RICE UNIVERSITY

Program Analysis and Transformation in Mathematical Programming

by

Joseph G. Young

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

Dr. Yin Zhang, Professor
Computational and Applied Mathematics

Dr. Mike Fagan, Researcher
Computer Science

Dr. Richard Tapia, University Professor
Computational and Applied Mathematics

Dr. David Gay, Principal Member
of Technical Staff
Sandia National Laboratory

Dr. Keith Cooper, Professor
Computer Science

HOUSTON, TEXAS

MAY 2008

Abstract

Over the years, mathematical models have become increasingly complex. Rarely can we accurately model a process using only linear or quadratic functions. Instead, we must employ complicated routines written in some programming language. At the same time, most algorithms rely on the ability to exploit structural features within a model. Thus, our ability to compute with a model directly relates to our ability to analyze it.

Mathematical programs exemplify these difficult modeling issues. Our desire to accurately model a process is mediated by our ability to solve the resulting problem. Nonetheless, many problems contain hidden structural features that, when identified, allow us to transform the problem into a more computable form. Thus, we must develop methods that not only recognize these hidden features, but exploit them by transforming one problem formulation into another.

We present a new domain specific language for mathematical programming. The goal of this language is to develop a system of techniques that allow us to automatically determine the structure of a problem then transform it into a more desirable form. Our technical contribution to this area includes the grammar, type system, and semantics of such a language. Then, we use these tools to develop a series of transformations that manipulate the mathematical model.

Acknowledgments

I would like to begin by thanking Dr. Walid Taha for his assistance. Dr. Taha introduced me to the type-centric approach of designing computer languages. His assistance allowed me to properly frame my problem and it strengthened my resolve to see this project to completion.

I give my deepest possible gratitude to Dr. Mike Fagan. Dr. Fagan helped me understand many key underlying properties of computer languages. He guided the transformation of my design from one that was intractable into another that was clear and practical. Most importantly, he provided invaluable assistance during many difficult and challenging moments of my research.

I would like to thank my adviser Dr. Yin Zhang. Dr. Zhang helped me in an innumerable number of ways during my tenure at Rice. His tutelage expanded my mathematical knowledge and profoundly influenced my understanding and perception of mathematical programs.

I am indebted to my departmental chair Dr. Dan Sorensen. Dr. Sorensen provided invaluable advice during the trials of graduate study. His administrative acumen helped insure my success.

I extend my loving gratitude to my parents John and Jewell and my brothers Daniel and Joshua. Dad taught me the importance of resolve while traversing turbulent waters. Mom demonstrated that a small, but formidable voice can result in an unlikely executive decision. Daniel provided a voice of reason when I began to wade into a hazardous mire.

Joshua showed me the importance of a silver-tongue.

Finally, I would like to thank my friend Steve for teaching me to work under pressure. I would also like to thank my friends John, Lena, and Jennifer for appreciating the gravity of my challenges and their valuable advice on how to persevere.

Contents

Chapter 1

Introduction

Mathematical programming refers to a collection of formal methods used to model and optimize a complicated process. These processes arise in many different settings such as minimizing the operating cost of an airline[14] or describing the optimal layout of a microprocessor[6]. The goal of solving these problems is to find an optimal set of parameters that characterize the original system.

Although there exist a wide variety of tools that can solve a mathematical program, these tools rarely understand the model in its natural form. Manually transforming one form to another is difficult and error prone. Therefore, most analysts use a modeling language. Modeling languages allow a user to describe a problem in general terms. Then, this description is converted into a form that the solver understands.

Once we model a problem, we still must choose which solver to use. The structure of the problem predicates this decision. For example, when the problem contains only linear functions, we may use a solver built upon the simplex method. A different problem may necessitate a different solver. This decision becomes more difficult when these structural components are not explicit. Frequently, we must transform one formulation to another in

order to take advantage of certain properties.

This problem becomes more challenging when we model our process with a general purpose code such as C. In this case, the mathematical properties of the routine are well hidden. As a result, we may be forced to use less powerful algorithms than what the problem dictates. Alternatively, we may be forced to transform the routine into some auxiliary form by hand.

As a result, we must develop the capacity to automatically analyze a general purpose routine. Then, we can transform the result into an alternative form. Of course, this is a very difficult goal. Thus, as a first step, we can instead develop and apply the same sort of tools and techniques to a modeling language. Simply, modeling languages represent a restricted subclass within general purpose routines.

We propose a new domain specific language for mathematical programming. The core of this research lies within two areas: clear, concise semantics that accurately represent a mathematical program and a rich type system that allows us to ascertain the mathematical properties of a particular problem instance. Based on this foundation, we define a series of transformations and prove their correctness.

1.1 General Purpose Routines

When we refer to the analysis of a general purpose routine, we mean the following. Consider the C function

```
double linear(double *x){
```

```
    double y=0.;
    int i=0;
    for(i=0;i<=5;i++)
        y+=i*x[i];
    return y;
}
```

Certainly, this routine is equivalent to the function $x \mapsto x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5$. Thus, we see the routine represents a nice, linear function. Yet, virtually all linear programming solvers can not understand this form. Even if a solver could call this routine, it can not guarantee its linearity.

Ultimately, we wish to communicate this information to a solver. When a solver does not understand this form, we want to transform the problem into one the solver does. However, for the time being, we focus our efforts toward modeling languages. This same situation occurs albeit in a different form within these languages.

1.2 Transformation

Transformations provide us one powerful technique for taking advantage of structure. In the simplest possible case, let us consider a problem of the form

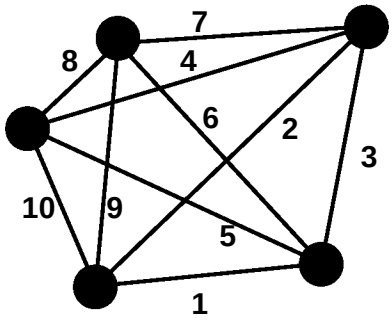
$$\min_{x \in \mathbb{R}} ax \quad \text{st} \quad bx \geq |x|$$

This problem is almost linear, but the constraint contains a nonlinear, nondifferentiable term, $|x|$. If we recall the definition of absolute value, this constraint states that $bx \geq \max(x, -x)$. However, when bx is greater than the maximum of both x and $-x$, it must be greater than both terms separately. This allows us to reformulate the problem into

$$\min_{x \in \mathbb{R}} ax \quad \text{st} \quad bx \geq x$$
$$bx \geq -x$$

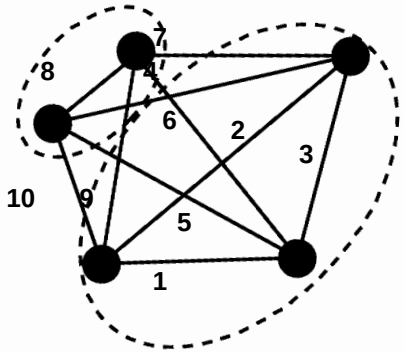
In this new problem, all functions are linear. Therefore, we can employ much more powerful algorithms to solve this formulation than the first. Now, we may question whether we need a computer tool to convert the constraint $bx \geq |x|$ into $bx \geq x$ and $bx \geq -x$. Certainly, we can easily recognize and manipulate this constraint by hand. However, in many cases this situation is less clear.

Let us consider the max-cut problem from graph theory. Consider a graph whose edges contain nonnegative weights. When two nodes are disconnected, we assume an edge exists with weight zero. For example, the following diagram denotes a suitable graph



The goal of the max-cut problem is to divide the nodes of a graph into two sets such that we maximize the sum of the weights that pass between the two sets. Using the same graph

as above, the following denotes a partition



The weight of this cut is 41. In order to model this problem, label each node in the graph with a unique index. Then, let w_{ij} denote the weight of an edge that connects node i to node j. This allows us to model the problem as

$$\max_{x\in\{-1,1\}^n} \frac{1}{4}\sum_{ij}(1 - x_i x_j)w_{ij}$$

The designation of x_i as 1 or -1 denotes whether node i belongs to the first or second set. Notice that when x_i and x_j have the same sign, the term $1 - x_i x_j = 0$ and w_{ij} does not contribute to the weight of the cut. When we want an upper bound on the solution, we can alternatively solve the following semidefinite program

$$\begin{aligned} \max_{x\in\mathbb{R}^n, X\in\mathcal{S}^n} \quad & -\tfrac{1}{4}\mathrm{tr}(WX) + \tfrac{1}{4}\mathrm{tr}(WE) \\ \text{st} \quad & X_{ii} = 1 \\ & \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \geq 0 \end{aligned}$$

where $W_{ij} = w_{ij}$ and E denotes the matrix of all ones. Since this problem is nice and

convex, it is vastly easier to solve than the first. However, this representation looks nothing like the original problem. In fact, it provides no indication that it is related to a max-cut problem at all. Nevertheless, there exists an intimate connection between the two and it is possible to derive the second formulation from the first in a mechanical fashion.

This example highlights two difficulties that arise during modeling. First, an algebraic model should be correspond closely to the specification of the problem. This helps us easily spot and diagnose problems in the original formulation. Second, any change the original problem necessitates a change to the reformulation. When the reformulation from one problem to another is complicated, filtering through any changes can be difficult.

1.3 Analysis

In many cases, identifying structural features requires more work. In the following example, we discuss the automated transformation of a constraint containing the chained singular function[20]

$$c(x) = \sum_{i \in J} (x_i + 10x_{i+1})^2 + 5(x_{i+2} - x_{i+3})^2 + (x_{i+1} - 2x_{i+2})^4 + 10(x_i - 10x_{i+3})^4$$

where $J = \{1, 5, \ldots, n-3\}$ and n is a multiple of 4. It is known that a constraint of the form $y \geq c(x)$ can be transformed into a second-order cone constraint[51]. Let us investigate one possible reformulation.

In the most simple case, we have a constraint of the form

$$y \geq (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - 10x_4)^4$$

By introducing two auxiliary variables, we can reformulate this constraint into

$$y \geq (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + u + v$$
$$u \geq (x_2 - 2x_3)^4$$
$$v \geq 10(x_1 - 10x_4)^4$$

Now, since the square of any real number is positive, we can further expand this system into

$$y \geq (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + u + v$$
$$u \geq s^2$$
$$s \geq (x_2 - 2x_3)^2$$
$$v \geq t^2$$
$$t \geq 10(x_1 - 10x_4)^2$$

At this point we notice that the variables u and v serve mostly as place holders. Thus, we

can simplify this system slightly into

$$y \geq (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + s^2 + t^2$$

$$s \geq (x_2 - 2x_3)^2$$

$$t \geq 10(x_1 - 10x_4)^2$$

Before we continue, let us consider a transformation common to convex cone programming. Consider a constraint of the form

$$0 \geq x^T A^T A x + a^T x + \alpha$$

Through clever manipulation, we can reformulate the constraint into

$$\begin{bmatrix} 1 - c^T x - \gamma \\ 2Ax \\ 1 + c^T x + \gamma \end{bmatrix} \succeq_Q 0$$

where $\succeq_Q$ represents the partial order defined by the second-order cone. In other words, we can transform a convex quadratic constraint into a second-order cone constraint. This new formulation is desirable since it allows us to employ special algorithms designed for second-order cone programs.

Now, let us return to our original problem. It turns out that each of the constraints

$$y \geq (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + s^2 + t^2$$

$$s \geq (x_2 - 2x_3)^2$$

$$t \geq 10(x_1 - 10x_4)^2$$

is both convex and quadratic. Therefore, we can reformulate them into second-order cone constraints. However, they do not possess the nice canonical form

$$0 \geq x^T A^T A x + a^T x + \alpha$$

Certainly, we should not constrain the user to this representation. Further, we claim it is unreasonable to even ask them to prove whether a function is convex or quadratic. It is a property that may be difficult to determine. As a result, we must analyze and determine this property automatically.

1.4 Overview

Our solution to these problems lies in the design of a new modeling language. The design consists of four separate parts: grammar, type system, semantics, and transformations.

The grammar specifies the internal structure of a mathematical program. For example, we specify that all problems must begin with the keyword *min* followed by the objective function. The objective function must be followed by the keyword *over* followed by vari-

ables, etc. While this specification is rigid, it serves as an analytic vessel. Thus, the user may specify a problem in a much friendlier syntax.

Next, we define a type system. The type system serves two purposes. First, it insures that a mathematical program is well-formed. In other words, it prevents absurd statements such as $y \geq x^{u \geq v}$. Second, we use the type system to assert properties about our problem. This is how we determine whether a function is convex, a polynomial, or monotonic.

Once we define the type system, we specify the semantics of our language. Up until this point, we essentially define a series of rules that manipulate grammar. The semantics assign an unambiguous mathematical meaning to this grammar. Once we define this meaning, we prove that our system is consistent. In other words, we prove that all well-typed programs correspond to actual mathematical programs. We also prove that when our type system asserts a certain property such as convexity, this property must hold . This gives us confidence that our design is correct.

Finally, we use this machinery to design a series of transformations. Each transformation maps one piece of grammar to another. Then, we use the semantics to prove that this reformulation is correct. Although their are an innumerable number of possible transformations, we focus on those necessary to reformulate our examples above. This provides good balance of transformations that reformulate convex and nonconvex problems.

Chapter 2

History

The study of modeling languages for optimization combines research from optimization, mathematical modeling, semantics, and type systems. As each of these areas possesses its own unique history, we do not provide a comprehensive chronicle of their results. Instead, we summarize many of the important developments of each area and highlights how these ideas intersect.

2.1 Optimization

Modern methods for computational optimization originated in 1947 with George Dantzig [25, 26]. At the time, Dantzig was tasked with solving transportation problems for the Airforce. These problems required a scheduler to optimize a linear objective function bounded by linear constraints. In order to solve this problem more efficiently, Dantzig developed an algorithm called the simplex method. The simplex method finds the solution of a linear program by traversing the vertices of the feasible region. Although the algorithm is combinatorial in nature and may run in exponential time, the algorithm finds a solution very quickly in practice. To this day, it remains one of the most important algorithms to solve linear

programs. It is implemented by many high performance packages such as CPLEX[12].

Since the simplex method relies on the structure of a linear program, it is not well-suited toward nonlinear programming. Thus, algorithms that solve nonlinear programs developed relatively independently from linear programming. The conditions for optimality of a constrained nonlinear program were developed by Karush in 1939 during his Masters thesis[47]. Khuhn and Tucker rediscovered this result in 1951[52]. Later, these conditions became known as the KKT conditions. This work was pivotal since it characterized conditions for optimality as a system of nonlinear equations that could be solved using Newton's method. During the 1960's, penalty and barrier methods became popular[31], but fell out favor since they lacked numerical stability. During the 1970's, sequential quadratic programming[74, 38, 13] and augmented lagrangian methods[43] were developed and they remain popular to this day. However, one of the most important developments in optimization history came during the 1980's with the interior point revolution[75].

In 1979 Khachian developed the ellipsoid method for linear programming[48]. It was the first algorithm developed that solved a linear program in polynomial time. It differed from the simplex method since it generalized ideas from nonlinear programming and did not rely the the combinatorial nature of the feasible region. Unfortunately, the algorithm did not perform well in practice and fell out of favor. In 1984, Karmarkar announced a new polynomial time algorithm for linear programming[46] that was far more efficient than the simplex method in practice. Although details of his work remained absent at the time, his algorithm was later shown to be equivalent to a logarithmic barrier function applied

to a linear program[39]. In practice, the algorithm was efficient, but not as revolutionary as originally announced. Nevertheless, it was a very important discovery since it began the unification between the theory of linear and nonlinear programming. Algorithms that used this new approach became known as interior point methods. Since the 1980's, interior point methods became an important and popular method for both linear and nonlinear programming.

In 1988 Nesterov and Nemirovski proved that polynomial-time algorithms could be extended to convex programs[60]. This led to the study of semidefinite and second-order cone programs during the 1990's[72]. This work was significant since it was the first time an efficient algorithm existed that found the global solution of a broad class of nonlinear programs outside of geometric programming[29, 28, 30]. In fact, as these algorithms became more mature, the distinction between linear and nonlinear programming became less pronounced. Instead, it became useful to distinguish between convex and nonconvex programs.

Although convex programs can be solved in polynomial time, efficient solvers based on this theory have only been realized for linear, second-order cone, and semidefinite programs. Although many high quality solvers exist for linear and semidefinite programming [15, 71, 36, 8, 69], only two can additionally solve second-order cone programs. As an alternative approach, some solvers can solve general convex programs very quickly using specialized algorithms from nonlinear programming[1], but generally can not represent matrix constraints necessary for semidefinite programming.

2.2 Mathematical Modeling

As algorithms to solve optimization problems evolved, so did the technology to represent them. Prior to the 1970's, interfacing to solvers was largely problem and solver dependent. In the late 1970's and early 1980's MPS files became the standard format to represent problems. MPS files store a problem using well-defined, very rigorous format. Since the format was not designed to create problems by hand, another application called a matrix generator[7] was needed to create an MPS file. Frequently, matrix generators were custom Fortran programs.

In the late 1970's, the World Bank funded the development of an algebraic modeling language for optimization called GAMS[11, 17]. This approach differed from that of matrix generators since it allowed a mathematical model to be represented in its algebraic form at a very high level[32]. It also allowed users to represent their problem over sets of data rather than rigidly defined indices. This improved the model's readability and reliability. Over the years, GAMS continued to evolve as it incorporated additional algebraic structures. It remains one of the more popular modeling languages.

In the early 1980's, Bell Labs became interested in small specialized programming languages for specific applications. After Karmarkar made his announcement of a new polynomial time algorithm for linear programming, David Gay, Bob Fourer, and Brian Kernighan began work on a language called AMPL[33, 34]. Similar to GAMS, the goal of AMPL was to provide a high-level algebraic description of a problem. A key feature of this was a flexible approach to defining sets of data. As AMPL developed, it supported many

new features such as stochastic programming. It remains a popular tool to this day.

In 1989, Peter Piela developed a new object-oriented, strongly typed modeling language for chemical processes called ASCEND[62]. Piela was partly motivated by the lack of formal semantics in modeling languages such as GAMS. Later, this led to the formal study of ASCEND's semantics by Bhargava, Krishnan, and Piela[9, 10]. ASCEND's design lies in stark contrast to most available modeling languages. It remains one of the only modeling languages formally designed, studied, and still used.

In 1990, Fernando Vicuna became the first person to develop formal semantics for a modeling language[73]. During his thesis, he cited inconsistencies in the semantics of GAMS, LINGO, and AMPL. Motivated by these problems, he developed the formal semantics of a modeling language called SML using attributed grammars. While the language SML never became popular, this work remains important since it demonstrated the necessity and utility of formal techniques applied to the design of a modeling language.

Although Vicuna and Bhargava et al. were the first to study the semantics of modeling languages, they were not the last. In 1994, Neustadter analyzed the semantics of what he termed executable modeling languages[61]. This encapsulated any algebraic modeling language processed by a computer. In 1995, Hong and Mannino studied the denotational semantics of the Unified Modeling Language, $\mathcal{L}_U$[44]. This work was notable since it was the first time the denotational style of semantics was applied to a modeling language.

In 1994, Gahinet, Nemirovskii, Laub, and Chilali developed the LMI toolbox[37]. The toolbox was implemented as an object oriented library inside of MATLAB. Up until this

point, there was no tool available that allowed the algebraic formulation of a semidefinite program. In addition to the modeling utility, the toolbox came packaged with a solver based on Nemirovskii and Gahinet's projective algorithm[59]. Unfortunately, the solutions produced by the solver were extremely poor. Therefore, the toolbox fell out of favor once alternative solvers became available.

In 2001, Johan Löfberg developed a MATLAB modeling tool called YALMIP[56]. Originally, it was designed to allow the algebraic specification of linear matrix inequalities. Later, it was extended to model a broad class of optimization problems. Unlike, the LMI toolbox, YALMIP was not tied to a particular solver. This led to a much larger user base than the LMI toolbox.

In 2005, Michael Grant developed a new language called CVX during his dissertation at Stanford[41, 42]. Although YALMIP could model convex cone programs, formulating them was often difficult. Frequently, a cone program was created by reformulating a nonlinear program. CVX automated many of these transformations. Michael advocated an idea called disciplined convex programming. In this approach, new models are created from a toolbox of functions that are known to be convex. These functions are manipulated by a series of transformations that are known to preserve convexity. His ideas were realized in a MATLAB toolbox. Subsequently, YALMIP incorporated many of these techniques.

