [Gra04, GBY06] is a framework to specify convex optimization problems. Any optimization problem following the rules of the framework can be solved fully automatically. An example of a DCP problem is the following [Gra04, equation (3.60)]:

$$\text{minimize} \quad cx \quad \text{subject to} \quad \exp(y) \leq \log(a\sqrt{x} + b), \quad ax + by = d \tag{$\star$}$$

where a, b, c, d are parameters with $a \geq 0$, and x, y are variables. For this problem to be DCP conformant, it is crucial that for instance the argument $a\sqrt{x} + b$ of the concave, nondecreasing function log is itself concave; that the convex expression $\exp(y)$ is on the left of $\leq$; and that the concave expression $\log(a\sqrt{x} + b)$ is on the right of $\leq$. The DCP rules allow for a systematic verification of all necessary conditions.

The DCP framework is implemented in the modeling systems CVX [GB14, GB08], CVXPy [AVDB18, DB16], Convex.jl [UMZ$^+$14], and CVXR [FNB20]; a related reduction system is implemented in YALMIP [Löf04]. These systems transform a DCP problem into conic form, a more constrained canonical form that subsumes linear, quadratic, and semidefinite programs. The conic problem is then solved by external solvers such as SeDuMi [Stu99] and SDPT3 [TTT99], and the result is translated back into a solution to the original DCP problem.

Conic problems have the following form, where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $G \in \mathbb{R}^{k \times n}$, $h \in \mathbb{R}^k$ are parameters and $x \in \mathbb{R}^n$ are variables.

$$\text{minimize} \quad c^t x \quad \text{subject to} \quad Ax = b, \quad Gx - h \in K$$

The parameter K is a convex cone. The conic solvers require K to be a cartesian product of cones supported by the solver—e.g., the nonnegative orthant $\{x \in \mathbb{R}^l \mid x_i \geq 0 \text{ for all } i\}$, the second-order cone $\{x \in \mathbb{R}^l \mid x_1 \geq \sqrt{x_2^2 + \cdots x_l^2}\}$, or the exponential cone $\{x \in \mathbb{R}^3 \mid x_1 \geq x_2 e^{x_3/x_2}, x_2 > 0\} \cup \{x \in \mathbb{R}^3 \mid x_1 \geq 0, x_2 = 0, x_3 \leq 0\}$.

Support for DCP in Lean will form the basis of our project. DCP is widely applicable, and the transformations are relatively simple. It is therefore a suitable testbed for the basic definitions described in Section 2 and for the potential of optimization tooling in proof assistants.

Our tool will accept a DCP problem specified in Lean and translate it into conic form while verifying that any solution of the conic problem yields a solution of the original problem. In this paper, we will focus on this first task. We envision that the tool will then send the conic problem to an external conic optimization solver. The solver will return a solution, along with a dual solution that will allow us to verify the correctness of the result independently in Lean.

We have considered sending the DCP problem directly to CVX or a similar high-level modeling system. However, to the best of our knowledge, the dual solutions that CVX provides do not allow us to verify the result independently, without canonizing the problem to conic form in Lean. Moreover, experimenting with these transformations will help us to prepare the groundwork for more complex problem transformations that cannot be handled fully automatically.

We demonstrate a problem transformation in Lean using the DCP program ($\star$) above. The transformation is shown in Figure 1. Ultimately, we would like to fully automatize such DCP transformations and to require

```
def prob₁ := minimization          def prob₂ := minimization          def prob₃ := minimization
  !vars x y                          !vars t₁ x y                       !vars t₂ t₁ x y
  !objective c * x                   !objective c * x                   !objective c * x
  !constraints                       !constraints                       !constraints
    exp y                              exp y ≤ t₁,                        t₂ ^ 2 ≤ x,
      ≤ log (a * sqrt x + b),          t₁ ≤ log (a * sqrt x + b),        exp y ≤ t₁,
    a * x + b * y = d,                 a * x + b * y = d,                t₁ ≤ log (a * t₂ + b),
    0 ≤ x,                             0 ≤ x,                            a * x + b * y = d,
    0 < a * sqrt x + b                 0 < a * sqrt x + b                0 ≤ x,
                                                                         0 < a * t₂ + b


def prob₄ := minimization          def prob₅ := minimization
  !vars t₃ t₂ t₁ x y                 !vars t₃ t₂ t₁ x y
  !objective c * x                   !objective c * x
  !constraints                       !constraints
    exp t₃ ≤ a * t₂ + b,               exp t₃ ≤ a * t₂ + b,
    t₂ ^ 2 ≤ x,                        t₂ ^ 2 ≤ x,
    exp y ≤ t₁,                        exp y ≤ t₁,
    t₁ ≤ t₃,                           t₁ ≤ t₃,
    a * x + b * y = d,                 a * x + b * y = d
    0 ≤ x,
    0 < a * t₂ + b
```

