Verified Optimization (work in progress)

Alexander Bentkamp[1,2], Jeremy Avigad[3]

[1]Vrije Universiteit Amsterdam, Department of Computer Science, De Boelelaan 1111, 1081 HV Amsterdam, Netherlands
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, No. 4, South Fourth Street, Zhong Guan Cun, Beijing, 100190, China
[3]Carnegie Mellon University, Department of Philosophy, Baker Hall 161, Pittsburgh, PA 15213

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.

Keywords

optimization, proof assistants, disciplined convex programming, Lean

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

FMM 2021: Fifth Workshop on Formal Mathematics for Mathematicians, 30–31 July 2021, Timisoara, Romania

a.bentkamp@vu.nl (A. Bentkamp); avigad@cmu.edu (J. Avigad)
https://www.cs.vu.nl/~abp290/ (A. Bentkamp); https://www.andrew.cmu.edu/user/avigad/ (J. Avigad)
0000-0002-7158-3595 (A. Bentkamp); 0000-0003-1275-315X (J. Avigad)

CEUR Workshop Proceedings (CEUR-WS.org)

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 [1].

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 [2]. 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 [3]. The development of Lean 3 has mostly stabilized; its library, mathlib [4], 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 ≤ 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 [5], 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) [6, 7] 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 [6, 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 [2, 8], CVXPy [9, 10], Convex.jl [11], and CVXR [12]; a related reduction system is implemented in YALMIP [13]. 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 [14] and SDPT3 [15], 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

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

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

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

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

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

Figure 1: A DCP transformation in Lean

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 automate such DCP transformations and to require user interaction only for more complex reductions. The program $(\star)$ is formulated above as found in Grant's thesis [6]. 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 ≤ x` and `0 < a * sqrt x + b` explicitly, resulting in the definition of prob$_1$.

The first transformation step, from prob$_1$ to prob$_2$, 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 ≤` t$_1$. The transformation can be justified by the schema `linearization_antimono` in Figure 2, which is parameterized

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

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

Figure 2: Examples of reduction schemas

by a function c that is monotone in its first argument. Instantiating it appropriately yields a reduction from prob$_1$ to prob$_2$. 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 reduction schema linearization_mono for monotone contexts that would introduce the constraint t$_1$ ≤ exp y instead.

Second, to reduce prob$_2$ to prob$_3$, 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 reduction schema 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 t$_2$ ^ 2 ≤ x, which yields a reduction from prob$_2$ to prob$_3$. 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 ≤ x and 0 < a * t$_2$ + b have served their purpose and can be removed, yielding prob$_5$. Using the constraints exp t$_3$ ≤ a * t$_2$ + b and t$_2$ ^ 2 ≤ 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$_1$ ≤ t$_3$ as a nonnegative orthant constraint; t$_2$ ^ 2 ≤ x as a second-order cone constraint; exp t$_3$ ≤ a * t$_2$ + b and exp y ≤ 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 applying the reductions manually and proving the side conditions. Our goal is to fully automate 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 [16], disciplined geometric programming [17], and disciplined quasiconvex programming [18], 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. [19] 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 transformations to be an excellent case study for our approach because they are practical and hard to automate.

Related work

Formal methods have been used to solve bounding problems [20, 21], constraint satisfaction problems [22], and optimization problems [23]. The literature is too large to cover here; [24]

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 [25] uses certificates from semidefinite programming in HOL Light, and Magron et al. [26] and Martin-Dorel and Roux [27] use similar certificates in Coq. Solovyev and Hales use a combination of symbolic and numeric methods in HOL Light [28]. Other approaches have focused on verifying symbolic and numeric algorithms instead. For example, Muñoz, Narkawicz, and Dutle [29] verify a decision procedure for univariate real arithmetic in PVS and Cordwell, Tan, and Platzer [30] verify another one in Isabelle. Narkawicz and Muñoz [31] have devised a verified numeric algorithm to find bounds and global optima. Cohen et al. [32, 33] have developed a framework for verifying optimization algorithms using the ANSI/ISO C Specification Language (ACSL) [34].

Although the notion of a convex set has been formalized in a number of theorem provers, we do not know of any full development of convex analysis. The Isabelle [35] HOL-Analysis library[1] includes properties of convex sets and functions, including Carathéodory's theorem on convex hulls, Radon's theorem, and Helly's theorem, as well as properties of convex sets and functions on normed spaces and Euclidean spaces. A theory of lower semicontinuous functions by Grechuk [36] in the Archive of Formal Proofs [37] includes properties of convex functions. Lean's mathlib includes a number of fundamental results,[2] including a formalization of the Riesz extension theorem by Kudryashov and Dupuis and a formalization of Jensen's inequality by Kudryashov. Allamigeon and Katz have formalized a theory of convex polyhedra in Coq with an eye towards applications to linear optimization [38]. We do not know of any project that has formalized reductions between optimization problems.