2.3 Semantics

The term semantics embodies three separate, but related fields in linguistics, logic, and computer science. In each case, the goal of semantics is to assign meaning to some object. In linguistics, the goal is to assign meaning to spoken language[16]. In logic, foundational questions are asked about meaning of truth. In computer science, meaning is assigned to a piece of code.

The semantics of logic originated in article by Frege[35] written in 1892. Frege discussed how the statement $a = b$ differed from that of $a = a$. In order to analyze this question, he differentiated between the sense of a statement and its nominatum. Although the nominatum of sentence could be defined in many different ways, he argued the correct definition was the statement's truth value. Then, he reduced the question to whether one sentence could be replaced with another. He concluded that even if two statements had the same truth value, if they differed in sense, they were not the same. Therefore, the statement $a = b$ was not equivalent to $a = a$ even when a and b possessed the same truth value.

Later, Tarski considered the semantic meaning of truth[70]. In his discussion, he analyzed the liar's paradox, "This statement is false." He argued the paradox arose because it was posed in a semantically closed language. A semantically closed language contained, in addition to its expressions, the names of the expressions as well as the concept of truth. In this way, a sentence could refer to the truth of itself. To resolve this antinomy, he employed two separate languages: the object language and the meta-language. The object language was the language under discussion. The meta-language was the language used to talk about

the first. In this framework, the liar's paradox could not be posed.

The formal semantics of computer languages arose out of necessity. In 1960, the report for Algol 60 was first published[3]. The language introduced many new features, but the semantics were given in prose. As a result, there existed many ambiguities in the design. Later, a revised report was published in an attempt to resolve these ambiguities[4]. Even after the revised report, many problems still existed[50]. It became clear, that formal methods were necessary to specify the semantics of a language.

In 1964, Landin noticed a many similarities between lambda calculus and programming languages[53]. Later, he commented more specifically about the similarities with Algol[54, 55]. He postulated that the semantics of Algol 60 could be specified by lambda calculus. In essence, the discussion of semantics would be reduced to describing the object language by the meta-language. This idea influenced the design of many other languages such as PL/I[57].

In 1971, Scott and Strachley reconsidered the use of lambda calculus as a meta-language [67, 66, 68]. Although lambda calculus provided a relatively clear semantics, they viewed it as an operational calculus rather than a mathematical semantics. Instead, they defined a series of rigorously defined domains. These domains included both syntax and mathematical objects. Then, they defined a set of functions that mapped one domain to another. These functions mapped a piece of syntax to its meaning. By restricting themselves to partially ordered domains that contained a bottom element and monotonic, continuous functions, they could model even non-terminating, recursive functions. Their work formed the foundation

of denotational semantics.

In 1985, Cousineau, Curien, and Mauny developed a new kind of denotational semantics for languages based on lambda calculus[22]. They noticed that cartesian closed categories provided the correct framework for modeling lambda calculus. As a result, they could map a program to a category. A morphism in the category could be thought of as a series of commands executed by a compiler. This led to the development of the OCaml programming language[58, 2, 63].

2.4 Type Systems

The theory of types originated when Bertrand Russell proposed a paradox within the framework of Frege's set theory[64, 65]. He found that the set of all sets that do not contain themselves led to a contradiction. In order to solve this antinomy, he proposed a hierarchy of propositional functions. The lowest level contained propositions that did not contain variables. These were called elementary propositions or individuals. The next level contained propositions whose variables ranged over terms in the level below. In this way, he avoided what he termed the, "vicious-circle principle."

In 1932, Church introduced lambda calculus as a new formal system of logic that aimed to avoid Russell's paradox[18]. Instead of using types, he divorced the law of excluded middle. He contended that this law led to Russell's and other paradoxes. Later, Church's students proved that this system was inconsistent as it was suffered from a variant of Richard's paradox[49]. Despite this difficulty, the system proved to be surprisingly

expressive. In 1940, Church incorporated type theory into lambda calculus which led to a consistent theory[19].

In 1936, Curry discovered a connection between types assignable to combinators and logical implications[23]. This connection was clarified in a paper published in 1942[24]. As the link between combinatory logic and lambda calculus became clear, this meant there was a direction connection between a program written in lambda calculus and a mathematical proof. This idea became known as the Curry-Howard correspondence[45].

Types were intimately connected to programming languages from their inception. When Fortran was designed, it contained only two types: integers and floating point numbers[5]. At the time, programmers used these types for performance rather than correctness reasons. Subsequent programming languages such as Algol and Pascal used types as a way to eliminate common programming errors. Certainly, these languages introduced many additional kinds of types such as characters or structures. However, fundamentally these types were relatively primitive.

Gordon, Milner, and Wadsworth introduced a new language called ML in 1979[40]. By this point, the connection of lambda calculus to programming languages was clear. Thus, there was great interest in creating languages suitable for proving theorems. As such, ML included a much richer variety of types such as universal quantifiers and arrow types. In addition, it did not require explicit type annotations as it was able to infer type information by employing the Hindley-Milner type inference algorithm. Originally intended for use with the proof system LCF[40], ML and its variants also formed the foundation for the

systems NuPRL[21] and Coq[27].

Chapter 3

Grammar

While we understand mathematical programming and transformations in mathematical terms, our goal is to rigorously develop a blueprint for a new language. We begin by developing the grammar of the language. The grammar defines how the language should look. For example, we can stipulate that all mathematical programs must start with either *min* or *max* followed by a function. In addition to developing grammar for the language, we must also develop grammar for a system of types. Types represent properties that an expression may possess. For example, the statement 1 + 2 has type integer which asserts that the expression must evaluate to an integer. While we develop the grammar for types now, we discuss the rules that define the type system later.

3.1 Preliminaries

Generally, there are two kinds of grammar. The abstract syntax defines an internal representation that is convenient for manipulating a program. The concrete syntax defines what a programmer must actually type. Certainly, there is a link between the two, but we focus on the abstract syntax. While this form may seem arcane at times, it is important to

stress that a programmer is not restricted to using this syntax. The concrete syntax takes a much friendlier form.

We define grammar using what is called Backus-Naur form (BNF). It is a notation that inductively defines the structure of a language. For example, we define an extremely simple calculator with the grammar

$$e \in E ::= c \mid f\{e, e\}$$

where $\{e, e\}$ denotes a tuple of expressions. This states that an expression is either a constant or a binary function. These functions can include algebraic operations such as addition, subtraction, multiplication, and division. It can also include logical operations such as *or* and *and*. Using this grammar, we can define the program

$$+\{*\{1, 2\}, 3\}$$

which we abbreviate as

$$1 * 2 + 3$$

Notice that we did not build operator precedence into this definition. This is a detail built into the concrete syntax, not the abstract.

Types also have grammar. For example, associated with the grammar above, we define types

$$t \in T ::= \mathbb{B} \mid \mathbb{Q}$$

In other words, every expression must be a boolean or an rational.

Finally, we must also define grammar for a device called a context. The context will contain type information about each of the built-in functions. We define the grammar for the context as

$$\Gamma \in G ::= \{f : \{t, t\} \to t\}_j^n$$

where $\{\}_j^n$ denotes a sequence of length n indexed by j. Using this grammar, we can define the context

$$\begin{aligned}
&\{ + : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}, \\
&\ - : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}, \\
&\ * : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}, \\
&\ / : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}, \\
&\ \mathit{and} : \{\mathbb{B}, \mathbb{B}\} \to \mathbb{B}, \\
&\ \mathit{or} : \{\mathbb{B}, \mathbb{B}\} \to \mathbb{B}\}
\end{aligned}$$

The context becomes important during the definition of typing rules.

3.2 Formal Definition

We build our mathematical program from a series of model functions. Since the type of these functions will be central to our understanding of the language, we present their grammar first.

Convexity	$c \in C ::= / \mid \smile \mid \frown \mid \perp$
Polynomality	$p \in P ::= \alpha \mid (\alpha, a) \mid (\alpha, a, A) \mid \perp$
Monotonicity	$m \in M ::= - \mid \nearrow \mid \searrow \mid \perp$
Codomain	$o \in O ::= \mathbb{R} \mid \mathbb{Z} \mid \{0, 1\} \mid \{-1, 1\}$
Symmetry	$y \in Y ::= \ddagger \mid \perp$
Types	$t \in T ::= \{\langle c, p, m, o\rangle_{ij}^{mn}, y\} \mid \eta \mid \diamond$
Domain	$d \in D ::= o^{m \times n} \mid \mathcal{S}^n$
Function Context	$\Gamma \in G ::= \{f : \{t\}_i^m \to t\}_j^n$
Decision Variable Context	$\Sigma \in S ::= \{x : d\}_i^n$

Our language determines whether a model function is convex, a polynomial, or monotonic. We represent the convexity of a function as one of four possible cases. A function is either affine, concave, convex, or something unknown. Similarly, we represent the polynomality of a function as one of four possible cases. A function is either constant, linear, quadratic, or something unknown. Notice that we not only determine whether a function is a low-order polynomial, but we also determine the coefficients associated with the function. Next, we represent the monotonicity of a function as one of four possible states. Either a function is constant, increasing, decreasing, or something unknown. In addition, we determine the codomain of each model function. The codomain must lie within the set of real, integer, zero-one integer, or plus-minus one integer numbers. We combine all four of these properties into a single type $\langle c, p, m, o\rangle_{ij}^{mn}$ which represents a matrix of the above properties of size $m \times n$ indexed by i and j. Additionally, we determine whether the codomain of the resulting

matrix function is symmetric or unknown. This combined with the naturals and constraints form the types that expressions may inhabit. Next, the domain of each model function is defined by the domain of the decision variables. This can be a matrix domain of any of the above sets or a real symmetric matrix. Finally, we define two contexts. The first contains the type of the predefined functions while the second contains the type of the decision variables. The syntax $\{x\}_i^n$ denotes a sequence of elements n long indexed by i. We define the empty sequence by {} and the concatenation of two sequences as $\{x\}_i^m, \{y\}_j^n$. In addition, we frequently omit indices when they are obvious. For example, $\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^p$ denotes a list of model function types. The notation c_{kij} denotes the (i, j)th convexity property of the kth argument.

For example, let us consider the properties of the function

$$X \in \mathbb{R}^{2\times 2}, y \in \mathbb{R}^2 \mapsto Xy = \begin{bmatrix} X_{11}y_1 + X_{12}y_2 \\ X_{21}y_1 + X_{22}y_2 \end{bmatrix} = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix}$$

This function maps $\mathbb{R}^{2\times 2} \times \mathbb{R}^2$ to $\mathbb{R}^2$. Since the codomain is not square, the result can not be symmetric. In addition, when we consider the functions f_1 and f_2, we see that each function is bilinear (quadratic), not convex, nor increasing. We represent these properties with the type

$$\left\{\begin{bmatrix} (\bot, (\alpha, a, A), \bot, \mathbb{R}) \\ (\bot, (\beta, b, B), \bot, \mathbb{R}) \end{bmatrix}, \bot\right\}$$

The variables α, a, A, β, b, and B are defined by

$$\alpha = 0 \qquad a = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 \end{bmatrix}$$

$$\beta = 0 \qquad b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \end{bmatrix}$$

Certainly, these types seem large and unwieldy. However, they give an extraordinary amount of information about the function in question. We give the rules necessary to derive the above type during the discussion of the type system.

The above types correspond to the terms

Mathematical Program $\quad n \in N ::= \min e \text{ over } \Sigma \text{ st } \{e\}_i^n$

Expressions $\quad e \in E ::= A \mid x \mid f\{e\}_i^n$

A mathematical program is built from an expression, a list of variables, and a list of expressions. Expressions can take several forms. The metavariable A ranges over constants

while x ranges over variables. The application rule allows us to build new model functions from existing ones. Notice that we don't explicitly include many useful operations such as subindexing or algebraic operations. Their functionality is subsumed by the application rule. Also notice that this language does not include definitions for user defined bindings or functions. These can be accomplished through macros. Their details are left as an implementation detail.

For example, using our grammar, we can represent the following linear program

$$\text{min } +\{x, y\} \text{ over } x : \mathbb{R}^{1\times 1}, y^{1\times 1} : \mathbb{R} \text{ st } \geq \{+\{x, y\}, 1\}$$

In the following discussion, we abbreviate this using the more readable form

$$\text{min } x + y \text{ over } x : \mathbb{R}, y : \mathbb{R} \text{ st } x + y \geq 1$$

Nonetheless, we see that this grammar can be used to represent a very broad class of nonlinear programs.

At this point, we must address a confusion of terms. In this context, we have two different kinds of variables. In one sense, we have variables that represent abstract terms in our language. For example, in the expression $x + 1$ we add the variable x to the constant 1. In another sense, we have decision variables that describe our solution. For example, in the code $\text{min } x + 1 \text{ over } x : \mathbb{R} \text{ st } \{\}$, x is a decision variable. In the following discussion, we refer to the first kind as a variable. We refer to the second kind as a decision variable.

Similarly, we have two different kinds of functions. In one sense, we have functions that map values to values in our language. For example, $\geq_{\mathbb{R}_+} \{x, 1\}$ represents the application of the $\geq_{\mathbb{R}_+}$ function. In another sense, we have functions that are built from decision variables. For example, in the code $\mathsf{min}\ x + 1\ \mathsf{over}\ x : \mathbb{R}\ \mathsf{st}\ \{\}$, $x + 1$ is a function built from a single decision variable. In fact, even x represents a function, namely $x \mapsto x$. In the following discussion, we will refer to the first kind as a function. We refer to the second kind as a model function.

Chapter 4

Type System

The type system serves two purposes. First, it insures that a program is well formed. For example, the statement $1 + \mathit{true}$ may be grammatically correct, but a good type checker should raise an error since we can not add an integer to a boolean. Second, it allows us to constructively prove properties about a statement. For example, a type system can prove that 1 + 1 is an integer. We use this idea to prove properties about the functions in a mathematical program such as whether a function is convex.

4.1 Preliminaries

Continuing our simple calculator example from above, we define the following typing rules

$$\Gamma \vdash c : \mathbb{B} \qquad \text{(Boolean Constant)}$$

$$\Gamma \vdash c : \mathbb{Q} \qquad \text{(Rational Constant)}$$

$$\frac{\hat{\Gamma}, f : \{t_1, t_2\} \to t, \hat{\Gamma}' \vdash f : \{t_1, t_2\} \to t \quad \{\Gamma \vdash e : t\}_i^2}{\Gamma \vdash f\{e_1, e_2\} : t} \qquad \text{(Application)}$$

The first two rules state that the bindings in Γ prove that a constant c is either a boolean or a rational. Of course, we assume that constants such as *true* or $false$ are boolean and

numbers are rational. The last rule states an implication. The conditions for this implication reside on the top of the line while the result lies below. It states that when the context contains a function with a particular signature and the arguments given two this function match this signature, then we know the type of the resulting application.

As an example, using the context defined during the grammar preliminaries, we use the application rule to show that 1 + 2 is rational

$$\frac{\hat{\Gamma}, + : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}, \hat{\Gamma}' \vdash + : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q} \quad \Gamma \vdash 1 : \mathbb{Q} \quad \Gamma \vdash 2 : \mathbb{Q}}{\Gamma \vdash 1 + 2 : \mathbb{Q}} \ (Add)$$

We combine these results to generate longer proofs. For example, in order to prove that 1 + 2 + 3 is rational, we have the following proof

$$\frac{\hat{\Gamma}, + : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}, \hat{\Gamma}' \vdash + : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q} \quad \dfrac{}{\Gamma \vdash 1 + 2 : \mathbb{Q}} \ (Add) \quad \Gamma \vdash 3 : \mathbb{Q}}{\Gamma \vdash 1 + 2 + 3 : \mathbb{Q}}$$

4.2 Formal Definition

The typing rules are described by

Mathematical Program $\boxed{\Gamma \vdash n}$

$$\frac{\Gamma; \Sigma \vdash e : \langle c, p, m, \mathbb{R} \rangle \quad \{\Gamma; \Sigma \vdash e_i : \diamond\}_i^n}{\Gamma \vdash \mathsf{min}\ e\ \mathsf{over}\ \Sigma\ \mathsf{st}\ \{e_i\}_i^n} \quad \text{(Prog)}$$

Expressions $\boxed{\Gamma; \Sigma \vdash e : t}$

$$\Gamma; \Sigma \vdash A : \eta \quad \text{(Index)}$$

$$\Gamma; \Sigma \vdash A : \langle /, A, -, \mathbb{R} \rangle \quad \text{(Scalar Const)}$$

$$\Gamma; \Sigma \vdash A : \left\{ \langle /, A_{ij}, -, \mathbb{R} \rangle_{ij}^{mn}, \mathcal{Y} \right\} \quad \text{(Matrix Const)}$$

$$\Gamma; \Sigma, x : o^{1\times 1}, \Sigma' \vdash x : \langle /, (0, e_{\iota(k,1,1)}), \nearrow, o \rangle \quad \text{(Scalar Var)}$$

$$\Gamma; \Sigma, x : o^{m\times n}, \Sigma' \vdash x : \{\langle /, (0, e_{\iota(k,i,j)}), \nearrow, o)\rangle_{ij}^{mn}, \perp\} \quad \text{(Matrix Var)}$$

$$\Gamma; \Sigma, x : \mathcal{S}^m, \Sigma' \vdash x : \{\langle /, (0, e_{\iota(k,i,j)}), \nearrow, \mathbb{R} \rangle_{ij}^{mm}, \ddagger\} \quad \text{(Symmetric Var)}$$

$$\frac{\hat{\Gamma}, f : \{t\}_i^m \to t, \hat{\Gamma}' \vdash f : \{t\}_i^m \to t \quad \{\Gamma; \Sigma \vdash e : t\}_i^m}{\Gamma; \Sigma \vdash f\{e\}_i^m : t} \quad \text{(App)}$$

where we abbreviate $\langle c, p, m, \mathbb{R} \rangle$ for $\{\langle c, p, m, \mathbb{R} \rangle_{ij}^{11}, \ddagger\}$. In addition, we specify that $\mathcal{Y} = \ddagger$ when A is symmetric and $\perp$ when it is not. Also, we define $e_{\iota(k,i,j)}$ to be a vector of length $\sum_{k=1}^{n} r_k c_k$ where $r_k \times c_k$ denotes the size of each matrix in the context Σ. This vector consists of all zeros save a single 1 in the $\iota(k, i, j)$th position where we define

$$\iota(k, i, j) = i + (j - 1)r_k + \sum_{p=1}^{k-1} r_p c_p$$

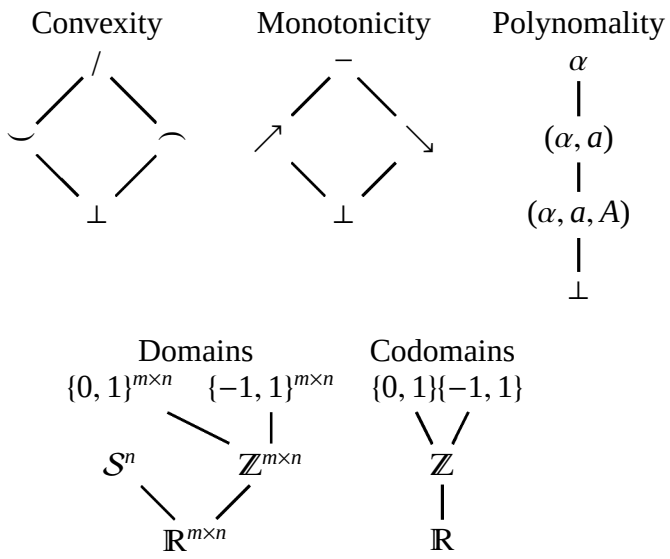
We refer to this vector as the $\iota(k, i, j)$th canonical vector. As a final note, we see that the typing rules for constants and indices are ambiguous. This does not cause a problem since the correct typing rule can always be determined from the context.

The typing rules of a mathematical program give insight into its structure. Initially, each mathematical program depends solely on a set of predefined functions whose type bindings are found within Γ. This context contains functions such as addition and subindexing. The objective function and constraints are type checked in the rule (Prog). Here, we require that objective be a model function and the constraints be well-typed. The typing rules for constants and variable lookups mirror those found within lambda-calculus. However, variable lookups appear slightly strange. In this case, each variable can be treated as a function of itself with the appropriate properties. For example, the above rules would prove the following

$$\Gamma; x : \mathbb{R}^{2\times 2}, y : \mathbb{R}^{1\times 1} \vdash y : \left\{ \left\langle /, \left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right), \nearrow, \mathbb{R} \right\rangle, \ddagger \right\}$$

In other words, y is a function that is linear in x and y.

The following Hasse diagrams detail the subtype relationships for convexity, monotonicity, polynomality, and sets. In the diagram, elements on the top of the graph are subtypes of those connected below.