Figure 1: A DCP transformation in Lean

```
def linearization_antimono (D : Type)        def graph_expansion_concave (D : Type)
(f g : D → ℝ) (c : ℝ → D → Prop)             (f g : D → ℝ) (c d : ℝ → D → Prop)
(h_mono: ∀ x r s, r ≤ s → c s x → c r x) :   (h_mono: ∀ x r s, r ≤ s → c r x → c s x)
Minimization.Reduction                       (hg : ∀ x v, c v x
{ Domain := D,                                    → isGreatest {y | d y x} (g x)) :
  objFun := f,                               Minimization.Reduction
  constraints := fun x => c (g x) x }        { Domain := D,
{ Domain := ℝ × D,                             objFun := f,
  objFun := fun y => f y.2,                     constraints := fun x => c (g x) x }
  constraints :=                             { Domain := ℝ × D,
    fun y => g y.2 ≤ y.1 ∧ c y.1 y.2 }         objFun := fun y => f y.2,
                                               constraints :=
                                                 fun y => d y.1 y.2 ∧ c y.1 y.2 }
```

Figure 2: Examples of transformation lemmas

user interaction only for more complex reductions. The program $(\star)$ is formulated above as found in Grant's thesis [Gra04]. Grant assumes that $\log(x)$ for $x \leq 0$ and $\sqrt{x}$ for $x < 0$ take value $-\infty$. We could do the same in Lean, but algebraically the extended real numbers are not a convenient number system to work with and Lean's library is substantially more comprehensive for the reals. Instead, we add the constraints $0 \leq$ x and $0 <$ a * sqrt x + b explicitly, resulting in the definition of prob₁.

The first transformation step, from prob₁ to prob₂, consists of moving exp y into a separate constraint. Grant calls this kind of transformation *linearization*. The occurrence of exp y is replaced by an auxiliary variable t_1 and we add the constraint exp y ≤ t₁. The transformation can be justified by the lemma linearization_antimono displayed in Figure 2. This lemma allows us to define a reduction from prob₁ to prob₂. The condition for the transformation is that exp y occurs in an antimonotone context—i.e., if the constraints hold for some value s of exp y, then they also hold for values smaller than s. There is an analogous lemma linearization_mono for monotone contexts that would introduce the constraint $t_1 \leq$ exp y instead.

Second, from prob₂ to prob₃, we eliminate the occurrence of sqrt by replacing it by its *graph implementation*. A graph implementation is a description of a concave [convex] function as a convex maximization [minimization] problem. For example, for $x \geq 0$, $\sqrt{x}$ can be described as the greatest number y such that $y^2 \leq x$. The lemma graph_expansion_concave in Figure 2 justifies the process of replacing a concave function by its graph implementation, called *graph expansion*. In our case, we replace both occurrences of sqrt x by an auxiliary variable t_2 and add the constraint t₂ ^ 2 ≤ x. We can construct a reduction from prob₂ to prob₃. As for

linearization, the condition for graph expansion is that `sqrt x` occurs in a monotone context. In fact, linearization is a special case of graph expansion using the trivial graph implementation defining $f(x)$ as the greatest number y such that $y \leq f(x)$. Next, we perform a graph expansion on `log (a * t`$_2$` + b)`, yielding `prob`$_4$ and a reduction from `prob`$_3$ to `prob`$_4$.

Once `log` and `sqrt` have been eliminated, the constraints $0 \leq$ `x` and $0 <$ `a * t`$_2$` + b` have served their purpose and can be removed, yielding `prob`$_5$. Using the constraints `exp t`$_3$ $\leq$ `a * t`$_2$` + b` and `t`$_2$ `^ 2` $\leq$ `x`, it is easy to show that these constraints are redundant. For this step, we can even show that `prob`$_4$ `=` `prob`$_5$.

Finally, problem `prob`$_5$ can be written in conic form. The constraint `a * x + b * y = d` constitutes the linear component of the conic form. We can write `t`$_1$ $\leq$ `t`$_3$ as a nonnegative orthant constraint; `t`$_2$ `^ 2` $\leq$ `x` as a second-order cone constraint; `exp t`$_3$ $\leq$ `a * t`$_2$` + b` and `exp y` $\leq$ `t`$_1$ as an exponential cone constraint. The product of these cones constitutes the cone K of the conic form.