Acknowledgments

We are grateful to Seulkee Baek, Geir Dullerud, Paul Jackson, John Miller, Ramon Fernández Mir, Ivan Papusha, and Ufuk Topcu for helpful discussions and advice. We also thank the anonymous reviewers for their corrections and suggestions.

References

[1] S. P. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2014. URL: https://web.stanford.edu/%7Eboyd/cvxbook/. doi:10.1017/CBO9780511804441.

[2] M. Grant, S. Boyd, CVX: Matlab software for disciplined convex programming, version 2.1, http://cvxr.com/cvx, 2014.

[3] L. M. de Moura, S. Kong, J. Avigad, F. van Doorn, J. von Raumer, The Lean theorem prover (system description), in: A. P. Felty, A. Middeldorp (Eds.), Automated Deduction (CADE-25), volume 9195 of *LNCS*, Springer, 2015, pp. 378–388.

[4] Mathlib Community, The lean mathematical library, in: J. Blanchette, C. Hritcu (Eds.), Certified Programs and Proofs (CPP 2020), ACM, 2020, pp. 367–381.

[1] https://isabelle.in.tum.de/dist/library/HOL/HOL-Analysis/

[2] https://github.com/leanprover-community/mathlib/tree/master/src/analysis/convex

[5] S. Ullrich, L. de Moura, Beyond notations: Hygienic macro expansion for theorem proving languages, in: N. Peltier, V. Sofronie-Stokkermans (Eds.), Automated Reasoning (IJCAR 2020), volume 12167 of *LNCS*, Springer, 2020, pp. 167–182.

[6] M. C. Grant, Disciplined Convex Programming, Ph.D. thesis, Stanford University, 2004.

[7] M. Grant, S. Boyd, Y. Ye, Disciplined convex programming, in: Global optimization, Springer, 2006, pp. 155–210.

[8] M. Grant, S. Boyd, Graph implementations for nonsmooth convex programs, in: V. Blondel, S. Boyd, H. Kimura (Eds.), Recent Advances in Learning and Control, volume 371 of *LNCIS*, Springer, 2008, pp. 95–110. URL: https://doi.org/10.1007/978-1-84800-155-8_7. doi:10.1007/978-1-84800-155-8_7.

[9] A. Agrawal, R. Verschueren, S. Diamond, S. Boyd, A rewriting system for convex optimization problems, J. Control and Decision 5 (2018) 42–60.

[10] S. Diamond, S. Boyd, CVXPY: A Python-embedded modeling language for convex optimization, J. Machine Learning Research 17 (2016) 1–5.

[11] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, S. Boyd, Convex optimization in Julia, SC14 Workshop on High Performance Technical Computing in Dynamic Languages (2014). arXiv:1410.4821.

[12] A. Fu, B. Narasimhan, S. Boyd, CVXR: An R package for disciplined convex optimization, Journal of Statistical Software 94 (2020) 1–34.

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

[14] J. F. Sturm, Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones, Optimization methods and software 11 (1999) 625–653.

[15] K.-C. Toh, M. J. Todd, R. H. Tütüncü, SDPT3—a MATLAB software package for semidefinite programming, version 1.3, Optimization methods and software 11 (1999) 545–581.

[16] X. Shen, S. Diamond, Y. Gu, S. P. Boyd, Disciplined convex-concave programming, in: Decision and Control (CDC 2016), IEEE, 2016, pp. 1009–1014.

[17] A. Agrawal, S. Diamond, S. P. Boyd, Disciplined geometric programming, Optim. Lett. 13 (2019) 961–976.

[18] A. Agrawal, S. P. Boyd, Disciplined quasiconvex programming, Optim. Lett. 14 (2020) 1643–1657.

[19] Q. Wang, M. Chen, B. Xue, N. Zhan, J. Katoen, Synthesizing invariant barrier certificates via difference-of-convex programming, in: J. Cyphert (Ed.), Computer Aided Verification (CAV 2021), LNCS, Springer, 2021. To appear, preprint at https://arxiv.org/abs/2105.14311.

[20] S. Ratschan, Z. She, Safety verification of hybrid systems by constraint propagation-based abstraction refinement, ACM Trans. Embed. Comput. Syst. 6 (2007) 8. URL: https://doi.org/10.1145/1210268.1210276. doi:10.1145/1210268.1210276.