Two composite types, $\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$, are considered subtypes of each other when their sizes, m and n, are the same and each component is a subtype of the other.

4.3 Builtin Functions and Their Type

In the above typing rules, we utilize a context Γ which contains the type of builtin functions. Notice, there was no introduction rule for the context Γ, so all functions within the context must be predefined. The following two sections detail these functions and their types. In the first section, we address functions that accept purely scalar arguments. Then, we generalize these ideas into functions that accept matrix arguments.

4.3.1 Scalar Functions

By scalar function, we mean a function where all model functions input into the function must have a scalar codomain. We type these functions as follows

Partial Order Defined by the Nonnegative Orthant

The partial order defined by the nonnegative orthant has type

$$\succeq_{\mathbb{R}_+} : \{\langle c, p, m, o\rangle\}_k^2 \to \diamond$$

where there are no restrictions on the properties of the function.

Equality

Equality has the following type

$$=: \{\langle c, p, m, o\rangle\}_k^2 \to \diamond$$

Like the partial order defined by the nonnegative orthant, we do not restrict the kind of functions used in equality constraints.

Addition

Addition between scalars has the following type,

$$+ : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle$$

where

$$
c = \begin{cases} / & c_k = / \\ \smile & c_k = \smile \\ \frown & c_k = \frown \\ \bot & otherwise \end{cases}
$$

$$
p = \begin{cases} \alpha_1 + \alpha_2 & p_k = \alpha_k \\ (\alpha_1 + \alpha_2, a_1 + a_2) & p_k = (\alpha, a)_k \\ (\alpha_1 + \alpha_2, a_1 + a_2, A_1 + A_2) & p_k = (\alpha, a, A)_k \\ \bot & otherwise \end{cases}
$$

$$
m = \begin{cases} - & m_k = - \\ \nearrow & m_k = \nearrow \\ \searrow & m_k = \searrow \\ \bot & otherwise \end{cases}
$$

$$
o = \begin{cases} \mathbb{Z} & o_k = \mathbb{Z} \\ \mathbb{R} & o_k = \mathbb{R} \end{cases}
$$

Negation

The negation of a scalar has the following type

$$
- : \{\langle c, p, m, o\rangle\}_k^1 \to \langle c, p, m, o\rangle
$$

where

$$
c = \begin{cases} / & c_1 = / \\ \smile & c_1 = \frown \\ \frown & c_1 = \smile \\ \bot & otherwise \end{cases}
$$

$$
p = \begin{cases} -\alpha_1 & p_1 = \alpha_1 \\ (-\alpha_1, -a_1) & p_1 = (\alpha, a)_1 \\ (-\alpha_1, -a_1, -A_1) & p_1 = (\alpha, a, A)_1 \\ \bot & otherwise \end{cases}
$$

$$
m = \begin{cases} - & m_1 = - \\ \nearrow & m_1 = \searrow \\ \searrow & m_1 = \nearrow \\ \bot & otherwise \end{cases}
$$

$$
o = \begin{cases} \{-1, 1\} & o_1 = \{-1, 1\} \\ \mathbb{Z} & o_1 = \mathbb{Z} \\ \mathbb{R} & o_1 = \mathbb{R} \end{cases}
$$

Subtraction

The subtraction of one scalar from another has the following type

$$
- : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle
$$

The resulting type is identical to the type of the composite function $f + (-g)$ where f and g are the two the arguments passed into negation.

Multiplication

Multiplication between two scalars has the following type

$$* : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle$$

where

$$c = \begin{cases} / & c_1 = /, p_2 = \alpha_2 \\ \smile & \begin{cases} c_1 = \smile, p_2 = \alpha_2 \geq 0 \\ c_1 = \frown, p_2 = \alpha_2 \leq 0 \\ p = (\alpha, a, A), A \geq 0 \end{cases} \\ \frown & \begin{cases} c_1 = \frown, p_2 = \alpha_2 \geq 0 \\ c_1 = \smile, p_2 = \alpha_2 \leq 0 \\ p = (\alpha, a, A), A \preceq 0 \end{cases} \\ \perp & \textit{otherwise} \end{cases}$$

$$p = \begin{cases} \alpha_1\alpha_2 & p_1 = \alpha_1, p_2 = \alpha_2 \\ (\alpha_1\alpha_2, a_1\alpha_2) & p_1 = (\alpha, a)_1, p_2 = \alpha_2 \\ (\alpha_1\alpha_2, a_1\alpha_2, A_1\alpha_2) & p_1 = (\alpha, a, A)_1, p_2 = \alpha_2 \\ (\alpha_1\alpha_2, a_1\alpha_2 + \alpha_1 a_2, (a_1 a_2^T + a_2 a_1^T)/2) & p_1 = (\alpha, a)_1, p_2 = (\alpha, a)_2 \\ \perp & \textit{otherwise} \end{cases}$$

$$
m = \begin{cases}
- & m_k = - \\
\nearrow & \begin{cases} m_1 = \nearrow, p_2 = \alpha_2 \geq 0 \\ m_1 = \searrow, p_2 = \alpha_2 \leq 0 \end{cases} \\
\searrow & \begin{cases} m_1 = \searrow, p_2 = \alpha_2 \geq 0 \\ m_1 = \nearrow, p_2 = \alpha_2 \leq 0 \end{cases} \\
\bot & \textit{otherwise}
\end{cases}
$$

$$
o = \begin{cases}
\{0, 1\} & o_k = \{0, 1\} \\
\{-1, 1\} & o_k = \{-1, 1\} \\
\mathbb{Z} & o_k = \mathbb{Z} \\
\mathbb{R} & o_k = \mathbb{R}
\end{cases}
$$

Scalar multiplication is also commutative, i.e. $x * y = y * x$. Thus, we can type the corresponding function in the same manner as above, but with the arguments reversed.

Division

The division of one scalar by another has the following type

$$/ : \{\langle c, p, m, o\rangle\}_k^2 \rightarrow \langle c, p, m, o\rangle$$

where

$$
c = \begin{cases}
/ & c_1 = /, p_2 = \alpha_2 \\
\smile & \begin{cases} c_1 = \smile, p_2 = \alpha_2 \geq 0 \\ c_1 = \frown, p_2 = \alpha_2 \leq 0 \\ p = (\alpha, a, A), A \geq 0 \end{cases} \\
\frown & \begin{cases} c_1 = \frown, p_2 = \alpha_2 \geq 0 \\ c_1 = \smile, p_2 = \alpha_2 \leq 0 \\ p = (\alpha, a, A), A \leq 0 \end{cases} \\
\bot & \textit{otherwise}
\end{cases}
$$

$$
p = \begin{cases}
\alpha_1/\alpha_2 & p_1 = \alpha_1, p_2 = \alpha_2 \\
(\alpha_1/\alpha_2, a_1/\alpha_2) & p_1 = (\alpha, a)_1, p_2 = \alpha_2 \\
(\alpha_1/\alpha_2, a_1/\alpha_2, A_1/\alpha_2) & p_1 = (\alpha, a, A)_1, p_2 = \alpha_2 \\
\bot & \textit{otherwise}
\end{cases}
$$

$$
m = \begin{cases}
- & m_1 = -, p_2 = \alpha_2 \\
\nearrow & \begin{cases} m_1 = \nearrow, p_2 = \alpha_2 \geq 0 \\ m_1 = \searrow, p_2 = \alpha_2 \leq 0 \end{cases} \\
\searrow & \begin{cases} m_1 = \searrow, p_2 = \alpha_2 \geq 0 \\ m_1 = \nearrow, p_2 = \alpha_2 \leq 0 \end{cases} \\
\bot & \textit{otherwise}
\end{cases}
$$

$$
o = \begin{cases}
\{-1, 1\} & o_k = \{-1, 1\} \\
\mathbb{Z} & o_1 = \mathbb{Z}, o_2 = \{-1, 1\} \\
\mathbb{R} & o_k = \mathbb{R}
\end{cases}
$$

Absolute Value

The absolute value of a scalar has the following type

$$|\cdot| : \{\langle c, p, m, o\rangle\}_k^1 \to \langle c, p, m, o\rangle$$

where

$$c = \begin{cases} / & p_1 = \alpha_1 \\ \bot & otherwise \end{cases}$$

$$p = \begin{cases} |\alpha_1| & p_1 = \alpha_1 \\ \bot & otherwise \end{cases}$$

$$m = \begin{cases} - & m_1 = - \\ \bot & otherwise \end{cases}$$

$$o = \begin{cases} \{0, 1\} & o_1 = \{0, 1\} \\ \{-1, 1\} & o_1 = \{-1, 1\} \\ \mathbb{Z} & o_1 = \mathbb{Z} \\ \mathbb{R} & o_1 = \mathbb{R} \end{cases}$$

Binary Maximum

We type the maximum between two scalars as follows

$$max : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle$$

where

$$
c = \begin{cases} / & p_k = \alpha_k \\ \smile & c_k = \smile \\ \perp & otherwise \end{cases}
$$

$$
p = \begin{cases} \max(\alpha_1, \alpha_2) & p_k = \alpha_k \\ \perp & otherwise \end{cases}
$$

$$
m = \begin{cases} - & m_k = - \\ \nearrow & m_k = \nearrow \\ \searrow & m_k = \searrow \\ \perp & otherwise \end{cases}
$$

$$
o = \begin{cases} \{0, 1\} & o_k = \{0, 1\} \\ \{-1, 1\} & o_k = \{-1, 1\} \\ \mathbb{Z} & o_k = \mathbb{Z} \\ \mathbb{R} & otherwise \end{cases}
$$

Binary Minimum

The binary minimum between two scalars types as

$$
min : \{\langle c, p, m, o\rangle\}_k^2 \rightarrow \langle c, p, m, o\rangle
$$

where

$$
c = \begin{cases} / & p_k = \alpha_k \\ \frown & c_k = \frown \\ \bot & \textit{otherwise} \end{cases}
$$

$$
p = \begin{cases} \min(\alpha_1, \alpha_2) & p_k = \alpha_k \\ \bot & \textit{otherwise} \end{cases}
$$

$$
m = \begin{cases} - & m_k = - \\ \nearrow & m_k = \nearrow \\ \searrow & m_k = \searrow \\ \bot & \textit{otherwise} \end{cases}
$$

$$
o = \begin{cases} \{0, 1\} & o_k = \{0, 1\} \\ \{-1, 1\} & o_k = \{-1, 1\} \\ \mathbb{Z} & o_k = \mathbb{Z} \\ \mathbb{R} & \textit{otherwise} \end{cases}
$$

Exponentiation

The exponentiation of one scalar by another has type

$$
\dot{\cdot} : \{\langle c, p, m, o \rangle\}_k^2 \to \langle c, p, m, o \rangle
$$

where

$$
c = \begin{cases}
/ & \begin{cases} p_k = \alpha_k \\ p_2 = 0 \\ c_1 = /, p_2 = 1 \end{cases} \\
\smile & \begin{cases} c_1 = /, p_2 = \alpha_2, \alpha_2 \bmod 2 = 0, \alpha_2 > 1 \\ c_1 = \smile, p_2 = 1 \\ c_1 = \smile, p_1 = (0, 0, A_1), p_2 = 1/2 \end{cases} \\
\frown & c_1 = \frown, p_2 = 1 \\
\bot & \textit{otherwise}
\end{cases}
$$

$$
p = \begin{cases}
\alpha_1^{\alpha_2} & p_1 = \alpha_1, p_2 = \alpha_2 \\
1 & p_2 = 0 \\
p_1 & p_2 = 1 \\
(\alpha_1^2, 2\alpha_1 a_1, a_1 a_1^T) & p_1 = (\alpha_1, a_1), p_2 = 2 \\
\bot & \textit{otherwise}
\end{cases}
$$

$$
m = \begin{cases}
- & m_k = - \\
\nearrow & m_1 = \nearrow, p_2 = \alpha_2, \alpha_2 \bmod 2 = 1, \alpha_2 > 0 \\
\searrow & m_k = \searrow, p_2 = \alpha_2, \alpha_2 \bmod 2 = 1, \alpha_2 > 0 \\
\bot & \textit{otherwise}
\end{cases}
$$

$$
o = \begin{cases}
\{0, 1\} & o_k = \{0, 1\} \\
\{-1, 1\} & o_k = \{-1, 1\} \\
\mathbb{Z} & o_k = \mathbb{Z}, p_2 = \alpha_2, \alpha_2 \geq 0 \\
\mathbb{R} & \textit{otherwise}
\end{cases}
$$

4.3.2 Matrix Functions

By matrix function, we mean a function that can accept any arbitrary model function as an input regardless if that function has a scalar or matrix codomain. These functions are typed as follows.

Partial Order Defined by the Nonnegative Orthant

The partial order defined by the nonnegative orthant has type

$$\succeq_{\mathbb{R}_+} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond$$

where we require that $m_1 = m_2$ and $n_1 = n_2$.

Partial Order Defined by the Second Order Cone

The partial order defined by the second order cone has type

$$\succeq_{Q} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond$$

where we require that $m_1 = m_2$, $n_1 = n_2$, and that either $m_1 = 1$ or $n_1 = 1$.

Partial Order Defined by the Cone of Positive Semidefinite Matrices

The partial order defined by the cone of positive semidefinite matrices has type

$$\succeq_{\mathcal{S}_+} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond$$

where we require that $m_1 = m_2 = n_1 = n_2$ and $y_1 = y_2 = \ddagger$.

Equality

Equality as the following type

$$=: \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond$$

where we require that $m_1 = m_2$ and $n_1 = n_2$.

Addition

Addition between matrices has the following type,

$$+ : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$$

where $m = m_1 = m_2$ and $n = n_1 = n_2$. Let f and g be the two arguments given to addition. Then, we type each (i, j) element of the codomain as $f_{ij} + g_{ij}$. Additionally, when $y_1 = y_2 = \ddagger, y = \ddagger$.

Negation

The negation of a matrix has the following type

$$- : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$$

where $m = m_1$ and $n = n_1$. When f is the argument passed into negation, we type each (i, j) pointwise function as $-f_{ij}$. If $y_1 = \ddagger$, we also specify that $y = \ddagger$.

Subtraction

The subtraction of one matrix from another has the following type

$$- : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$$

where $m = m_1 = m_2$ and $n = n_1 = n_2$. Let the two arguments given to subtraction be f and g. Then, we type each (i, j) element of the codomain as $f_{ij} - g_{ij}$. Additionally, when $y_1 = y_2 = \ddagger, y = \ddagger$.

Multiplication by a Scalar

The multiplication of a matrix by a scalar has the following type

$$* : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$$

where $m = m_1$, $n = n_1$, and we require that $m_2 = n_2 = 1$. Let f and g be the two arguments. Then, we type the (i, j)th pointwise function as $f_{ij} * g$. When $y_1 = \ddagger$, we also specify that $y = \ddagger$. Finally, as multiplication by a scalar is commutative, we type the corresponding function in the same manner as above with the arguments reversed.

Division by a Scalar

The division of a matrix by a scalar has the following type

$$/ : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$$

where $m = m_1$, $n = n_1$, and we require that $m_2 = n_2 = 1$. When f and g are our two arguments, we type the (i, j)th pointwise function as f_{ij}/g. As with multiplication, when $y_1 = \ddagger$, we specify that $y = \ddagger$.

Submatrixing

The submatrixing of a matrix has the following type

$$\cdot_{\cdot:\cdot,\cdot:\cdot} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1, \{\eta\}_k^4 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$$

where we require that $m_1 \geq \eta_3 \geq \eta_1$ and $n_1 \geq \eta_4 \geq \eta_2$. When these conditions are satisfied, we specify that $m = \eta_2 - \eta_1 + 1$ and $n = \eta_4 - \eta_3 + 1$. Let the first argument be f. Then, the type of the (i, j)th element in the result is that of $f_{(i+\eta_1)(j+\eta_3)}$. Further, we state that $y = \ddagger$ when $y_1 = \ddagger$ and $\eta_1 = \eta_3, \eta_2 = \eta_4$.

Subindexing

The subindexing of a matrix has the following type

$$\cdot_{,,\cdot} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1, \{\eta\}_k^2 \to \langle c, p, m, o\rangle$$

where the type is nearly identical to the submatrixing function with the matrix indices $(\eta_1 : \eta_1, \eta_2 : \eta_2)$. The only difference is that since the result is a scalar, it must be symmetric.

Matrix Multiplication

The multiplication of two matrices has the following type

$$* : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$$

where we stipulate that $m = m_1$, $n = n_2$, and $n_1 = m_2$. Before we define the possible type combinations, let us recall the definition of matrix multiplication

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Now, recall that we are considering matrix valued functions. Thus, we have that

$$h_{ij}(x) = \sum_{k=1}^{n} f_{ik}(x) g_{kj}(x)$$

Each function f_{ik} and g_{kj} is scalar valued. Thus, we can determine the type of h_{ij} by combining the typing rules for addition, scalar multiplication, and subindexing. Finally, since we can not guarantee symmetry, $y = \perp$.

Absolute Value

The pointwise application of absolute value has the following type

$$|\cdot| : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$$

where $m = m_1$ and $n = n_2$. Let f be the argument given. Then, we can type each pointwise function of the absolute value function by $|f_{ij}|$. Moreover, $y = y_1$.

Elementwise Maximum

The elementwise maximum of a matrix has the following type

$$max : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle$$

where each type combination is determined by recursively applying the binary maximum between two scalars in the following manner. Let f be the argument passed into *max*. Then, we determine the type of max using the following relation

$$max(f)$$
$$= max(f_{11}, max(f_{21}, \ldots, max(f_{m1}, max(f_{12}, \ldots, max(f_{m-1,n}, f_{mn})\ldots)\ldots)\ldots)\ldots)$$

When $m_1 = n_1 = 1$, the type of *max* is that of its argument

Elementwise Minimum

The elementwise minimum of a matrix has the following type

$$min : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \rightarrow \langle c, p, m, o\rangle$$

where the type combinations are defined similarly to elementwise maximum. When f is the argument passed into *min*, we determine the type of *min* with the following

$$\begin{aligned} &min(f) \\ &\quad = min(f_{11}, min(f_{21}, \ldots, min(f_{m1}, min(f_{12}, \ldots, min(f_{m-1,n}, f_{mn}) \ldots) \ldots) \ldots) \ldots) \end{aligned}$$

When $m_1 = n_1 = 1$, the type of *min* is the same as its argument.

Summation

The sum of all elements in a matrix has type

$$sum : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \rightarrow \langle c, p, m, o\rangle$$

where each type combination is defined using a method similar to elementwise min and max. Let the argument handed to sum be denoted by f. Then, we define the type of sum as

$$sum(f) = f_{11} + f_{21} + \cdots + f_{m1} + f_{12} + \cdots + f_{m-1,n} + f_{mn}$$

In the case that $m_1 = n_1 = 1$, the type of *sum* coincides with f.

P-Norms

All p-norms with $p \geq 1$ have the following type

$$\|\cdot\|_p : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle$$

where

$$c = \begin{cases} / & p = \alpha \\ \smile & c_{1ij} = / \text{ for all } i, j \\ \perp & otherwise \end{cases}$$

$$p = \begin{cases} \|A\|_p & p_1 = \alpha_{1ij} \text{ for all } i, j \text{ and } A_{ij} = \alpha_{1ij} \\ \perp & otherwise \end{cases}$$

$$m = \begin{cases} - & p = \alpha \\ \perp & otherwise \end{cases}$$

$$o = \mathbb{R}$$

Transpose

The transpose of a matrix has the following type

$$\cdot^T : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$$

where $m = n_1$ and $n = m_1$. Then, the type of the $(i, j)th$ element in the result is equal to the type of the $(j, i)th$ element of the argument. Further, as this operation is symmetry preserving, $y = y_1$.

Trace

The trace of a matrix has type

$$tr : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \rightarrow \langle c, p, m, o\rangle$$

where we require that $m_1 = n_1$. Recall, the trace of a matrix is defined as

$$tr(X) = \sum_{i=1}^{n} X_{ii}$$

Thus, we determine the type of trace by combining the typing rules for subindexing and addition.

Horizontal Concatenation

The horizontal concatenation of two matrices has type

$$[\cdot\,\cdot] : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \rightarrow \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$$

where we stipulate that $m_1 = m_2$. When we satisfy this condition, we specify that $m = m_1$ and $n = n_1 + n_2$. When we denote our two arguments by f and g respectively, we type the

(i, j)th pointwise function as f_{ij} for $j \leq n_1$ and $g_{i,j-n_1}$, otherwise. We always specify that $y = \perp$.