We have proven the above transformation correct in Lean by manually invoking the lemmas and proving the side conditions. Our goal is to fully automatize DCP canonization. To this end, we will need a library of graph implementations, a tactic for proving monotonicity, a tactic to derive the actual conic form from the fully graph expanded problem, and an overarching tactic to guide the transformation process.

4 Future Plans

Our approach will reveal its full potential when dealing with problem transformations that are hard to automate because they are specific to a particular problem. The DCP methodology relies on experts to program the necessary graph implementations into the system. We believe that a verified toolbox makes it easier for users—both experts and novices—to extend the system to handle new reductions and to get the details right. For extensions of the DCP methodology such as disciplined convex-concave programming [SDGB16], disciplined geometric programming [ADB19], and disciplined quasiconvex programming [AB20], this is even more crucial. For instance, disciplined convex-concave programming requires information about sub- and supergradients, and quasiconvex programming requires representations of the sublevel sets of the employed quasi-convex functions. Another example is the barrier method that requires the user to come up with appropriate barrier functions. It is impossible to devise a library that includes all functions that users will ever need, but the verified approach provides a safe environment to derive the required information interactively.

We aim to test and evaluate our toolset with concrete applications. Optimization and feasibility tools are often used in control theory to establish stability and asymptotic stability of systems, as well as safety properties. There is now a substantial literature on the use of formal methods to support this, and, in particular, to develop ways of reliably reducing verification problems to problems that can be checked by symbolic and numeric methods. We believe that a system like the one we are developing can contribute in two ways: first, by providing a general mathematical library and tools to verify the soundness of the theoretical reductions, and second, by providing an interactive tool for *applying* the reductions to specific problems, ensuring that the data is in the right form and that the side conditions are met.

For example, a recent paper by Wang et al. [WCX$^+$21] that we might explore in a case study uses optimization to synthesize invariants of hybrid systems and thereby prove safety over the infinite time horizon. The synthesized invariant is a barrier certificate, encoded as an optimization problem constrained by bilinear matrix inequalities. To make the problem amenable to conic solvers, Wang et al. transform the inequalities into difference-of-convex constraints. Through a transformation resembling disciplined convex-concave programming, they bring the problem into conic form. Finally, they use the branch-and-bound framework to ensure finding the global optimum, not a local one. We consider these transformation to be an excellent case study for our approach because the transformations are practical and hard to automate.

Related work

Formal methods have been used to solve bounding problems [RS07, GAC12], constraint satisfaction problems [FHT$^+$07], and optimization problems [KSG18]. The literature is too large to cover here; [DS19] surveys some of the methods that are used in connection with the verification of cyber-physical systems.

Proof assistants have been used to verify bounds in various ways. Some approaches use certificates from numerical packages; Harrison [Har07] uses certificates from semidefinite programming in HOL Light, and Magron et al. [MAGW15] and Dorel and Roux [MR17] use similar certificates in Coq. Solovyev and Hales use a combination of symbolic and numeric methods in HOL Light [SH13]. Other approaches have focused on verifying symbolic and numeric algorithms instead. For example, Muñoz, Narkawicz, and Aaron Dutle [MND18] verify a decision

procedure for univariate real arithmetic in PVS and Cordwell, Tan, and Platzer [CTP21] verify another one in Isabelle. Narkawicz and Muñoz [NM13] devised a verified numeric algorithm to find bounds and global optima.

Although the notion of a convex set has been formalized in a number of theorem provers, we do not know of any substantial formal development of convex analysis, or any project that has formalized reductions between optimization problems.

Acknowledgements

References

[AB20]   Akshay Agrawal and Stephen P. Boyd. Disciplined quasiconvex programming. *Optim. Lett.*, 14(7):1643–1657, 2020.

[ADB19]   Akshay Agrawal, Steven Diamond, and Stephen P. Boyd. Disciplined geometric programming. *Optim. Lett.*, 13(5):961–976, 2019.

[AVDB18]   Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *J. Control and Decision*, 5(1):42–60, 2018.

[BV14]   Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2014.

[CTP21]   Katherine Cordwell, Yong Kiam Tan, and André Platzer. A verified decision procedure for univariate real arithmetic with the BKR algorithm. In Liron Cohen and Cezary Kaliszyk, editors, *Interactive Theorem Proving (ITP 2021)*, volume 193 of *LIPIcs*, pages 14:1–14:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[DB16]   Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *J. Machine Learning Research*, 17(83):1–5, 2016.

[dMKA+15]   Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction (CADE-25)*, volume 9195 of *LNCS*, pages 378–388. Springer, 2015.

[DS19]   Jyotirmoy V. Deshmukh and Sriram Sankaranarayanan. *Formal Techniques for Verification and Testing of Cyber-Physical Systems*, pages 69–105. Springer International Publishing, Cham, 2019.