[21] S. Gao, J. Avigad, E. M. Clarke, δ-complete decision procedures for satisfiability over the reals, in: B. Gramlich, D. Miller, U. Sattler (Eds.), Automated Reasoning (IJCAR 2012), volume 7364 of *LNCS*, Springer, 2012, pp. 286–300.

[22] M. Fränzle, C. Herde, T. Teige, S. Ratschan, T. Schubert, Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure, J. Satisf. Boolean Model. Comput. 1 (2007) 209–236. URL: https://doi.org/10.3233/sat190012. doi:10.3233/sat190012.

[23] S. Kong, A. Solar-Lezama, S. Gao, Delta-decision procedures for exists-forall problems over the reals, in: H. Chockler, G. Weissenbacher (Eds.), Computer Aided Verification (CAV 2018, Part II), volume 10982 of *LNCS*, Springer, 2018, pp. 219–235.

[24] J. V. Deshmukh, S. Sankaranarayanan, Formal techniques for verification and testing of cyber-physical systems, in: M. A. Al Faruque, A. Canedo (Eds.), Design Automation of Cyber-Physical Systems, Springer International Publishing, Cham, 2019, pp. 69–105.

[25] J. Harrison, Verifying nonlinear real formulas via sums of squares, in: K. Schneider, J. Brandt (Eds.), Theorem Proving in Higher Order Logics (TPHOLs 2007), volume 4732 of *LNCS*, Springer, 2007, pp. 102–118.

[26] V. Magron, X. Allamigeon, S. Gaubert, B. Werner, Formal proofs for nonlinear optimization, J. Formaliz. Reason. 8 (2015) 1–24. URL: https://doi.org/10.6092/issn.1972-5787/4319. doi:10.6092/issn.1972-5787/4319.

[27] É. Martin-Dorel, P. Roux, A reflexive tactic for polynomial positivity using numerical solvers and floating-point computations, in: Y. Bertot, V. Vafeiadis (Eds.), Certified Programs and Proofs (CPP 2017), ACM, 2017, pp. 90–99. URL: https://doi.org/10.1145/3018610.3018622. doi:10.1145/3018610.3018622.

[28] A. Solovyev, T. C. Hales, Formal verification of nonlinear inequalities with Taylor interval approximations, in: G. Brat, N. Rungta, A. Venet (Eds.), NASA Formal Methods (NFM 2013), volume 7871 of *LNCS*, Springer, 2013, pp. 383–397.

[29] C. A. Muñoz, A. J. Narkawicz, A. Dutle, A decision procedure for univariate polynomial systems based on root counting and interval subdivision, J. Formaliz. Reason. 11 (2018) 19–41. URL: https://doi.org/10.6092/issn.1972-5787/8212. doi:10.6092/issn.1972-5787/8212.

[30] K. Cordwell, Y. K. Tan, A. Platzer, A verified decision procedure for univariate real arithmetic with the BKR algorithm, in: L. Cohen, C. Kaliszyk (Eds.), Interactive Theorem Proving (ITP 2021), volume 193 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 14:1–14:20.

[31] A. Narkawicz, C. A. Muñoz, A formally verified generic branching algorithm for global optimization, in: E. Cohen, A. Rybalchenko (Eds.), Verified Software: Theories, Tools, Experiments (VSTTE 2013), volume 8164 of *LNCS*, Springer, 2013, pp. 326–343.

[32] R. Cohen, G. Davy, E. Feron, P.-L. Garoche, Formal verification for embedded implementation of convex optimization algorithms, IFAC-PapersOnLine 50 (2017) 5867–5874. 20th IFAC World Congress.

[33] R. Cohen, E. Feron, P. Garoche, Verification and validation of convex optimization algorithms for model predictive control, Journal of Aerospace Information Systems 17 (2020) 257–270.

[34] P. Baudin, P. Cuoq, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, V. Prevosto, Acsl: Ansi/iso c specification language, 2020. URL: https://frama-c.com/html/acsl.html, version 1.17.

[35] T. Nipkow, L. C. Paulson, M. Wenzel, Isabelle/HOL - A Proof Assistant for Higher-Order Logic, volume 2283 of *LNCS*, Springer, 2002.

[36] B. Grechuk, Lower semicontinuous functions, Archive of Formal Proofs (2011). https://isa-afp.org/entries/Lower_Semicontinuous.html, Formal proof development.

[37] J. C. Blanchette, M. W. Haslbeck, D. Matichuk, T. Nipkow, Mining the archive of formal proofs, in: M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, V. Sorge (Eds.), Intelligent Computer

Mathematics (CICM 2015), volume 9150 of *LNCS*, Springer, 2015, pp. 3–17.

[38] X. Allamigeon, R. D. Katz, A formalization of convex polyhedra based on the simplex method, J. Autom. Reason. 63 (2019) 323–345.