Vertical Concatenation

Similar to horizontal concatenation, the vertical concatenation of two matrices has type

$$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} : \left\{ \{\langle c, p, m, o \rangle_{ij}^{mn}, y\} \right\}_k^2 \to \{\langle c, p, m, o \rangle_{ij}^{mn}, y\}$$

where we require that $n_1 = n_2$. Should we satisfy this condition, we define that $m = m_1 + m_2$ and $n = n_1$. When we denote our two arguments by f and g respectively, we type the (i, j)th pointwise function as f_{ij} for $i \leq m_1$ and $g_{i-m_1,j}$, otherwise. We always note that $y = \perp$.

Symmetric Concatenation

Symmetric concatenation allows us to combine matrices while ensuring symmetry. This function has type

$$\begin{bmatrix} \cdot & \cdot \\ & \\ & \cdot \end{bmatrix} = \left\{ \{\langle c, p, m, o \rangle_{ij}^{mn}, y\} \right\}_k^3 \to \{\langle c, p, m, o \rangle_{ij}^{mn}, y\}$$

where we must guarantee that $m_1 = m_2$, $n_2 = n_3$, and $y_1 = y_3 = \ddagger$. Once we meet these conditions, we specify that $m = m_1 + m_3$ and $n = n_1 + n_2$. Let the three arguments be denoted by f, g, and h and denote the result by r. We denote the type of the (i, j)th element

of the result as

$$r_{ij} = \begin{cases} f_{ij} & 1 \leq i \leq m_1, 1 \leq j \leq n_1 \\ g_{i,j-n_1} & 1 \leq i \leq m_1, n_1 + 1 \leq j \leq n_1 + n_2 \\ g_{j,i-m_1} & m_1 + 1 \leq i \leq m_1 + m_3, 1 \leq j \leq n_1 \\ h_{i-m_1,j-n_1} & m_1 + 1 \leq i \leq m_1 + m_3, n_1 + 1 \leq j \leq n_1 + n_2 \end{cases}$$

Finally, we specify that $y = \ddagger$.

Maximum Eigenvalue

The maximum eigenvalue function has the following type

$$\lambda_{max} : \left\{ \{ \langle c, p, m, o \rangle_{ij}^{mn}, y \} \right\}_k^1 \to \langle c, p, m, o \rangle$$

where

$$c = \begin{cases} / & p = \alpha \\ \smile & c_{1ij} = / \text{ for all } i, j \\ \perp & \textit{otherwise} \end{cases}$$

$$p = \begin{cases} \lambda_{max}(A) & p_1 = \alpha_{1ij} \text{ for all } i, j \text{ and } A_{ij} = \alpha_{1ij} \\ \perp & \textit{otherwise} \end{cases}$$

$$m = \begin{cases} - & p = \alpha \\ \perp & \textit{otherwise} \end{cases}$$

$$o = \mathbb{R}$$

We require that $m_1 = n_1$ and $y_1 = \ddagger$.

Minimum Eigenvalue

We type the minimum eigenvalue function in a manner similar to the maximum

$$\lambda_{min} : \left\{ \{ \langle c, p, m, o \rangle_{ij}^{mn}, y \} \right\}_k^1 \to \langle c, p, m, o \rangle$$

where

$$c = \begin{cases} / & p = \alpha \\ \frown & c_{1ij} = / \text{ for all } i, j \\ \perp & otherwise \end{cases}$$

$$p = \begin{cases} \lambda_{min}(A) & p_1 = \alpha_{1ij} \text{ for all } i, j \text{ and } A_{ij} = \alpha_{1ij} \\ \perp & otherwise \end{cases}$$

$$m = \begin{cases} - & p = \alpha \\ \perp & otherwise \end{cases}$$

$$o = \mathbb{R}$$

Additionally, we require that $m_1 = n_1$ and $y_1 = \ddagger$.

Chapter 5

Semantics

Up until this point, we have defined a series of rules that specify the form a mathematical program. This form by itself has no inherent meaning. In other words, a program is simply text. We have defined rules to insure this text is well-formed. The semantics of a language define a series of functions that map text to mathematical meaning. Once we define our semantics, we can prove that our above typing rules are correct. For example, we can prove that when an expression has type $\smile$, then this expression represents a convex function.

5.1 Preliminaries

In the following presentation we introduce key concepts used within the formal semantics. This includes a brief description of denotational semantics. It also includes the definitions and basic results about convexity, polynomality, and monotonicity. During their discussion, we pay special attention to how we apply these concepts to functions of matrices. Next, we will describe the relations that define our constraints. This includes how a pointed convex cone describes a partial order which is central to cone programming.

5.1.1 Denotational Semantics

The core idea behind denotational semantics is the definition of a function that maps grammar to mathematical meaning. There are two contrasting approaches that we may choose. In the Curry-style approach, we assign a meaning to every term regardless of whether it is well-typed. After we define the meaning of each term, we discard ill-typed terms. In the Church-style approach, we reverse the order of these steps. Thus, we type check each term and give meaning only to the remaining well-typed terms. In the following presentation, we employ the Church-style of semantics and define meaning only on well-typed terms.

Continuing our calculator example, we define two sets. First, let $\mathsf{T} = \{\mathbb{Q}, \mathbb{B}\}$ be the set of that contains two elements, the set of rationals and the set of booleans. Second, we define the set of all possible binary functions that operate on rationals and booleans as $\mathsf{F} = \{[\mathsf{t}_1 \times \mathsf{t}_2 \to \mathsf{t}_3]\}_{\mathsf{t}_i \in \mathsf{T}}$. This allows us to define the semantics of our calculator as

Types $\boxed{[\![\cdot]\!] : T \to \mathsf{T}}$

$[\![\mathbb{Q}]\!] = \mathbb{Q}$

$[\![\mathbb{B}]\!] = \mathbb{B}$

Function Types $\boxed{[\![\cdot]\!] : (\{T, T\} \to T) \to \mathsf{F}}$

$[\![\{t_1, t_2\} \to t]\!] = [[\![t_1]\!] \times [\![t_2]\!] \to [\![t]\!]]$

Expressions $\boxed{[\![\cdot]\!] : G \times E \times T \rightharpoonup \bigcup_{\mathsf{t} \in \mathsf{T}} \mathsf{t}}$

$[\![\Gamma \vdash c : t]\!] = c$

$[\![\Gamma \vdash f\{e_1, e_2\} : t]\!] = [\![\Gamma \vdash f : \{t_1, t_2\} \to t]\!] \{[\![\Gamma \vdash e_1 : t_1]\!], [\![\Gamma \vdash e_2 : t_2]\!]\}$

Functions

$$\boxed{[\![\cdot]\!] : G \times E \times (\{T, T\} \to T) \rightharpoonup \bigcup_{\mathsf{f} \in \mathsf{F}} \mathsf{f}}$$

$$[\![\Gamma \vdash + : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}]\!] = x, y \mapsto x + y$$

$$[\![\Gamma \vdash - : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}]\!] = x, y \mapsto x - y$$

$$[\![\Gamma \vdash * : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}]\!] = x, y \mapsto xy$$

$$[\![\Gamma \vdash / : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}]\!] = x, y \mapsto x/y$$

$$[\![\Gamma \vdash \mathit{and} : \{\mathbb{B}, \mathbb{B}\} \to \mathbb{B}]\!] = x, y \mapsto x \text{ and } y$$

$$[\![\Gamma \vdash \mathit{or} : \{\mathbb{B}, \mathbb{B}\} \to \mathbb{B}]\!] = x, y \mapsto x \text{ or } y$$

As an example, we define the meaning of $1 + 2 + 3$ as

$$\begin{aligned}
[\![\Gamma \vdash 1 + 2 + 3 : \mathbb{Q}]\!] &= [\![\Gamma \vdash + : \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}]\!] \{[\![\Gamma \vdash 1 + 2 : \mathbb{Q}]\!], [\![\Gamma \vdash 3 : \mathbb{Q}]\!]\} \\
&= (x, y \mapsto x + y) ([\![\Gamma \vdash + : \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}]\!] \{[\![\Gamma \vdash 1 : \mathbb{Q}]\!], [\![\Gamma \vdash 2 : \mathbb{Q}]\!]\}, 3) \\
&= (x, y \mapsto x + y)((x, y \mapsto x + y)(1, 2), 3) \\
&= (x, y \mapsto x + y)(1 + 2, 3) \\
&= (x, y \mapsto x + y)(3, 3) \\
&= 3 + 3 \\
&= 6
\end{aligned}$$

Thus, we have mapped the meaning of a textual program, 1+2+3, to its actual mathematical meaning 6.

5.1.2 Soundness and Completeness

A technical result called soundness gives us confidence that we correctly designed a language. Recall, using denotational semantics, the meaning of every type is a set and the meaning of every well-typed expression is a mathematical object. Soundness states that the meaning of every well-typed expression lies within the meaning of its type. In other words, it states that $[\![\Gamma \vdash e : t]\!] \in [\![t]\!]$. Once we prove this result, we know that if our typing rules prove that an expression has type, say, $\mathbb{Q}$, then the meaning of this expression must be a rational number.

We prove the soundness of our calculator with the following argument. Assume that c represents a rational constant. Then, our typing rules tell us that $\Gamma \vdash c : \mathbb{Q}$. Thus, we must show that $[\![\Gamma \vdash c : \mathbb{Q}]\!] \in [\![\mathbb{Q}]\!]$. By definition, $[\![\Gamma \vdash c : \mathbb{Q}]\!] = c$ and $[\![\mathbb{Q}]\!] = \mathbb{Q}$. Since we assumed that $c \in \mathbb{Q}$, our definition is sound. Similarly, assume that c represents a boolean constant. Our typing rules stipulate that $\Gamma \vdash c : \mathbb{B}$. Therefore, we must show that $[\![\Gamma \vdash c : \mathbb{B}]\!] \in [\![\mathbb{B}]\!]$. Since we defined that $[\![\Gamma \vdash c : \mathbb{B}]\!] = c \in \mathbb{B} = [\![\mathbb{B}]\!]$, our definition is sound. Finally, we must consider function applications. Recall, we define that $[\![\Gamma \vdash f\{e_1, e_2\} : t]\!] = [\![\Gamma \vdash f : \{t_1, t_2\} \to t]\!]\{[\![\Gamma \vdash e_1 : t_1]\!], [\![\Gamma \vdash e_2 : t_2]\!]\}$. By induction, we know that $[\![\Gamma \vdash e_1 : t_1]\!] \in [\![t_1]\!]$ and $[\![\Gamma \vdash e_2 : t_2]\!] \in [\![t_2]\!]$. Therefore, we must show that the definition of each function definition within Γ is sound.

We begin with addition where we define that $[\![\Gamma \vdash + : \{\mathbb{Q}, \mathbb{Q}\} \to \mathbb{Q}]\!] = x, y \mapsto x + y$. Addition maps two rationals to a rational. By induction, we know both arguments to + must be rational. Therefore, the result of application must be rational and the meaning is

sound. This same argument applies to $-$, $*$, and $/$. Thus, let us consider the logical and between two booleans. We define that $[\![\Gamma \vdash \mathit{and} : \{\mathbb{B}, \mathbb{B}\} \to \mathbb{B}]\!] = x, y \mapsto x$ and y. Logical and maps two booleans to another boolean. By induction, we know both arguments to *and* are boolean. Hence, the result of application must be boolean and the meaning is sound. This same argument applies to the function *or*.

We have shown that the meaning of all expressions in the calculator is sound. Therefore, our definition of the calculator language is sound.

A related, but less important, result is completeness. Completeness considers whether a language can represent an arbitrary mathematical object in its domain. In other words, it considers whether for any $c \in [\![t]\!]$ there exists e such that $[\![\Gamma \vdash e : t]\!] = c$. This result gives us an idea of the expressiveness of our language.

The definition of our calculator is trivially complete. Given any $c \in \mathbb{Q}$, we define that $[\![\Gamma \vdash c : \mathbb{Q}]\!] = c$. Similarly, for any $c \in \mathbb{B}$, we define that $[\![\Gamma \vdash c : \mathbb{B}]\!] = c$. Therefore, the language is complete. Unfortunately, this by itself does not give us much information about the expressiveness of our language. We learn far more information about the expressiveness by considering the completeness of our functions. Notice that $[[\![t_1]\!] \times [\![t_2]\!] \to [\![t]\!]]$ represents a set that contains an uncountably infinite number of functions. The calculator may only express six functions in this set: $+$, $-$, $*$, $/$, *and*, and *or*. Therefore, the true expressiveness of our calculator is restricted by the number of functions that we define.

5.1.3 Convexity

Intuitively, we view convex objects as items that are bowl shaped. The formal definition mirrors this idea closely. A set S is convex when for any $x, y \in S$ and $\lambda \in [0, 1]$ we have that

$$\lambda x + (1 - \lambda)y \in S$$

A function $f : S \to \mathbb{R}$ is convex when

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y)$$

Similarly, a function is concave when we reverse the above inequality. A function is affine when it is both convex and concave. Notice that all functions fall into four categories: convex, concave, affine, or neither.

We must make one special note. Recall that the domain of each function in our language is defined by the decision variables. This includes sets such as integers which are nonconvex. Properly, convexity does not make sense on these sets. However, since solvers typically handle integer constraints specially, we assume that the domain of each variable is real, and hence convex.

5.1.4 Polynomality

A function is a polynomial when it can be represented exactly by some Taylor expansion of a finite order. When a mathematical program is comprised entirely of linear or quadratic

functions, a solver may be able to use specialized algorithms to solve the problem. A function is a quadratic polynomial when it is defined exactly by its second order Taylor series. Thus, a function $f : \mathbb{R}^n \to \mathbb{R}$ is quadratic if we can represent it exactly by q where

$$\begin{aligned} q(x) &= f(0) + \langle \nabla f(0), x \rangle + \frac{1}{2}\langle \nabla^2 f(0)x, x \rangle \\ &= \alpha + \langle a, x \rangle + \langle Ax, x \rangle \end{aligned}$$

where $\alpha \in \mathbb{R}$, $a \in \mathbb{R}^n$, and $A = A^T \in \mathbb{R}^{n \times n}$. Analogously, a function is a linear polynomial when it is defined exactly by its first order Taylor series. In our discussion, we only describe quadratic polynomials in detail since linear and constant polynomials are subsumed by quadratics.

We generalize this concept to matrices in a natural way using tensors. Given a function $f : \mathbb{R}^{m \times n} \to \mathbb{R}$, the second order Taylor approximation is given by

$$\begin{aligned} q(x) &= f(0) + \langle \nabla f(0), x \rangle + \frac{1}{2}\langle \nabla^2 f(0)(x), x \rangle \\ &= \alpha + \langle a, x \rangle + \langle A(x), x \rangle \end{aligned}$$

where $\alpha \in \mathbb{R}$, $a \in \mathbb{R}^{m \times n}$, and $A : \mathbb{R}^{m \times n} \to \mathbb{R}^{m \times n}$ is a linear operator where

$$[A(x)]_{ij} = \langle A_{ij}, x \rangle$$

and $A_{ij} \in \mathbb{R}^{m \times n}$. Thus, we can represent A as a tensor where

$$A_{ijkl} = \frac{1}{2} \frac{\partial f}{\partial x_{ij} \partial x_{kl}}$$

Unfortunately, this isn't quite enough. We would like to recognize bilinear functions as quadratic. For example, the constraint $Xy = c$ defines a system of bilinear, or quadratic constraints. Thus, each quadratic must be defined on all decision variables, not just a single variable. Hence, we are concerned with functions where $f : \mathbb{R}^{r_1 \times c_1 + \cdots + r_n \times c_n} \to \mathbb{R}$. As the domain of f is awkward, we must be careful with how we represent elements in its domain.

There are several ways to visualize elements in $\mathbb{R}^{r_1 \times c_1 + \cdots + r_n \times c_n}$. One possible approach is to flatten the elements into a vector. Alternatively, we can view elements as a sequence of size $r_k \times c_k$ matrices. In the first case, objects are easy to manipulate, but they lose structure. In the second, elements possess a natural structure, but we must define operations such as multiplication and factorization. Our presentation will combine both approaches. Since we need to manipulate the Hessian during certain transformations, we represent the coefficients in the Taylor series as vectors and matrices. In order to preserve the structure of x we will continue to view it as a sequence of matrices, but define the conversion operator *vec* as

$$vec(x) = \bigoplus_{k=1}^{n} vec(x_k)$$

where $vec(x_k)$ represents the normal vectorization of the matrix x_k. As a result, we see that the $(k, i, j)th$ element of x corresponds to the $\iota(k, i, j)th$ element of $vec(x)$.

Using this notation, the second order Taylor approximation is given by

$$q(x) = f(0) + \langle \nabla f(0), vec(x) \rangle + \frac{1}{2} \langle \nabla^2 f(0) vec(x), vec(x) \rangle$$
$$= \alpha + \langle a, vec(x) \rangle + \langle A vec(x), vec(x) \rangle$$

where

$$a_{\iota(k,i,j)} = \frac{\partial f}{\partial x_{kij}}$$
$$A_{\iota(k,i,j)\iota(\bar{k},\bar{i},\bar{j})} = \frac{\partial^2 f}{\partial x_{kij} x_{\bar{k}\bar{i}\bar{j}}}$$

Since quadratic functions possess a constant Hessian, the order of these partial derivatives is inconsequential. Thus, A is symmetric.

For example, consider $f : \mathbb{R}^{1\times 2} \times \mathbb{R}^{2\times 1} \to \mathbb{R}$ where $f(z_1, z_2) = z_1 z_2 + \begin{bmatrix} 1 & 1 \end{bmatrix} z_2$. This function is quadratic in z where

$$f(z) = \left\langle \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, vec(z) \right\rangle + \left\langle \begin{bmatrix} 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 \\ 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \end{bmatrix} vec(z), vec(z) \right\rangle$$

As a final note, similar to convexity, we have trouble defining polynomials on integer domains. Thus, we assume that each variable is real and ignore this requirement.

5.1.5 Monotonicity

Monotonicity describes whether a function is either increasing or decreasing. This property is important when determining if the composition of two convex functions is convex. Formally, a function $f : \mathbb{R} \to \mathbb{R}$ is monotonically increasing when $f(x) \geq f(y)$ for $x \geq y$. Similarly, a function a monotonically decreasing when we reverse the first inequality above. Notice that all functions fall into four categories: increasing, decreasing, neither, or both.

This concept generalizes to the Cartesian product of matrix domains with a suitable partial order. Let V_i be some matrix domain such as $\mathbb{R}^{m\times n}$ or $\mathbb{Z}^{p\times q}$. Given two points $x, y \in \prod_{k=1}^{n} V_k$, $x \succeq_{\mathbb{R}_+} y$ when $x_{kij} \geq y_{kij}$ for each k, i, and j. Thus, we say that a function $f : \prod_{k=1}^{n} V_i \to V$ is monotonically increasing when $f(x) \geq f(y)$ for $x \succeq_{\mathbb{R}_+} y$.

5.1.6 Relations

A binary relation R between two sets A and B is a subset of $A \times B$. For $a \in A$ and $b \in B$, we say that aRb when $(a, b) \in R$. We are only concerned with relations commonly used in optimization. Thus, we restrict our attention to equality, pointwise nonnegativity, and the partial orders defined by the second-order cone and the cone of positive semidefinite matrices.

Since we are operating on matrices, we define equality and nonnegativity pointwise. Thus, for $A, B \in \mathbb{C}^{m\times n}$, $A = B$ when $A_{ij} = B_{ij}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$. Similarly, for $A, B \in \mathbb{R}^{m\times n}$, $A \succeq_{\mathbb{R}_+} B$ when $A_{ij} \geq B_{ij}$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$.