[FHT+07]   Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. *J. Satisf. Boolean Model. Comput.*, 1(3-4):209–236, 2007.

[FNB20]   Anqi Fu, Balasubramanian Narasimhan, and Stephen Boyd. CVXR: An R package for disciplined convex optimization. *Journal of Statistical Software*, 94(14):1–34, 2020.

[GAC12]   Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. δ-complete decision procedures for satisfiability over the reals. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning (IJCAR 2012)*, volume 7364 of *LNCS*, pages 286–300. Springer, 2012.

[GB08]   Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, volume 371 of *LNCIS*, pages 95–110. Springer, 2008.

[GB14]   Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[GBY06]   Michael Grant, Stephen Boyd, and Yinyu Ye. Disciplined convex programming. In *Global optimization*, pages 155–210. Springer, 2006.

[Gra04]   Micheal Charls Grant. *Disciplined Convex Programming*. PhD thesis, Stanford University, 2004.

[Har07]   John Harrison. Verifying nonlinear real formulas via sums of squares. In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2007)*, volume 4732 of *LNCS*, pages 102–118. Springer, 2007.

[KSG18]   Soonho Kong, Armando Solar-Lezama, and Sicun Gao. Delta-decision procedures for exists-forall problems over the reals. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification (CAV 2018, Part II)*, volume 10982 of *LNCS*, pages 219–235. Springer, 2018.

[Löf04]   J. Löfberg. Yalmip : A toolbox for modeling and optimization in matlab. In *Computer Aided Control System Design (CACSD 2004)*, pages 284–289, 2004.

[MAGW15]   Victor Magron, Xavier Allamigeon, Stéphane Gaubert, and Benjamin Werner. Formal proofs for nonlinear optimization. *J. Formaliz. Reason.*, 8(1):1–24, 2015.

[Mat20]   Mathlib Community. The lean mathematical library. In Jasmin Blanchette and Catalin Hritcu, editors, *Certified Programs and Proofs (CPP 2020)*, pages 367–381. ACM, 2020.

[MND18]   César A. Muñoz, Anthony J. Narkawicz, and Aaron Dutle. A decision procedure for univariate polynomial systems based on root counting and interval subdivision. *J. Formaliz. Reason.*, 11(1):19–41, 2018.

[MR17]   Érik Martin-Dorel and Pierre Roux. A reflexive tactic for polynomial positivity using numerical solvers and floating-point computations. In Yves Bertot and Viktor Vafeiadis, editors, *Certified Programs and Proofs (CPP 2017)*, pages 90–99. ACM, 2017.

[NM13]   Anthony Narkawicz and César A. Muñoz. A formally verified generic branching algorithm for global optimization. In Ernie Cohen and Andrey Rybalchenko, editors, *Verified Software: Theories, Tools, Experiments (VSTTE 2013)*, volume 8164 of *LNCS*, pages 326–343. Springer, 2013.

[RS07]   Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Trans. Embed. Comput. Syst.*, 6(1):8, 2007.

[SDGB16]   Xinyue Shen, Steven Diamond, Yuantao Gu, and Stephen P. Boyd. Disciplined convex-concave programming. In *Decision and Control (CDC 2016)*, pages 1009–1014. IEEE, 2016.

[SH13]   Alexey Solovyev and Thomas C. Hales. Formal verification of nonlinear inequalities with taylor interval approximations. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *NASA Formal Methods (NFM 2013)*, volume 7871 of *LNCS*, pages 383–397. Springer, 2013.

[Stu99]   Jos F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625–653, 1999.

[TTT99]   Kim-Chuan Toh, Michael J Todd, and Reha H Tütüncü. SDPT3—a MATLAB software package for semidefinite programming, version 1.3. *Optimization methods and software*, 11(1-4):545–581, 1999.

[UdM20]   Sebastian Ullrich and Leonardo de Moura. Beyond notations: Hygienic macro expansion for theorem proving languages. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning (IJCAR 2020)*, volume 12167 of *LNCS*, pages 167–182. Springer, 2020.

[UMZ+14]   Madeleine Udell, Karanveer Mohan, David Zeng, Jenny Hong, Steven Diamond, and Stephen Boyd. Convex optimization in Julia. *SC14 Workshop on High Performance Technical Computing in Dynamic Languages*, 2014.

[WCX+21]   Qiuye Wang, Mingshuai Chen, Bai Xue, Naijun Zhan, and Joost-Pieter Katoen. Synthesizing invariant barrier certificates via difference-of-convex programming. In John Cyphert, editor, *Computer Aided Verification (CAV 2021)*, LNCS. Springer, 2021. To appear, preprint at https://arxiv.org/abs/2105.14311.