Partial orders defined by pointed convex cones require more care. Recall, a partial order is a binary relation that satisfies four properties

1. reflexivity: $a \succeq a$
2. antisymmetry: if both $a \succeq b$ and $b \succeq a$, then $a = b$
3. transitivity: if both $a \succeq b$ and $b \succeq c$, then $a \succeq c$
4. compatibility with linear operations
   (a) homogeneity: if $a \succeq b$ and $\lambda \in \mathbb{R} \geq 0$, then $\lambda a \succeq \lambda b$
   (b) additivity: if both $a \succeq b$ and $c \succeq d$, then $a + c \succeq b + d$

Notice that this differs from an ordering since there may be some elements that we can not compare. For example, our definition of pointwise nonnegativity defines a partial order. Thus, in $\mathbb{R}^2$ we can see that

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 4 \\ 3 \end{bmatrix} \geq \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

But, we also see that

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \not\geq \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 1 \\ 0 \end{bmatrix} \not\geq \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Let us define a binary relation $\succeq_K$ where

$$a \succeq_K b \iff a - b \succeq_K 0 \iff a - b \in K$$

where K is a pointed convex cone. Recall, all pointed convex cones must satisfy three properties

1. K is nonempty and closed under addition: $a, b \in K \implies a + b \in K$
2. K is a conic set: $a \in K, \lambda \geq 0 \implies \lambda a \in K$
3. K is pointed: $a \in K$ and $-a \in K \implies a = 0$

This leads us to the following lemma

Lemma 1. *The binary relation $\succeq_K$ defines a partial order*

Proof. We must verify the four properties of partial orders

1. reflexivity: Since K is a cone, for any $a \in K$, $\lambda a \in K$ for $\lambda \geq 0$. Let $\lambda = 0$. Thus, we see that $0 \in K \implies a - a \in K \implies a \succeq_K a$.
2. antisymmetry: Let $a \succeq_K b$ and $b \succeq_K a$. This implies that $a - b \in K$ and $b - a = -(a - b) \in K$. But since K is pointed, this implies that $a - b = 0 \implies a = b$.
3. transitivity: Let $a \succeq_K b$ and $b \succeq_K c$. Then we have that $a - b \in K$ and $b - c \in K$. But, since K is nonempty and closed under addition, we must have that $(a - b) + (b - c) = a - c \in K \implies a \succeq_K c$.
4. compatibility with linear operations
   (a) homogeneity: Let $a \succeq_K b$ and $\lambda \geq 0$. Since K is a conic set and $a - b \in K$, we see that $\lambda(a - b) \in K \implies \lambda a \succeq_K \lambda b$

(b) additivity: Let $a \succeq_K b$ and $c \succeq_K d$. Since K is nonempty and closed under addition, $a-b \in K$, and $c-d \in K$, we find that $(a-b)+(c-d) = a+c-b-d \in K \implies a+c-b-d \succeq_K 0 \implies a+c \succeq_K b+d$

□

Three useful pointed convex cones include the nonnegative orthant, the second order cone, and the cone of positive semidefinite matrices. The nonnegative orthant $\mathbb{R}^n_+$ is defined as the set

$$\mathbb{R}^n_+ = \{x \in \mathbb{R}^n : x_i \geq 0\}$$

The second order cone Q^n is the set defined by

$$Q^n = \left\{ x \in \mathbb{R}^n : x_1 \geq \sqrt{\sum_{i=2}^{n} x_i^2} \right\}$$

Finally, the cone of positive semidefinite matrices is the set defined by

$$\mathcal{S}^n_+ = \left\{ X \in \mathbb{R}^{n \times n} : X = X^T, \forall d \in \mathbb{R}^n, d^T X d \geq 0 \right\}$$

We prove that each of these is a pointed convex cone in the following lemma

Lemma 2. *The nonnegative orthant, second order cone, and the cone of positive semidefinite matrices are all pointed convex cones.*

Proof. We begin with the nonnegative orthant. Let $x, y \in \mathbb{R}^n_+$. We must verify three properties. First, let $z = x + y$. We see that $z_i \geq 0$ for each i since $x_i \geq 0$, $y_i \geq 0$, and $z_i = x_i + y_i$.

Thus, $z \in \mathbb{R}^n_+$ and we have shown that $\mathbb{R}^n_+$ is closed under addition. Second, let $\lambda \geq 0$ and $z = \lambda x$. We see that $z_i = \lambda x_i \geq 0$ since both $x_i \geq 0$ and $\lambda \geq 0$. Thus, $z \in \mathbb{R}^n_+$ and we have shown that $\mathbb{R}^n_+$ is a cone. Finally, assume that $-x \in \mathbb{R}^n_+$. Then for each i, $x_i \geq 0$ and $x_i \leq 0$. Thus, $x_i = 0$ and we have that $\mathbb{R}^n_+$ is pointed. Therefore, $\mathbb{R}^n_+$ is a pointed convex cone.

Next, we consider the second order cone. Let $x, y \in \mathcal{Q}^n$ where

$$x = \begin{bmatrix} x_0 \\ \\ \bar{x} \end{bmatrix} \quad y = \begin{bmatrix} y_0 \\ \\ \bar{y} \end{bmatrix}$$

First, let $z = x + y$. Then, we have that

$$z = \begin{bmatrix} x_0 + y_0 \\ \\ \bar{x} + \bar{y} \end{bmatrix} = \begin{bmatrix} z_0 \\ \\ \bar{z} \end{bmatrix}$$

Thus, we must verify that $z_0 \geq ||\bar{z}||_2$. Since $x_0 \geq ||\bar{x}||_2$ and $y_0 \geq ||\bar{y}||_2$ we have that

$$z_0 = x_0 + y_0 \geq ||\bar{x}||_2 + ||\bar{y}||_2 \geq ||\bar{x} + \bar{y}||_2 = ||\bar{z}||_2$$

from the triangle inequality. Thus, $z \in \mathcal{Q}^n$ and we have shown that $\mathcal{Q}^n$ is closed under addition. Second, let $\lambda \geq 0$ and $z = \lambda x$. Since $\lambda \geq 0$ and $x \in \mathcal{Q}^n$,

$$z_0 = \lambda x_0 \geq \lambda ||\bar{x}||_2 = ||\lambda \bar{x}||_2 = ||\bar{z}||_2$$

Thus, $z \in \mathcal{Q}^n$ and we have shown that $\mathcal{Q}^n$ is a cone. Finally, assume that $-x \in \mathcal{Q}^n$. Then, we

have that

$$-x_0 \geq \| - \overline{x}\|_2 = \|\overline{x}\|_2$$

Thus, we have that

$$\|\overline{x}\|_2 \leq x_0 \leq -\|\overline{x}\|_2$$

Since $\|\overline{x}\| \geq 0$, this implies that $x = 0$. Thus, $0 \in \mathcal{Q}^n$ and we have shown that $\mathcal{Q}^n$ is pointed. Therefore, $\mathcal{Q}^n$ is a pointed convex cone.

Finally, we consider the cone of positive semidefinite matrices. Let $X, Y \in \mathcal{S}_+^n$. First, let $Z = X + Y$. Then for any $d \in \mathbb{R}^n$, we have that

$$d^T Z d = d^T (X + Y) d = d^T X d + d^T Y d \geq 0$$

since both $X, Y \in \mathcal{S}_+^n$. Thus, we have shown that $\mathcal{S}_+^n$ is closed under addition. Second, let $\lambda \geq 0$ and $Z = \lambda X$. Then for any $d \in \mathbb{R}^n$ we have that

$$d^T Z d = d^T (\lambda X) d = \lambda d^T X d \geq 0$$

since both $\lambda \geq 0$ and $X \in \mathcal{S}_+^n$. Thus, we have shown that $\mathcal{S}_+^n$ is a cone. Finally, assume that $-X \in S_+^n$. Then for any $d \in \mathbb{R}^n$, we have that $d^T X d \geq 0$ and $d^T X d \leq 0$. Thus, $X = 0$ and we have shown that $\mathcal{S}_+^n$ is pointed. Therefore, S_+^n is a pointed convex cone. □

Since each of these cones defines a partial order, we will use the following convention. When $x \in \mathbb{R}_+^n$, we denote $x \succeq_{\mathbb{R}_+} 0$. Notice that this is a special case of pointwise inequality

defined above. Second, when $x \in \mathcal{Q}^n$, we denote $x \succeq_Q 0$. Finally, when $x \in \mathcal{S}_+^n$, we denote $x \succeq_{\mathcal{S}_+} 0$. In each case, if $-x \geq 0$ then we state that $x \preceq 0$.

Practically, we consider these partial orders for two reasons. First, there exists a great variety of problems that can be modeled using these partial orders. Second, we can efficiently solve problems that contain these inequalities with primal-dual interior point methods.

5.1.7 Matrix Types

One central design difficulty stems from the ability to manipulate and analyze matrix valued functions. Some properties such as convexity and polynomality only make sense with scalar valued functions. Other properties such as symmetry are only useful when considering matrix valued functions.

In order to understand matrix valued functions, let us consider an example. Let $f : \mathbb{R}^{m \times n} \times \mathbb{R}^{n \times 1} \rightarrow \mathbb{R}^{m \times 1}$ where $f(X, y) = Xy$ and let $C \in \mathbb{R}^{m \times 1}$ denote a constant matrix. Then, the function f represents a *system* of quadratic functions and the statement $f(X, y) = C$ delineates a *system* of quadratic equations. This idea generalizes other classes of functions and equations.

Thus, in one sense, we view matrix valued functions as a system of functions. This allows us to consider the convexity, polynomality, and monotonicity of each function in the system. In another sense, we view matrix valued functions as simply a function with a matrix domain. This allows us to consider whether element in the range is symmetric. It also allows us to define constraints based on partial orders such as second-order cone

constraints and semidefinite constraints.

In our definition of types, we specify that model functions have type

$$\left\{\langle c, p, m, o\rangle_{ij}^{mn}, y\right\}$$

This tells us that we have a matrix valued function where

$$f(x) = \begin{bmatrix} f_{11}(x) & f_{12}(x) & \dots & f_{1n}(x) \\ f_{21}(x) & f_{22}(x) & \dots & f_{2n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1}(x) & f_{m2}(x) & \dots & f_{mn}(x) \end{bmatrix}$$

Each individual function f_{ij} is scalar valued and it possesses the properties defined by the tuple $\langle c, p, m, o\rangle_{ij}$. The property y tells us if the resulting matrix is symmetric.

5.2 Formal Definition

The grammar and type-system give structure to a language. In other words, they ensure that terms within the language are well-formed. However, these terms are simply text; they do not have meaning. Denotational semantics assigns these pieces of text a precise, unambiguous mathematical meaning. Once we know the meaning of these terms, we can then prove results about our language.

The discussion of the semantics falls into two parts. In the first part, we will assign a formal meaning to each type of the language. This will define a collection of sets. For

example, $[\![\smile]\!]$ defines the set of all convex functions. In the second part, we will assign a formal meaning to each well-typed term. In both of these cases, $[\![\cdot]\!]$ denotes the semantic function that assigns meaning to terms.

In the following discussion, let the set of all possible codomains for scalar valued functions be defined as

$$\mathsf{O} = \{\mathbb{R}, \mathbb{Z}, \{0, 1\}, \{-1, 1\}\}$$

This allows us to denote the set of all possible domains of a decision variable to be

$$\mathsf{D} = \{\mathsf{o}^{m \times n}, \mathcal{S}^n\}_{m,n \in \mathbb{N}, \mathsf{o} \in \mathsf{O}}$$

Since the domain of each function in an optimization problem is defined by the combined domains of all decision variables, we define the set of all possible function domains as

$$\mathsf{S} = \left\{ \prod_{i=1}^{n} \mathsf{d}_i \right\}_{n \in \mathbb{N}, \mathsf{d}_i \in \mathsf{D}}$$

Thus, the set of all possible systems of model functions is delineated by

$$\mathsf{M} = \left\{ \prod_{i=1}^{m} \prod_{j=1}^{n} [\mathsf{s} \to \mathsf{o}_{ij}] \right\}_{i,j \in \mathbb{N}, \mathsf{o}_{ij} \in \mathsf{O}, s \in \mathsf{S}}$$

In a similar manner, we recall that any expression is either a model function, a natural, or a

constraint. Hence, the set of all possible meanings for these expressions is given by

$$\mathsf{T} = \{\mathsf{m}\}_{\mathsf{m}\in\mathsf{M}} \cup \{\{\eta\}\}_{\eta\in\mathbb{N}} \cup \{\square\}_{s\in\mathsf{S},\square\subseteq\mathsf{S}}$$

The set of all functions that map an arbitrary number of elements from T into another element in T is given by

$$\mathsf{F} = \left\{\left[\prod_{i=1}^{n} \mathsf{t}_i \to \mathsf{t}\right]\right\}_{n\in\mathbb{N},\mathsf{t},\mathsf{t}_i\in\mathsf{T}}$$

The meaning of the types is defined as follows

Convexity

$[\![\cdot]\!] : C \to \{[\mathsf{s} \to \mathsf{o}]\}_{\mathsf{o}\in\mathsf{S},\mathsf{s}\in\mathsf{S}}$

$[\![/]\!]=\{f \in [\mathsf{s} \to \mathbb{R}] : \forall\lambda \in [0,1],\ f(\lambda x + (1-\lambda)y) = \lambda f(x) + (1-\lambda) f(y)\}_{\mathsf{s}\in\mathsf{S}}$

$[\![\smile]\!]=\{f \in [\mathsf{s} \to \mathbb{R}] : \forall\lambda \in [0,1],\ f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)\}_{\mathsf{s}\in\mathsf{S}}$

$[\![\frown]\!]=\{f \in [\mathsf{s} \to \mathbb{R}] : \forall\lambda \in [0,1],\ f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda) f(y)\}_{\mathsf{s}\in\mathsf{S}}$

$[\![\perp]\!]=\{f \in [\mathsf{s} \to \mathsf{o}]\}_{\mathsf{o}\in\mathsf{O},\mathsf{s}\in\mathsf{S}}$

Polynomality

$[\![\cdot]\!] : P \to \{[\mathsf{s} \to \mathsf{o}]\}_{\mathsf{o}\in\mathsf{S},\mathsf{s}\in\mathsf{S}}$

$[\![\alpha]\!]=\{f \in [\mathsf{s} \to \mathbb{R}] :\ f(x) = \alpha\}_{\mathsf{s}\in\mathsf{S}}$

$[\![(\alpha, a)]\!]=\{f \in [\mathsf{s} \to \mathbb{R}] :\ f(x) = \alpha + \langle a, vec(x)\rangle\}_{\mathsf{s}\in\mathsf{S}}$

$[\![(\alpha, a, A)]\!]=\{f \in [\mathsf{s} \to \mathbb{R}] :\ f(x) = \alpha + \langle a, vec(x)\rangle + \langle Avec(x), vec(x)\rangle\}_{\mathsf{s}\in\mathsf{S}}$

$[\![\perp]\!]=\{f \in [\mathsf{s} \to \mathsf{o}]\}_{\mathsf{o}\in\mathsf{O},\mathsf{s}\in\mathsf{S}}$

Monotonicity

$[\![\cdot]\!] : M \to \{[\mathsf{s} \to \mathsf{o}]\}_{\mathsf{o}\in\mathsf{S},\mathsf{s}\in\mathsf{S}}$

$[\![-]\!]=\{f \in [\mathsf{s} \to \mathbb{R}] : \forall x \geq y,\, f(x) = f(y)\}_{\mathsf{s}\in\mathsf{S}}$

$[\![\nearrow]\!]=\{f \in [\mathsf{s} \to \mathbb{R}] : \forall x \geq y,\, f(x) \geq f(y)\}_{\mathsf{s}\in\mathsf{S}}$

$[\![\searrow]\!]=\{f \in [\mathsf{s} \to \mathbb{R}] : x \geq y,\, f(x) \leq f(y)\}_{\mathsf{s}\in\mathsf{S}}$

$[\![\perp]\!]=\{f \in [\mathsf{s} \to \mathsf{o}]\}_{\mathsf{o}\in\mathsf{O},\mathsf{s}\in\mathsf{S}}$

Codomain

$\boxed{[\![\cdot]\!] : O \to \mathsf{O}}$

$[\![\mathbb{R}]\!] = \mathbb{R}$

$[\![\mathbb{C}]\!] = \mathbb{C}$

$[\![\mathbb{Z}]\!] = \mathbb{Z}$

$[\![\{0, 1\}]\!] = \{0, 1\}$

$[\![\{-1, 1\}]\!] = \{-1, 1\}$

Symmetry

$\boxed{[\![\cdot]\!] : Y \to \mathsf{M}}$

$[\![\ddagger]\!]=\{f \in \mathsf{m} :\, f(x) = f(x)^T\}_{\mathsf{m}\in\mathsf{M}}$

$[\![\perp]\!]=\{f \in \mathsf{m}\}_{\mathsf{m}\in\mathsf{M}}$

Types

$\boxed{[\![\cdot]\!] : T \to \mathsf{T}}$

$$\left[\!\left[\left\{\langle c, p, m, o\rangle_{ij}^{mn}, y\right\}\right]\!\right]$$

$$=\left\{f \in [\mathsf{s} \to \mathbb{R}^{m\times n}] \cap [\![y]\!] :\, f_{ij} \in \left[\!\left[c_{ij}\right]\!\right] \cap \left[\!\left[p_{ij}\right]\!\right] \cap \left[\!\left[m_{ij}\right]\!\right] \cap \left\{g \in \left[\mathsf{s} \to \left[\!\left[o_{ij}\right]\!\right]\right]\right\}\right\}_{\mathsf{s}\in\mathsf{S}}$$

$[\![\eta]\!]=\{\eta\}$

$[\![\diamond]\!]=\{x \in X : X \subseteq \mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$

Function Types

$[\![\cdot]\!] : (\{T\}_i^n \to T) \to \mathsf{F}$

$$\left[\!\left[\{t\}_i^m \to t\right]\!\right]=\left[\prod_{i=1}^{m} [\![t_i]\!] \to [\![t]\!]\right]$$

Domain

$[\![\cdot]\!] : D \to \mathsf{D}$

$$[\![o^{m\times n}]\!] = \{[\![o]\!]^{m\times n}\}_{m,n\in\mathbb{N}}$$

$$[\![\mathcal{S}^n]\!] = \{\mathcal{S}^n\}_{n\in\mathbb{N}}$$

Decision Variable Context

$[\![\cdot]\!] : S \to \mathsf{S}$

$$\left[\!\left[\{x_i : d_i\}_{i=1}^n\right]\!\right]=\prod_{i=1}^{n} [\![d_i]\!]$$

As expected, the types associated with convexity, polynomality, monotonicity, and symmetry all map to a set of functions. Since we also track the pointwise codomain of each function, the meaning of each codomain type is a set of numbers. Combining these types, each expression must either be a model function, a natural number, or a set of feasible points. The remaining definitions characterize the meaning of each context.

The precise meaning of each remaining term depends on its type. Thus, the remaining semantics are defined on typing judgments. The meaning of these subexpressions is defined inductively as

Mathematical Program

$[\![\cdot]\!] : G \times N \rightharpoonup \mathbb{R}$

$$\left[\!\left[\Gamma \vdash \mathsf{min}\ e\ \mathsf{over}\ \Sigma\ \mathsf{st}\ \{e\}_i^n\right]\!\right]$$

$$= \min_{x \in A \cap B} [\![\Gamma; \Sigma \vdash e : \langle c, p, m, \mathbb{R} \rangle]\!](x)$$

where $A = [\![\Sigma]\!]$

$$B = \bigcap_{i=1}^{n} [\![\Gamma; \Sigma \vdash e_i : \diamond]\!]$$

Expressions

$[\![\cdot]\!] : G \times S \times E \times T \rightharpoonup \bigcup_{t \in T} t$

$[\![\Gamma; \Sigma \vdash A : \eta]\!] = A$

$[\![\Gamma; \Sigma \vdash A : \langle /, A, -, \mathbb{R} \rangle]\!] = x \in [\![\Sigma]\!] \mapsto A$

$\left[\!\left[\Gamma; \Sigma \vdash A : \left\{ \langle /, A_{ij}, -, \mathbb{R} \rangle_{ij}^{mn}, \mathcal{Y} \right\}\right]\!\right] = x \in [\![\Sigma]\!] \mapsto A$

$\left[\!\left[\Gamma; \{x : d\}_k^n \vdash x_k : \langle /, (0, e_{\iota(k,1,1)}), \nearrow, o \rangle\right]\!\right] = x \in \left[\!\left[\{x : d\}_k^n\right]\!\right] \mapsto x_k$

$\left[\!\left[\Gamma; \{x : d\}_k^n \vdash x_k : \left\{ \langle /, (0, e_{\iota(k,i,j)}), \nearrow, o \rangle_{ij}^{mn}, \perp \right\}\right]\!\right] = x \in \left[\!\left[\{x : d\}_k^n\right]\!\right] \mapsto x_k$

$\left[\!\left[\Gamma; \{x : d\}_k^n \vdash x_k : \left\{ \langle /, (0, e_{\iota(k,i,j)}), \nearrow, \mathbb{R} \rangle_{ij}^{mm}, \ddagger \right\}\right]\!\right] = x \in \left[\!\left[\{x : d\}_k^n\right]\!\right] \mapsto x_k$

$\left[\!\left[\Gamma; \Sigma \vdash f\{e\}_i^m : t\right]\!\right] = \left[\!\left[\Gamma \vdash f : \{t\}_i^m \to t\right]\!\right] \{[\![\Gamma; \Sigma \vdash e : t]\!]\}_i^m$

Scalar Functions

$[\![\cdot]\!] : G \times E \times (\{T\}_i^n \to T) \rightharpoonup \bigcup_{f \in F} f$

$\left[\!\left[\Gamma \vdash \geq_{\mathbb{R}_+} : \{\langle c, p, m, o \rangle\}_k^2 \to \diamond\right]\!\right] = f, g \mapsto \{x : f(x) \geq g(x)\}$

$\left[\!\left[\Gamma \vdash = : \{\langle c, p, m, o \rangle\}_k^2 \to \diamond\right]\!\right] = f, g \mapsto \{x : f(x) = g(x)\}$

$\left[\!\left[\Gamma \vdash + : \{\langle c, p, m, o \rangle\}_k^2 \to \langle c, p, m, o \rangle\right]\!\right] = f, g \mapsto (x \mapsto f(x) + g(x))$

$\left[\!\left[\Gamma \vdash - : \{\langle c, p, m, o \rangle\}_k^1 \to \langle c, p, m, o \rangle\right]\!\right] = f \mapsto (x \mapsto -f(x))$

$\left[\!\left[\Gamma \vdash - : \{\langle c, p, m, o \rangle\}_k^2 \to \langle c, p, m, o \rangle\right]\!\right] = f, g \mapsto (x \mapsto f(x) - g(x))$

$$\left[\!\left[\Gamma \vdash * : \{\langle c,p,m,o\rangle\}_k^2 \to \langle c,p,m,o\rangle\right]\!\right] = f,g \mapsto (x \mapsto f(x)g(x))$$

$$\left[\!\left[\Gamma \vdash / : \{\langle c,p,m,o\rangle\}_k^2 \to \langle c,p,m,o\rangle\right]\!\right] = f,g \mapsto (x \mapsto f(x)/g(x))$$

$$\left[\!\left[\Gamma \vdash |\cdot| : \{\langle c,p,m,o\rangle\}_k^1 \to \langle c,p,m,o\rangle\right]\!\right] = f \mapsto (x \mapsto |f(x)|)$$

$$\left[\!\left[\Gamma \vdash max : \{\langle c,p,m,o\rangle\}_k^2 \to \langle c,p,m,o\rangle\right]\!\right] = f,g \mapsto (x \mapsto \max(f(x),g(x)))$$

$$\left[\!\left[\Gamma \vdash min : \{\langle c,p,m,o\rangle\}_k^2 \to \langle c,p,m,o\rangle\right]\!\right] = f,g \mapsto (x \mapsto \min(f(x),g(x)))$$

$$\left[\!\left[\Gamma \vdash \cdot^{\cdot} : \{\langle c,p,m,o\rangle\}_k^2 \to \langle c,p,m,o\rangle\right]\!\right] = f,g \mapsto (x \mapsto f(x)^{g(x)})$$

Matrix Functions

$$[\![\cdot]\!] : G \times E \times (\{T\}_i^n \to T) \rightharpoonup \bigcup_{\mathsf{f}\in\mathsf{F}} \mathsf{f}$$

$$\left[\!\left[\Gamma \vdash \geq_{\mathbb{R}_+} : \left\{\{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \Diamond\right]\!\right] = f,g \mapsto \{x : f(x) \geq_{\mathbb{R}_+} g(x)\}$$

$$\left[\!\left[\Gamma \vdash \geq_{\mathbb{Q}} : \left\{\{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \Diamond\right]\!\right] = f,g \mapsto \{x : f(x) \geq_{\mathbb{Q}} g(x)\}$$

$$\left[\!\left[\Gamma \vdash \geq_{\mathcal{S}_+} : \left\{\{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \Diamond\right]\!\right] = f,g \mapsto \{x : f(x) \geq_{\mathcal{S}_+} g(x)\}$$

$$\left[\!\left[\Gamma \vdash =: \left\{\{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \Diamond\right]\!\right] = f,g \mapsto \{x : f(x) = g(x)\}$$

$$\left[\!\left[\Gamma \vdash + : \left\{\{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right]\!\right] = f,g \mapsto (x \mapsto f(x) + g(x))$$

$$\left[\!\left[\Gamma \vdash - : \left\{\{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right]\!\right] = f \mapsto (x \mapsto -f(x))$$

$$\left[\!\left[\Gamma \vdash - : \left\{\{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right]\!\right] = f,g \mapsto (x \mapsto f(x) - g(x))$$

$$\left[\!\left[\Gamma \vdash * : \left\{\{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right]\!\right] = f,g \mapsto (x \mapsto f(x)g(x))$$

$$\left[\!\left[\Gamma \vdash / : \left\{\{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right]\!\right] = f,g \mapsto (x \mapsto f(x)/g(x))$$

$$\left[\!\left[\Gamma \vdash \cdot_{:,:,:,:} : \left\{\{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right\}_k^1, \{\eta\}_k^4 \to \{\langle c,p,m,o\rangle_{ij}^{mn}, y\}\right]\!\right] = f,a,b,c,d \mapsto f_{a:b,c:d}(x)$$

$$\left[\!\left[\Gamma \vdash \cdot_{..} : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1, \{\eta\}_k^2 \to \langle c,p,m,o\rangle\right]\!\right] = f, a, b \mapsto f_{ab}(x)$$

$$\left[\!\left[\Gamma \vdash |\cdot| : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}\right]\!\right] = f \mapsto (x \mapsto Y) \text{ where } Y_{ij} = |f_{ij}(x)|$$

$$\left[\!\left[\Gamma \vdash max : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c,p,m,o\rangle\right]\!\right] = f \mapsto \left(x \mapsto \max_{i,j} f_{ij}(x)\right)$$

$$\left[\!\left[\Gamma \vdash min : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c,p,m,o\rangle\right]\!\right] = f \mapsto \left(x \mapsto \min_{i,j} f_{ij}(x)\right)$$

$$\left[\!\left[\Gamma \vdash sum : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c,p,m,o\rangle\right]\!\right] = f \mapsto \left(x \mapsto \sum_{i,j} f_{ij}(x)\right)$$

$$\left[\!\left[\Gamma \vdash \|\cdot\|_\infty : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c,p,m,o\rangle\right]\!\right] = f \mapsto (x \mapsto \|f(x)\|_\infty)$$

$$\left[\!\left[\Gamma \vdash \|\cdot\|_1 : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c,p,m,o\rangle\right]\!\right] = f \mapsto (x \mapsto \|f(x)\|_1)$$

$$\left[\!\left[\Gamma \vdash \|\cdot\|_2 : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c,p,m,o\rangle\right]\!\right] = f \mapsto (x \mapsto \|f(x)\|_2)$$

$$\left[\!\left[\Gamma \vdash \cdot^T : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}\right]\!\right] = f \mapsto (x \mapsto x^T)$$

$$\left[\!\left[\Gamma \vdash tr : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c,p,m,o\rangle\right]\!\right] = f \mapsto (x \mapsto \mathrm{tr}(x))$$

$$\left[\!\left[\Gamma \vdash [\cdot\,\cdot] : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^2 \to \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}\right]\!\right] = f, g \mapsto (x \mapsto [f(x)\, g(x)])$$

$$\left[\!\left[\Gamma \vdash \begin{bmatrix}\cdot\\ \cdot\end{bmatrix} : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^2 \to \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}\right]\!\right] = f, g \mapsto \left(x \mapsto \begin{bmatrix} f(x) \\ g(x)\end{bmatrix}\right)$$

$$\left[\!\left[\Gamma \vdash \begin{bmatrix}\cdot & \cdot\\ & \cdot\end{bmatrix} : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^3 \to \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}\right]\!\right] = f, g, h \mapsto \left(x \mapsto \begin{bmatrix} f(x) & g(x) \\ g^T(x) & h(x)\end{bmatrix}\right)$$

$$\left[\!\left[\Gamma; \Sigma \vdash \lambda_{max} : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c,p,m,o\rangle\right]\!\right] = f \mapsto (x \mapsto \lambda_{max}(f(x)))$$

$$\left[\!\left[\Gamma; \Sigma \vdash \lambda_{min} : \left\{\langle c,p,m,o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c,p,m,o\rangle\right]\!\right] = f \mapsto (x \mapsto \lambda_{min}(f(x)))$$

The meaning of a mathematical program follows our expectations. It consists of an objective function, the domain of all decision variables, and a list of constraints. Expressions consists of four possibilities. They can be a natural number, a function that maps its argument to a constant, a function that projects out the kth element, or a function application. The functions allowed during a function application fall into two categories. Either they define a set of feasible points or they compose their arguments into another function.

For example, consider the meaning of $x + |y|$ where $\Sigma = \{x : \mathbb{R}^{1\times 1}, y : \mathbb{R}^{1\times 1}\}$. We see that

$$
\begin{aligned}
&[\![\Gamma; \Sigma \vdash x + |y| : \langle \bot, \bot, \bot, \mathbb{R}\rangle]\!] \\
=&\left[\!\!\left[\Gamma \vdash + : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\!\right] \\
&\left\{\left[\!\!\left[\Gamma; \Sigma \vdash x : \left\langle /, \left(0, \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right), \nearrow, \mathbb{R}\right\rangle\right]\!\!\right], [\![\Gamma; \Sigma \vdash |y| : \langle \bot, \bot, \bot, \mathbb{R}\rangle]\!]\right\} \\
=&(f, g \mapsto (x \mapsto f(x) + g(x)))\{x \mapsto x_1, \\
&\left[\!\!\left[\Gamma \vdash |\cdot| : \{\langle c, p, m, o\rangle\}_k^1 \to \langle c, p, m, o\rangle\right]\!\!\right]\left\{\left[\!\!\left[\Gamma; \Sigma \vdash y : \left\langle /, \left(0, \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right), \nearrow, \mathbb{R}\right\rangle\right]\!\!\right]\right\}\right\} \\
=&(f, g \mapsto (x \mapsto f(x) + g(x)))\{x \mapsto x_1, (f \mapsto (x \mapsto |f(x)|))\{x \mapsto x_2\}\} \\
=&(f, g \mapsto (x \mapsto f(x) + g(x)))\{x \mapsto x_1, x \mapsto |x_2|\} \\
=&x \mapsto x_1 + |x_2|
\end{aligned}
$$

It is important to recognize that this function depends on the context Σ. For instance, if instead we defined $\Sigma = \{t : \mathbb{R}^{7\times 9}, x : \mathbb{R}^{1\times 1}, y : \mathbb{R}^{1\times 1}\}$, the meaning of $x + |y|$ would be $x \mapsto x_2 + |x_3|$. This seemingly minor detail will become extremely important during the discussion of transformations.

5.3 Soundness and Completeness

In the following section, we will prove two results. We will show that the meaning of any mathematical program is a real number. Additionally, we will show that it is not possible to represent every real number by the meaning of a mathematical program using our grammar. In other words, we will show that our language is sound, but not complete.

Theorem 1 (Soundness of Expressions)**.** *For $\Gamma \in G$ defined above and any $\Sigma \in S$, $e \in E$, and $t \in T$ such that $\Gamma; \Sigma \vdash e : t$, $[\![\Gamma; \Sigma \vdash e : t]\!] \in [\![t]\!]$.*

Proof. In the above presentation, we have been careful to separate scalar functions from matrix functions. We do this since proving soundness on scalar functions is more straightforward than matrix functions. In the following proof, we will consider all scalar cases first. Then, we will use these results to help prove the matrix cases. In both cases, the proof follows from induction on the structure of e.

In the first case, we must verify that $[\![\Gamma; \Sigma \vdash A : \eta]\!] \in [\![\eta]\!]$. Recall, our typing rules stipulate that A must be a natural number denoted by η. Thus, soundness follows immediately since $[\![\Gamma; \Sigma \vdash A : \eta]\!] = A$ and $[\![\eta]\!] = \{\eta\}$.

Next, we must check that

$$\left[\!\left[\Gamma; \{x : d\}_k^n \vdash x_k : \langle /, (0, e_{\iota(k,1,1)}), \nearrow, o\rangle\right]\!\right] \in [\![\langle /, (0, e_{\iota(k,1,1)}), \nearrow, o\rangle]\!]$$

Our strategy will be to analyze the properties of the function on the left hand side. Then, we will show that this function lies within the set defined on the right hand side.

We define the meaning of a scalar variable lookup as

$$\left[\!\left[\Gamma; \{x : d\}_k^n \vdash x_k : \langle /, (0, e_{\iota(k,1,1)}), \nearrow, o\rangle\right]\!\right] = x \in \left[\!\left[\{x : d\}_k^n\right]\!\right] \mapsto x_k$$

Certainly, this function is affine since

$$(\lambda x + (1 - \lambda)y)_k = \lambda x_k + (1 - \lambda)y_k$$

In addition, we see that this function is linear. Notice that

$$\frac{\partial(x \in \prod_{i=1}^n \mathcal{D}_k \mapsto x_k)}{\partial x_{\bar{k}}} = \begin{cases} 1 & \iota(k,1,1) = \bar{k} \\ 0 & otherwise \end{cases}$$

where $\mathcal{D}_p \subseteq \mathbb{R}^{r_p \times c_p}$ and we recall that $\iota(k,1,1) = 1 + \sum_{p=1}^{k-1} r_p c_p$. Therefore, we can exactly represent this function by its Taylor series

$$x_k = \langle e_{\iota(k,1,1)}, vec(x)\rangle$$

Further, this function is increasing. Take $x \succeq_{\mathbb{R}_+} y$. Then, by definition, $x_k \succeq_{\mathbb{R}_+} y_k$ for each k. Finally, we also see by definition that $x_k \in o$. Therefore, we can safely assume that the codomain of $x \in \left[\!\left[\{x : d\}_k^n\right]\!\right] \mapsto x_k$ lies in o.

Next, we must show that the above function lies within the following set

$$[\![\langle /, (0, e_{\iota(k,1,1)}), \nearrow, o\rangle]\!]$$
$$= \{f \in [\mathsf{s} \to \mathbb{R}] \cap [\![\ddagger]\!] : f \in [\![/]\!] \cap [\![(0, e_{\iota(k,1,1)})]\!] \cap [\![\nearrow]\!] \cap \{g \in [\mathsf{s} \to [\![o]\!]]\}\}_{\mathsf{s}\in\mathsf{S}}$$

Let us expand these terms. The first term becomes

$$f \in [\mathsf{s} \to \mathbb{R}] \cap [\![\ddagger]\!] = f \in [\mathsf{s} \to \mathbb{R}] \cap \{f \in \mathsf{m} : f(x) = f(x)^T\}_{\mathsf{m}\in\mathsf{M}}$$
$$= f \in [\mathsf{s} \to \mathbb{R}]$$

We can eliminate the explicit symmetry requirement as all scalar functions are symmetric. Thus, each element of the set must map some domain to a real number. The next term requires

$$f \in [\![/]\!] = \{f \in [\mathsf{s} \to \mathbb{R}] : \forall \lambda \in [0,1], f(\lambda x + (1-\lambda)y) = \lambda f(x) + (1-\lambda) f(y)\}_{\mathsf{s}\in\mathsf{S}}$$

In other words, this term stipulates that all functions in the set must be affine. Next, we define that

$$f \in [\![(0, e_{\iota(k,1,1)})]\!] = \{f \in [\mathsf{s} \to \mathbb{R}] : f(x) = \langle e_{\iota(k,1,1)}, vec(x)\rangle\}_{\mathsf{s}\in\mathsf{S}}$$

This states that each function in the set must be a linear polynomial where the linear coefficient contains only a single one in the $\iota(k, 1, 1)$ position. We also state that

$$f \in [\![\nearrow]\!] = \{f \in [\mathsf{s} \to \mathbb{R}] : \forall x \geq y,\, f(x) \geq f(y)\}_{\mathsf{s} \in \mathsf{S}}$$

This mandates that each function in the set must be increasing. Finally, we define that

$$\{g \in [\mathsf{s} \to [\![o]\!]]\} = \{g \in [\mathsf{s} \to o]\}$$

Thus, each function in the set must have codomain o. Since we verified that the function $x \in \left[\!\left[\{x : d\}_k^n\right]\!\right] \mapsto x_k$ possesses each of these properties, our definition of variable lookups is sound.

The soundness of a scalar constant follows in a similar manner. We must show that

$$[\![\Gamma; \Sigma \vdash A : \langle /, A, -, \mathbb{R}\rangle]\!] \in [\![\langle /, A, -, \mathbb{R}\rangle]\!]$$

By definition,

$$[\![\Gamma; \Sigma \vdash A : \langle /, A, -, \mathbb{R}\rangle]\!] = x \in [\![\Sigma]\!] \mapsto A$$

This trivially lies in the set

$$[\![\langle /, A, -, \mathbb{R}\rangle]\!]$$
$$= \left\{f \in [\mathsf{s} \to \mathbb{R}] \cap [\![\ddagger]\!] : f \in [\![/]\!] \cap \left[\!\left[A_{ij}\right]\!\right] \cap [\![-]\!] \cap \{g \in [\mathsf{s} \to [\![\mathbb{R}]\!]]\}\right\}_{\mathsf{s} \in \mathsf{S}}$$

Since this function is affine, constant, and both increasing and decreasing.

Now we consider the soundness of function application. Thus, we must show that

$$\left[\!\left[\Gamma; \Sigma \vdash f\{e\}_i^m : t\right]\!\right] \in [\![t]\!]$$

We define that

$$\left[\!\left[\Gamma; \Sigma \vdash f\{e\}_i^m : t\right]\!\right] = \left[\!\left[\Gamma \vdash f : \{t\}_i^m \to t\right]\!\right]\{[\![\Gamma; \Sigma \vdash e : t]\!]\}_i^m$$

By induction, we know that the meaning of each of the arguments is sound. Thus, we must check the soundness of each built-in function.

Consider two relations defined on scalars: the partial order defined by the positive orthant and equality. We must show that

$$\left[\!\left[\Gamma \vdash \geq_{\mathbb{R}_+} : \{\langle c, p, m, o\rangle\}_k^2 \to \diamond\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o\rangle\}_k^2 \to \diamond\right]\!\right]$$

$$\left[\!\left[\Gamma \vdash = : \{\langle c, p, m, o\rangle\}_k^2 \to \diamond\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o\rangle\}_k^2 \to \diamond\right]\!\right]$$

By definition, we see that

$$\left[\!\left[\Gamma \vdash \geq_{\mathbb{R}_+} : \{\langle c, p, m, o\rangle\}_k^2 \to \diamond\right]\!\right] = f, g \mapsto \{x : f(x) \geq g(x)\}$$

$$\left[\!\left[\Gamma \vdash = : \{\langle c, p, m, o\rangle\}_k^2 \to \diamond\right]\!\right] = f, g \mapsto \{x : f(x) = g(x)\}$$

Thus, the meaning of both of these judgments is a function that maps two functions to a

feasible set. We see that the meaning their types is

$$\left[\!\left[\{\langle c, p, m, o\rangle\}_k^2 \to \diamond\right]\!\right] = \left[\prod_{k=1}^{2} [\![\langle c, p, m, o\rangle_k]\!] \to [\![\diamond]\!]\right]$$

Thus, it is a set of functions where each function in the set maps two scalar model functions into a set of feasible points. Certainly, both functions above fall within this set. Therefore, their meaning is sound.

Next, we consider the meaning of addition. We must verify that

$$\left[\!\left[\Gamma \vdash + : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right]$$

for each type combination defined on page 35. By definition, we see that

$$\left[\!\left[\Gamma \vdash + : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] = f, g \mapsto (x \mapsto f(x) + g(x))$$

Since addition is an increasing, affine function, its composition with two affine functions is affine, two convex functions, convex, and two concave, concave. Further, the addition of two polynomials is another polynomial whose coefficients are the sum of those in the two composing functions. We also know that the addition of two constant functions is constant, increasing functions, increasing, and decreasing functions, decreasing. Finally, the addition of two integers must be an integer and the addition of two reals is real. These properties match those given by the typing rules, thus the meaning is sound.

We now examine the meaning of negation. We must check that

$$\left[\!\left[\Gamma \vdash - : \{\langle c, p, m, o\rangle\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o\rangle\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right]$$

for all the type combinations defined earlier on page 36. We specify that

$$\left[\!\left[\Gamma \vdash - : \{\langle c, p, m, o\rangle\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] = f \mapsto (x \mapsto -f(x))$$

Composition with negation reverses the convexity of a function. Thus, affine functions remain affine, convex become concave, and concave become convex. Similarly, composing with negation reverses the sort of monotonicity. Negation also preserves polynomality, but negates the coefficients. Finally, the negation of a plus minus-one integer remains plus-minus one integer, integer remains integer, real, real. As this matches the description given in the typing rules, the meaning is sound.

We declare the meaning of subtraction to be

$$\left[\!\left[\Gamma \vdash - : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] = f, g \mapsto (x \mapsto f(x) - g(x))$$

We must verify that

$$\left[\!\left[\Gamma \vdash - : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right]$$

for each type combination on page 37. Subtraction can be defined by combining addition

and negation. In other words, $f - g = f + (-g)$. Since we type subtraction by combining the addition and negation rules and their meaning is sound, the meaning of subtraction is also sound.

We define the meaning of multiplication as

$$\left[\!\left[\Gamma \vdash * : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] = f, g \mapsto (x \mapsto f(x)g(x))$$

We must determine whether

$$\left[\!\left[\Gamma \vdash * : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right]$$

for every type combination defined above on page 38. When a constant is positive, a constant scaling of a function preserves the convexity. Conversely, when a constant is negative, a constant scaling reverses the convexity. Next, when one function is a polynomial, a constant scaling of that polynomial remains a polynomial with scaled coefficients. When both functions are linear polynomials, their product is quadratic. Let $f(x) = \alpha + \langle a, x\rangle$ and

$g(x) = \beta + \langle b, x \rangle$. Then

$$\begin{aligned} f(x)g(x) &= (\alpha + \langle a, x \rangle)(\beta + \langle b, x \rangle) \\ &= (\alpha + \beta) + (\alpha\langle b, x \rangle + \beta\langle a, x \rangle) + \langle a, x \rangle\langle b, x \rangle \\ &= (\alpha + \beta) + \langle \alpha b + \beta a, x \rangle + x^T a b^T x \\ &= (\alpha + \beta) + \langle \alpha b + \beta a, x \rangle + \langle a b^T x, x \rangle \\ &= (\alpha + \beta) + \langle \alpha b + \beta a, x \rangle + \left\langle \frac{a b^T + b a^T}{2} x, x \right\rangle \end{aligned}$$

In the final step, we find the symmetric form of the quadratic coefficient. In addition, a positive scaling preserves the monotonicity of a function while a negative scaling reverses the monotonicity. Finally, if both arguments are zero-one, we know their product must be zero-one. Similarly, the product of two plus-minus one numbers remains plus-minus one, integer, integer, and real, real. Since each of these properties matches the type description, its meaning is sound.

Next, we consider division. We must ascertain whether

$$\left[\!\left[\Gamma \vdash / : \{\langle c, p, m, o \rangle\}_k^2 \to \langle c, p, m, o \rangle\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o \rangle\}_k^2 \to \langle c, p, m, o \rangle\right]\!\right]$$

for each type combination on page 39. By definition, we see that

$$\left[\!\left[\Gamma \vdash / : \{\langle c, p, m, o \rangle\}_k^2 \to \langle c, p, m, o \rangle\right]\!\right] = f, g \mapsto (x \mapsto f(x)/g(x))$$

When the second argument is constant, we have a constant scaling of the first function. This behaves identically to multiplying the first function by a constant scalar. In addition, we know that a plus-minus one number divided by another plus-minus one remains plus-minus one, an integer divided by a plus-minus one remains integer, and a real divided by a real remains real. As these properties align with the type description, the meaning is sound.

Examine the meaning of the absolute value function

$$\left[\!\left[\Gamma \vdash |\cdot| : \{\langle c, p, m, o\rangle\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] = f \mapsto (x \mapsto |f(x)|)$$

We must show that

$$\left[\!\left[\Gamma \vdash |\cdot| : \{\langle c, p, m, o\rangle\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o\rangle\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right]$$

for every type combination defined above on page 41. When the sole argument is constant, the resulting composition is constant. Therefore, the resulting function must be affine, a constant polynomial, and both increasing and decreasing. Further, the absolute value preserves each of the codomains that we define above. Since these properties agree with the type descriptions, the meaning is sound.

Now, we must certify that

$$\left[\!\left[\Gamma \vdash max : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right]$$

for each defined typing combination on page 41. We define that

$$\left[\!\left[\Gamma \vdash max : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] = f, g \mapsto (x \mapsto \max(f(x), g(x)))$$

When both arguments are constant, the resulting composition must be constant, hence, affine. Next, we show that when both functions are convex, the resulting composition is convex. We know that the max function is convex since

$$\begin{aligned}
\max(\lambda x_1 + (1-\lambda)y_1, \lambda x_2 + (1-\lambda)y_2) &= \max_{i\in\{1,2\}} \lambda x_i + (1-\lambda)y_i \\
&\leq \max_{i\in\{1,2\}} \lambda x_i + \max_{i\in\{1,2\}} (1-\lambda)y_i \\
&= \lambda \max_{i\in\{1,2\}} x_i + (1-\lambda) \max_{i\in\{1,2\}} y_i \\
&= \lambda \max(x_1, x_2) + (1-\lambda)\max(y_1, y_2)
\end{aligned}$$

It is also increasing since for $x \leq y$, $\max(x_1, x_2) \leq \max(y_1, y_2)$. Therefore, the composition between max and a set of convex functions must be convex. As mentioned above, when both arguments are constant, the resulting composition must be constant. Therefore, the composition will be a constant polynomial whose constant coefficient is equal to the maximum of the two constant coefficients. In addition, since the max function is increasing, when both arguments are increasing, the result is increasing. Conversely, when both

arguments are decreasing, we have a similar result. Notice that

$$\max\{f_1(x), f_2(x)\} \geq f_1(x) \geq f_1(y)$$

$$\max\{f_1(x), f_2(x)\} \geq f_2(x) \geq f_2(y)$$

Therefore, $\max\{f_1(x), f_2(x)\} \geq \max\{f_1(y), f_2(y)\}$. Thus, the composition is decreasing. Finally, we see that the maximum between two $\{0, 1\}$ integers remains $\{0, 1\}$, $\{-1, 1\}$ integer remains $\{-1, 1\}$, integer, integer, and real, real. As each of these observations matches the type descriptions, the meaning is sound.

Next, we assert that

$$\left[\!\left[\Gamma \vdash min : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right]$$

for each typing combination on page 42. We specify that

$$\left[\!\left[\Gamma \vdash min : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] = f, g \mapsto (x \mapsto \min(f(x), g(x)))$$

As with the max function, when both arguments are constants, the result must be constant and hence affine. By itself, the min function is concave where the proof follows from a similar argument as the one used to show the convexity of the max function. In addition, it is decreasing. Thus, the composition between min and a pair of concave functions remains concave. As mentioned above, the composition with two constants functions is constant.

The coefficient is equal to the minimum of the two arguments. Additionally, since the min function is decreasing, the composition with two decreasing functions remains decreasing. The composition with two increasing functions remains increasing where the proof follows analogously to the max case. Finally, we see that when both arguments have a $\{0, 1\}$ codomain, the composition maintains the $\{0, 1\}$ codomain. This also follows for $\{-1, 1\}$ integer, integer, and real codomains. Each of these properties follows the type descriptions. Thus, the meaning is sound.

Finally, we reestablish the definition of exponentiation as

$$\left[\!\left[\Gamma \vdash \cdot^{\cdot} : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] = f, g \mapsto (x \mapsto f(x)^{g(x)})$$

We must show that

$$\left[\!\left[\Gamma \vdash \cdot^{\cdot} : \{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right] \in \left[\!\left[\{\langle c, p, m, o\rangle\}_k^2 \to \langle c, p, m, o\rangle\right]\!\right]$$

for each of the typing combinations formerly defined on page 43. First, let us consider when the composition is affine. When both arguments are constant, the result must be constant, hence, affine. Alternatively, when the second argument is 0, then the resulting function must be a constant 1 which is also affine. In addition, when the first argument is affine and the second argument is 1, the composition is equivalent to the first argument. Therefore, the result is affine. Next, we consider when the composition is convex. Certainly, when the first argument is convex and the second is 1, the composition is equal to the first argument.

Thus, it must be convex. Next, we notice that $x \mapsto x^p$ is convex when p is positive and even since

$$\frac{\partial^2}{\partial x^2}(x \mapsto x^p) = x \mapsto p(p-1)x^{p-2}$$

and $p(p-1)x^{p-2} \geq 0$ for all x. Therefore, the composition of this function with an affine function must be convex. Alternatively, consider the case where the first argument is a convex, pure quadratic and the second argument is $1/2$. Let A_1 be the second-order coefficient of the first argument. Since this function is convex, $A_1 \succeq_{\mathcal{S}_+} 0$. Thus, we can factor A_1 into $U^T U$. Then, we notice that

$$\begin{aligned}
\sqrt{\langle A_1 vec(x), vec(x)\rangle} &= \sqrt{\langle U^T U vec(x), vec(x)\rangle} \\
&= \sqrt{\langle U vec(x), U vec(x)\rangle} \\
&= ||U vec(x)||_2 \\
&= ||vec(x)||_U
\end{aligned}$$

Since all norms are convex, the composition must be convex. Next, in a similar manner as before, when the first argument is concave and the second argument is 1, the result must be concave. Now, let us determine when the composition is a low-order polynomial. Of course, when both arguments are constant, we can directly compute the exponentiation which yields a constant polynomial. In a similar manner, when the second argument is 0, the result must be a constant 1. Next, when the first argument is a low-order polynomial and the second argument is 1, the result remains a low-order polynomial with the same

coefficients as the first argument. Alternatively, consider the case when the first argument is a linear polynomial of the form $\alpha + \langle a, vec(x)\rangle$ and we square the result. Then, we see that

$$\begin{aligned}(\alpha + \langle a, vec(x)\rangle)^2 =& \alpha^2 + 2\alpha\langle a, vec(x)\rangle + \langle a, vec(x)\rangle\langle a, vec(x)\rangle \\ =& \alpha^2 + \langle 2\alpha a, vec(x)\rangle + vec(x)^T aa^T vec(x) \\ =& \alpha^2 + \langle 2\alpha a, vec(x)\rangle + \langle aa^T vec(x), vec(x)\rangle\end{aligned}$$

Therefore, the composition is quadratic with coefficients $(\alpha^2, 2\alpha a, aa^T)$. Now, let us consider when the composition is monotonic. As we've mentioned before, when both arguments are constant, the result must be constant. Next, we notice that the function $x \mapsto x^p$ is increasing when $p \geq 1$ and odd since

$$\frac{\partial}{\partial x} x \mapsto x^p = x \mapsto px^{p-1}$$

and $px^{p-1} \geq 0$ for all x. Therefore, the composition of this function with an increasing function is increasing and the the composition with a decreasing function is decreasing. Finally, we consider the codomain of the function. When both arguments are zero-one integer, we have four possible cases: 0^0, 0^1, 1^0, and 1^1. As long as we define 0^0 as 0 or 1, in each case, the result is either 0 or 1. Next, when both arguments are plus-minus one integer, we also have four possible cases: 1^1, 1^{-1}, -1^1, and -1^{-1}. Thus, the result remains ± 1 integer. When both arguments produce an integer result and the second argument is a

positive constant, exponentiation reduces to multiplication. Since the product of an integer with another integer is an integer, the result of composition remains integer. Since each of these properties matches our definition, the meaning of exponentiation is sound.

At this point, we have verified soundness for all scalar functions. Now, we address matrix functions. This includes analyzing the soundness of matrix variable lookups and soundness of matrix functions. we use these results from the scalars cases to prove these results.

We begin by considering matrix variable lookups. There are two cases

$$\left[\!\left[\Gamma; \{x : d\}_k^n \vdash x_k : \left\{\langle /, (0, e_{\iota(k,i,j)}), \nearrow, o\rangle_{ij}^{mn}, \perp\right\}\right]\!\right] \in \left[\!\left[\left\{\langle /, (0, e_{\iota(k,i,j)}), \nearrow, o\rangle_{ij}^{mn}, \perp\right\}\right]\!\right]$$

$$\left[\!\left[\Gamma; \{x : d\}_k^n \vdash x_k : \left\{\langle /, (0, e_{\iota(k,i,j)}), \nearrow, \mathbb{R}\rangle_{ij}^{mm}, \ddagger\right\}\right]\!\right] \in \left[\!\left[\left\{\langle /, (0, e_{\iota(k,i,j)}), \nearrow, \mathbb{R}\rangle_{ij}^{mm}, \ddagger\right\}\right]\!\right]$$

The meaning of both cases is defined by

$$\begin{aligned}
&\left[\!\left[\Gamma; \{x : d\}_k^n \vdash x_k : \left\{\langle /, (0, e_{\iota(k,i,j)}), \nearrow, o\rangle_{ij}^{mn}, \perp\right\}\right]\!\right] \\
=&\left[\!\left[\Gamma; \{x : d\}_k^n \vdash x_k : \left\{\langle /, (0, e_{\iota(k,i,j)}), \nearrow, \mathbb{R}\rangle_{ij}^{mm}, \ddagger\right\}\right]\!\right] \\
=&x \in \left[\!\left[\{x : d\}_k^n\right]\!\right] \mapsto x_k
\end{aligned}$$

For each pointwise function, this reduces to the scalar case. Thus, we know that it is affine, a linear polynomial, and increasing for each pointwise function. Further, we can reason that the domain of each element is defined by o in the case of a general matrix or a real number in the case of a symmetric variable. Hence, the only property that we can not extrapolate

from the scalar case is symmetry. From the definition of the meaning of the type

$$\{f \in [\mathsf{s} \to \mathbb{R}] \cap [\![y]\!] : \ldots\}_{\mathsf{s} \in \mathsf{S}}$$

When $y = \ddagger$, we have that

$$[\![\ddagger]\!] = \{f \in \mathsf{m} : f(x) = f(x)^T\}_{\mathsf{m} \in \mathsf{M}}$$

Thus, when $x_k : \mathcal{S}^m$, we must verify that $x \mapsto x_k = x \mapsto x_k^T$. However, this follows immediately since the kth projection of the domain must be an element of the set $[\![\mathcal{S}^m]\!] = \mathcal{S}^m$. Thus, the meaning of variable lookups is sound.

We use a similar argument for matrix constants. We must show that

$$\left[\!\left[\Gamma; \Sigma \vdash A : \left\{\langle /, A_{ij}, -, \mathbb{R}\rangle_{ij}^{mn}, \mathcal{Y}\right\}\right]\!\right] \in \left[\!\left[\left\{\langle /, A_{ij}, -, \mathbb{R}\rangle_{ij}^{mn}, \mathcal{Y}\right\}\right]\!\right]$$

The pointwise properties of the constant function follow from those in the scalar case. Therefore, each element is affine, a constant polynomial, and both increasing and decreasing. Thus, we simply need to check symmetry. We stipulate that $\mathcal{Y} = \ddagger$ when the constant A is symmetric and $\perp$ when it is not. Since the function $x \mapsto A$ has a symmetric codomain when A is symmetric and our specification matches this property, our definition is sound.

We have already established that the soundness of application depends on the soundness of the built-in functions. Thus, we must check the remaining built-in functions; those that operate on matrices.

Consider the relations equality and the partial orders defined by the nonnegative orthant, second-order cone, and the cone of positive semidefinite matrices. We must show that

$$\left[\!\left[\Gamma \vdash =: \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond\right]\!\right] \in \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond$$

$$\left[\!\left[\Gamma \vdash \succeq_{\mathbb{R}_+}: \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond\right]\!\right] \in \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond$$

$$\left[\!\left[\Gamma \vdash \succeq_{Q}: \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond\right]\!\right] \in \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond$$

$$\left[\!\left[\Gamma \vdash \succeq_{\mathcal{S}_+}: \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond\right]\!\right] \in \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond$$

We stipulate that

$$\left[\!\left[\Gamma \vdash =: \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond\right]\!\right] = f, g \mapsto \{x : f(x) = g(x)\}$$

$$\left[\!\left[\Gamma \vdash \succeq_{\mathbb{R}_+}: \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond\right]\!\right] = f, g \mapsto \{x : f(x) \succeq_{\mathbb{R}_+} g(x)\}$$

$$\left[\!\left[\Gamma \vdash \succeq_{Q}: \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond\right]\!\right] = f, g \mapsto \{x : f(x) \succeq_{Q} g(x)\}$$

$$\left[\!\left[\Gamma \vdash \succeq_{\mathcal{S}_+}: \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond\right]\!\right] = f, g \mapsto \{x : f(x) \succeq_{\mathcal{S}_+} g(x)\}$$

The meaning of the common type is

$$\left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \diamond\right]\!\right] = \left[\prod_{k=1}^{2} \left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k\right]\!\right] \to [\![\diamond]\!]\right]$$

It is a set of functions where each function in the set maps two functions to a feasible region. The meaning of equality and nonnegativity lie in this set as long as both of their arguments are they same size. Fortunately, our typing rules require that $m_1 = m_2$ and $n_1 = n_2$. Next,

the meaning of the partial order defined by the second order cone lies in this set when both arguments are the same size and are vectors. The typing rules also require this. Finally, the meaning of the partial order defined by the cone of positive semidefinite matrices lies in the above set when both arguments are the same size, square, and symmetric. However, again, each of these requirements is enforced by the typing rules. Therefore, the meaning each of these functions is sound.

Now, we evaluate addition. We must verify that

$$\begin{aligned}\left[\!\left[\Gamma \vdash + : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] \\ \in \left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right]\end{aligned}$$

for each typing combination on page 46 where we define that

$$\left[\!\left[\Gamma \vdash + : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] = f, g \mapsto (x \mapsto f(x) + g(x))$$

Certainly, matrix addition can be defined pointwise. Thus, the soundness of each pointwise function follows from soundness of the scalar version of addition. We still must verify that both arguments have the same size and determine when the resulting function is symmetric. The typing rules require that $m_1 = m_2$ and $n_1 = n_2$. Thus, the composition is well-formed. In addition, when $y_1 = y_2 = \ddagger$, by induction, we know that both arguments must be symmetric. But, certainly the sum of two symmetric functions remains symmetric. Therefore, the meaning of addition is sound.

Next, we consider negation. We must ascertain whether

$$\begin{aligned}\left[\!\left[\Gamma \vdash - : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] \\ \in \left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right]\end{aligned}$$

for all typing combinations on page 46 where we define

$$\left[\!\left[\Gamma \vdash - : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] = f \mapsto (x \mapsto -f(x))$$

Like addition, we define matrix negation pointwise. Therefore, the soundness of each pointwise function follows from the soundness of scalar negation. Now, we must verify that the input and output arguments have the same size and determine when the output is symmetric. Since we require that $m = m_1$ and $n = n_1$, the composition will be well-formed. Also, when $y_1 = \ddagger$, by induction we know that the argument is symmetric. Since the negation of a symmetric matrix is also symmetric and we require that $y = \ddagger$ in this case, the meaning of negation is sound.

Now, we can examine subtraction. However, since we define subtraction by combining addition and negation and these functions are sound, the soundness of subtraction immediately follows.

We define the multiplication of a matrix by a scalar as

$$\left[\!\left[\Gamma \vdash * : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] = f, g \mapsto (x \mapsto f(x)g(x))$$

and must verify that

$$\left[\!\left[\Gamma \vdash * : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] \\ \in \left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right]$$

As with our other functions, this operation is defined pointwise on each element. Thus, the soundness of each pointwise property follows from the soundness of scalar multiplication. The typing rules also require that one of the arguments be scalar and that the size of the output must align with the matrix argument. Thus, the composition is well-formed. Finally, we know that a scaled symmetric matrix remains symmetric. But, the typing rules also state that $y = y_1$. Hence, the meaning of scalar multiplication is sound.

Similarly, we define the division of a matrix by a scalar as

$$\left[\!\left[\Gamma \vdash / : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] = f, g \mapsto (x \mapsto f(x)/g(x))$$

and must verify that

$$\left[\!\left[\Gamma \vdash / : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] \\ \in \left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right]$$

This case follows identically to scalar multiplication where the soundness follows from soundness of scalar division.

Next we consider submatrixing. We must determine whether

$$\left[\!\left[\Gamma \vdash \cdot_{\cdot:\cdot,\cdot:\cdot} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1, \{\eta\}_k^4 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] \\ \in \left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1, \{\eta\}_k^4 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right]$$

where we define that

$$\left[\!\left[\Gamma \vdash \cdot_{\cdot:\cdot,\cdot:\cdot} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1, \{\eta\}_k^4 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] = f, a, b, c, d \mapsto f_{a:b,c:d}(x)$$

Submatrixing doesn't change any of the pointwise properties; it merely selects certain elements. Thus, as long as we select the correct elements, the soundness of the pointwise properties follows from induction. We state that type of the (i, j)th element is that of $f_{(i+\eta_1)(j+\eta_3)}$ where f is the first argument. This aligns with the definition of submatrixing, so the pointwise properties are sound. Next, we must check the size of the result and conclude when it is symmetry. We require that $m_1 \geq \eta_3 \geq \eta_1$ and $n_1 \geq \eta_4 \geq \eta_2$. Further, we state that $m = \eta_2 - \eta_1 + 1$ and $n = \eta_4 - \eta_3 + 1$. So, we can not have any negatively sized matrices nor can the result be larger than the input. Since this size also matches the definition of submatrixing, the definition of size is sound. Further, when $y_1 = \ddagger$, by induction we know that the input is symmetric. Thus, all of the principle submatrices are symmetric. This occurs when $\eta_1 = \eta_3$ and $\eta_2 = \eta_4$. But, in this case, we state that $y = \ddagger$. Thus, the meaning of submatrixing is sound.

The meaning of subindexing, save symmetry, is defined in terms of submatrixing. Since the result is scalar, it must be symmetric. Therefore, since the meaning of submatrixing is

sound, the soundness of subindexing immediately follows.

We define the meaning of matrix multiplication as

$$\left[\!\left[\Gamma \vdash * : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] = f, g \mapsto (x \mapsto f(x)g(x))$$

We must verify that

$$\begin{aligned}\left[\!\left[\Gamma \vdash * : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] \\ \in \left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right]\end{aligned}$$

for each type combination on page 49. We defined these type combinations in terms of addition, scalar multiplication, and submatrixing. Since the meaning of each of these operations is sound and matrix multiplication is defined by these operations, the meaning of multiplication is sound.

We give the meaning of absolute value as

$$\left[\!\left[\Gamma \vdash |\cdot| : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] = f \mapsto (x \mapsto Y) \text{ where } Y_{ij} = |f_{ij}(x)|$$

We must determine that

$$\begin{aligned}\left[\!\left[\Gamma \vdash |\cdot| : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] \\ \in \left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right]\end{aligned}$$

for all defined typing combinations on page 50. Since the absolute value function is defined

pointwise by the scalar absolute value function, the soundness of each pointwise property follows. We simply need to check the size and determine when the result is symmetric. Our definition above requires that the input and output have identical size. But, we require this during typing since $m = m_1$ and $n = n_1$. Further, the result must be symmetric when the input is symmetric. But, we require that $y = y_1$. Thus, the meaning of absolute value is sound.

Next, we define the meaning of elementwise maximum as

$$\left[\!\left[\Gamma \vdash max : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] = f \mapsto \left(x \mapsto \max_{i,j} f_{ij}(x)\right)$$

and must verify that

$$\begin{aligned}&\left[\!\left[\Gamma \vdash max : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] \\ &\quad \in \left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right]\end{aligned}$$

for each typing combination on page 50. Notice that we can equivalently define elementwise maximum with binary maximum using the following recursive application

$$\begin{aligned}&max(f) \\ &\quad = max(f_{11}, max(f_{21}, \ldots, max(f_{m1}, max(f_{12}, \ldots, max(f_{m-1,n}, f_{mn})\ldots)\ldots)\ldots)\ldots)\end{aligned}$$

Thus, the soundness of elementwise maximum follows from the soundness of binary maximum. In the degenerate case, when $m = n = 1$, $\max_{i,j} f_{ij}(x) = f_{11}(x)$. In this case, soundness follows from induction. Therefore, the meaning of elementwise maximum is

sound.

The meaning of elementwise minimum follows in the exact same manner as elementwise maximum. Its soundness follows from the soundness of binary minimum.

We recount the meaning summation as

$$\left[\!\left[\Gamma \vdash sum : \left\{\langle c, p, m, o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] = f \mapsto \left(x \mapsto \sum_{i,j} f_{ij}(x)\right)$$

We must see if

$$\left[\!\left[\Gamma \vdash sum : \left\{\langle c, p, m, o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] \in \left[\!\left[\left\{\langle c, p, m, o\rangle_{ij}^{mn}, y\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right]$$

for all typing combinations on page 51. Similar to elementwise minimum and maximum, we notice that we can define summation using the following recursive definition

$$sum(f) = f_{11} + f_{21} + \cdots + f_{m1} + f_{12} + \cdots + f_{m-1,n} + f_{mn}$$

Therefore, the soundness of summation follows the soundness of addition. Further, when $m_1 = n_1 = 1$, we sum nothing and the soundness follows from induction. Therefore, the meaning of summation is sound.

Next, we recall the meaning of each defined p-norm as

$$\left[\!\left[\Gamma \vdash \|\cdot\|_\infty : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] = f \mapsto (x \mapsto \|f(x)\|_\infty)$$
$$\left[\!\left[\Gamma \vdash \|\cdot\|_1 : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] = f \mapsto (x \mapsto \|f(x)\|_1)$$
$$\left[\!\left[\Gamma \vdash \|\cdot\|_2 : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] = f \mapsto (x \mapsto \|f(x)\|_2)$$

We must verify that all three of these functions lie within the set

$$\left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right]$$

for the same set of typing combinations defined above on page 52. For any p-norm, when the input is completely constant, the output will also be constant. Simply, it is the p-norm of the input. Thus, in this case, the output is a constant polynomial. In addition, when this occurs, we know that the output must also be affine and both increasing and decreasing. Next, since the definition of a norm requires it to be subadditive and positively homogeneous, all norms are convex. As a result, when it is composed with an affine function, the result must be convex. Finally, composition with this function always returns a real number. Since these properties align with the type definitions, the meaning of each p-norm is sound.

The meaning of transpose is defined as

$$\left[\!\left[\Gamma \vdash \cdot^T : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] = f \mapsto (x \mapsto x^T)$$

We must check whether this function lies in the set

$$\left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right]$$

for each typing combination formerly defined on page 52. Recall that transpose does not change the pointwise properties of a matrix valued function. Simply, it rearranges the elements. In our typing rules, we stipulate that $m = n_1$, $n = m_1$, and that the type of the (i, j)th element in the result is equal to the type of the (j, i)th element in the argument. Further, we require that $y = y_1$. As this follows the definition of transpose, the meaning is sound.

Now, we define the meaning of trace as

$$\left[\!\left[\Gamma \vdash tr : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] = f \mapsto (x \mapsto \mathrm{tr}(x))$$

We must show that this function lies within

$$\left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right]$$

for each typing combination defined above on page 53. We defined these typing combinations in terms of addition and subindexing. As the meaning of both addition and subindexing is sound and their combination follows the definition of trace, the meaning of trace must be sound.

Next, we consider the meaning of horizontal concatenation

$$\left[\!\left[\Gamma \vdash [\cdot\,\cdot] : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] = f, g \mapsto (x \mapsto [f(x)\; g(x)])$$

We must verify that this function lies within

$$\left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right]$$

for the typing combinations defined previously on page 53. The horizontal concatenation of two matrices lies within the set $[\mathbb{R}^{m\times n_1} \times \mathbb{R}^{m\times n_2} \to \mathbb{R}^{m\times n_1+n_2}]$. We satisfy this condition by requiring that $m_1 = m_2$ and specifying that $m = m_1$, $n = n_1 + n_2$. In addition, we clearly see that pointwise properties of this function are defined by f and g. We denote this fact in our typing rules when we state that (i, j)th property mirrors f_{ij} for $j \leq n_1$ and $g_{i,j-n_1}$, otherwise. Thus, the meaning of horizontal concatenation is sound.

In a similar manner, we define the meaning of vertical concatenation by

$$\left[\!\left[\Gamma \vdash \begin{bmatrix}\cdot\\ \cdot\end{bmatrix} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right] = f, g \mapsto \left(x \mapsto \begin{bmatrix} f(x)\\ g(x)\end{bmatrix}\right)$$

We must demonstrate that this function lies within

$$\left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^2 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right]\!\right]$$

for each typing combination defined above on page 54. By definition, the vertical concatenation of two matrices lies in the set $[\mathbb{R}^{m_1 \times n} \times \mathbb{R}^{m_2 \times n} \to \mathbb{R}^{m_1+m_2 \times n}$. We fulfill this requirement by mandating that $n_1 = n_2$ and defining that $m = m_1 + m_2$, $n = n_1$. Similar to horizontal concatenation, the functions f and g define the pointwise properties of the result. We indicate this property in our typing rules when we define that the (i, j)th property is equal to f_{ij} for $i \leq m_1$ and $g_{i-m_1,j}$, otherwise. Therefore, the meaning of vertical concatenation is sound.

In addition, we specify the meaning of symmetric concatenation as

$$\left[\!\left[\Gamma \vdash \begin{bmatrix} \cdot & \cdot \\ & \\ & \cdot \end{bmatrix} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^3 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\} \right]\!\right] = f, g, h \mapsto \left(x \mapsto \begin{bmatrix} f(x) & g(x) \\ g^T(x) & h(x) \end{bmatrix} \right)$$

We must check whether this function belongs to the set

$$\left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^3 \to \{\langle c, p, m, o\rangle_{ij}^{mn}, y\} \right]\!\right]$$

for all typing combinations previously defined on page 54. The symmetric concatenation of three matrices has size defined by

$$\begin{array}{c} n_1 + n_2 \\ \left[\begin{array}{cc} \overset{n_1}{\boxed{f(x)}}_{m_1} & \overset{n_2}{\boxed{g(x)}}_{m_2} \\ \overset{m_2}{\boxed{g^T(x)}}_{n_2} & \overset{n_3}{\boxed{h(x)}}_{m_3} \end{array} \right] \end{array} m_2 + m_3$$

Thus, the function requires that $m_1 = m_2$, $n_2 = n_3$, $n_1 = m_1 = m_2$, and $n_2 = m_3 = n_3$. We

also see that the result has size n_1+n_2 by m_2+m_3. Next, we can clearly see that the pointwise properties of the result depend on the pointwise properties of the functions. Consider the (i, j)th property of the result. When $1 \leq i \leq m_1$ and $1 \leq j \leq n_1$, this property is defined by f_{ij}. Moving to the right, we offset the column index by n_1. Thus, when $1 \leq i \leq m_1$ and $n_1 + 1 \leq j \leq n_1 + n_2$, the property is specified by $g_{i,j-n_1}$. When considering the lower left corner, we offset the row index by m_1. Therefore, when $m_1+1 \leq i \leq m_1+m_3$ and $1 \leq j \leq n_1$, the functional properties are denoted by $g_{j,i-m_1}$. Finally, we consider the last block, we offset the row index by m_1 and the column index by n_1. Thus, when $m_1 + 1 \leq i \leq m_1 + m_3$ and $n_1 + 1 \leq j \leq n_1 + n_2$, the pointwise properties are found by analyzing $h_{i-m_1,j-n_1}$. Finally, we must verify symmetry of the result. However, this follows immediately from the block structure of the matrix and since both $f(x)$ and $h(x)$ are symmetric. As these properties match the type definition, the meaning is sound.

Now, let us recall the meaning of the maximum eigenvalue function

$$\left[\!\left[\Gamma; \Sigma \vdash \lambda_{max} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] = f \mapsto (x \mapsto \lambda_{max}(f(x)))$$

We must determine whether this function lies in the set

$$\left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right]$$

for each typing combination defined before on page 55. For the maximum eigenvalue function to be well-defined, its argument must be symmetric. Otherwise, complex eigenvalues

may exist and the maximum is not defined. We require this by stating that $m_1 = n_1$ and $y = \ddagger$. Next, when the argument is completely constant, we can compute the maximum eigenvalue directly. In this case, the result of the composition is constant and equal to the above computation. Also in this situation, the result must be affine and both increasing and decreasing. Next, we see that the maximum eigenvalue function is convex since

$$\begin{aligned}
\lambda_{max}(\alpha X + (1-\alpha)Y) &= \max_{\|z\|_2=1} z^T(\alpha X + (1-\alpha)Y)z \\
&= \max_{\|z\|_2=1} \alpha z^T Xz + (1-\alpha)z^T Yz \\
&\leq \max_{\|z\|_2=1} \alpha z^T Xz + \max_{\|z\|_2=1}(1-\alpha)z^T Yz \\
&= \alpha \max_{\|z\|_2=1} z^T Xz + (1-\alpha) \max_{\|z\|_2=1} z^T Yz \\
&= \alpha\lambda_{max}(X) + (1-\alpha)\lambda_{max}(Y)
\end{aligned}$$

Since the composition of a convex function with an affine function is convex, when the argument is affine, the result must be convex. Since each of these properties matches our type definitions, our meaning is sound.

Finally, we recollect the definition of the minimum eigenvalue function as

$$\left[\!\left[\Gamma; \Sigma \vdash \lambda_{min} : \left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right] = f \mapsto (x \mapsto \lambda_{min}(f(x)))$$

We must resolve whether this composition lies within

$$\left[\!\left[\left\{\{\langle c, p, m, o\rangle_{ij}^{mn}, y\}\right\}_k^1 \to \langle c, p, m, o\rangle\right]\!\right]$$

based on our definition above on page 56. As with the maximum eigenvalue function, this function is well-defined only when the argument is square and symmetric. Assuming a well-formed argument, when the argument is constant, we compute the minimum eigenvalue directly. Therefore, the result must be a constant polynomial, affine, and both increasing and decreasing. Next, we observe that the minimum eigenvalue function is concave since

$$\begin{aligned}
\lambda_{min}(\alpha X + (1-\alpha)Y) &= \min_{\|z\|_2=1} z^T(\alpha X + (1-\alpha)Y)z \\
&= \min_{\|z\|_2=1} \alpha z^T X z + (1-\alpha) z^T Y z \\
&\geq \min_{\|z\|_2=1} \alpha z^T X z + \min_{\|z\|_2=1} (1-\alpha) z^T Y z \\
&= \alpha \min_{\|z\|_2=1} z^T X z + (1-\alpha) \min_{\|z\|_2=1} z^T Y z \\
&= \alpha \lambda_{min}(X) + (1-\alpha)\lambda_{min}(Y)
\end{aligned}$$

Therefore, its composition with an affine function is concave. As each of these properties follows the type definitions, the meaning is sound.

□

Since we compose mathematical programs from expressions, we now have all the results necessary to prove the soundness of mathematical programs. This is characterized in the following theorem.

Theorem 2 (Soundness of Mathematical Programs)**.** *For* $\Gamma \in G$ *defined above and any* $n \in N$ *such that* $\Gamma \vdash n$, $[\![\Gamma \vdash n]\!] \in \mathbb{R}$.

Proof. We define that

$$\left[\!\left[\Gamma \vdash \mathsf{min}\ e\ \mathsf{over}\ \Sigma\ \mathsf{st}\ \{e\}_i^n\right]\!\right] = \min_{x \in A \cap B} [\![\Gamma; \Sigma \vdash e : \langle c, p, m, \mathbb{R}\rangle]\!](x)$$

$$\text{where} \quad A = [\![\Sigma]\!]$$

$$B = \bigcap_{i=1}^{n} [\![\Gamma; \Sigma \vdash e_i : \diamond]\!]$$

This will be a well-formed mathematical program when $[\![\Gamma; \Sigma \vdash e : \langle c, p, m, \mathbb{R}\rangle]\!]$ is a real scalar function and both A and B define a feasible region. From the soundness of expressions, we know that $[\![\Gamma; \Sigma \vdash e : \langle c, p, m, \mathbb{R}\rangle]\!]$ is a real scalar function with a domain defined by Σ. In addition, the soundness of expressions insures that B defines a feasible region. We simply need to verify that $[\![\Sigma]\!]$ defines a feasible region. By definition,

$$\left[\!\left[\{x_i : d_i\}_{i=1}^n\right]\!\right] = \prod_{i=1}^{n} [\![d_i]\!]$$

But, this is trivially a feasible region. Therefore, the mathematical program is well-formed.

Of course, it is possible that the feasible region is empty or the solution is unbounded. When there are no feasible points, we say that the solution is $+\infty$. When the solution is unbounded, we say that the solution is $-\infty$. Then, by real, we really mean the extended real numbers. Using this definition, the meaning of mathematical programs is sound. □

Related to soundness is completeness. Although neither mathematical programs nor expressions are complete, the incompleteness of mathematical programs does not stem from the incompleteness of expressions. We see this in the following two theorems.

Theorem 3 (Incompleteness of Mathematical Programs)**.** *For any arbitrary $\alpha \in \mathbb{R}$, we can not necessarily construct an $n \in N$ such that for $\Gamma \in G$ that contains all previously defined builtin-in functions, $[\![\Gamma \vdash n]\!] = \alpha$.*

Proof. Simply, there are an uncountable number of real numbers. Since our language can only represent a countable number of programs, the language can not possibly be complete.

□

The above proof does not give much information as to how complete our language truly is. The following theorem gives far more insight

Theorem 4 (Completeness of Mathematical Programs with Respect to Rationals)**.** *For any arbitrary $\alpha \in \mathbb{Q}$, there exists $n \in N$ such that for $\Gamma \in G$ that contains all previously defined builtin-in functions, $[\![\Gamma \vdash n]\!] = \alpha$.*

Proof. Let α be any rational number. Then notice that

$$\left[\!\left[\Gamma \vdash \min \alpha \text{ over } x : \mathbb{R}^{1\times 1} \text{ st } \{\}\right]\!\right] = \min_{x\in\mathbb{R}} \alpha = \alpha$$

□

This result is slightly surprising since it holds even when we do not define any built-in functions. Further, it does not require completeness of expressions since we only use rational constants.

Of course, incompleteness of mathematical programs is not particularly restrictive. We wish to accurately represent problems using a mathematical program, not construct trivial

examples that give a particular result. Our ability to represent problems is directly related to the functions we have available. Thus, the completeness of expressions gives far more information about the richness of the language. These results are summarized in the following theorem.

Theorem 5 (Incompleteness of Expressions)**.** *For any arbitrary* $\tau \in \bigcup_{t \in T} t$*, we can not necessarily construct a* $\Sigma \in S$*,* $e \in E$*, and* $t \in T$ *such that for* $\Gamma \in G$ *that contains all previously defined builtin-in functions,* $[\![\Gamma; \Sigma \vdash e : t]\!] = \tau$*.*

Proof. The set $\bigcup_{t \in T} t$ contains an uncountable number of model functions. Since our language can only define a countable number of built-in functions, expressions can not be complete.

□

With a sufficiently thorough set of base functions, expressions become far more complete. Since there are an uncountable number of model functions, we can never achieve full completeness. However, we can define enough functions to make the language useful.

Unfortunately, there is a more subtle problem with our language. Even if we can represent a function, the type of that function may be more general than necessary. In other words, a function may possess a property, but our type system can not prove the result. For example, let $\Gamma = x : \mathbb{R}^{1\times 1}$ and consider the expression $x * x * x$. This expression has type

$\langle \perp, \perp, \perp, \mathbb{R} \rangle$ since

$$\dfrac{(Mult1) \quad (Var) \qquad \dfrac{(Mult2) \quad (Var) \quad (Var)}{\Gamma; \Sigma \vdash x * x : \langle \cup, (0,0,1), \perp, \mathbb{R} \rangle}}{\Gamma; \Sigma \vdash x * x * x : \langle \perp, \perp, \perp, \mathbb{R} \rangle}$$

$$\Gamma; \Sigma \vdash x : \langle /, (0,1), \nearrow, \mathbb{R} \rangle \quad (Var)$$

$$\Gamma \vdash * : \{\langle /, (0,1), \nearrow, \mathbb{R} \rangle, \langle \cup, (0,0,1), \perp, \mathbb{R} \rangle\} \to \langle \perp, \perp, \perp, \mathbb{R} \rangle \quad (Mult1)$$

$$\Gamma \vdash * : \{\langle /, (0,1), \nearrow, \mathbb{R} \rangle, \langle /, (0,1), \nearrow, \mathbb{R} \rangle\} \to \langle \cup, (0,0,1), \perp, \mathbb{R} \rangle \quad (Mult2)$$

Yet, $f = x \mapsto x * x * x$ is monotonically increasing since $f'(x) = 3x^2 \geq 0$ for all x. We can further see that this is not simply a problem with monotonicity. Consider the expression $x * x * x * x$. This expression has type $\langle \perp, \perp, \perp, \mathbb{R} \rangle$ since

$$\dfrac{(Mult3) \quad (Var) \quad \Gamma; \Sigma \vdash x * x * x : \langle \perp, \perp, \perp, \mathbb{R} \rangle}{\Gamma; \Sigma \vdash x * x * x * x : \langle \perp, \perp, \perp, \mathbb{R} \rangle}$$

$$\Gamma \vdash * : \{\langle /, (0,1), \nearrow, \mathbb{R} \rangle, \langle \perp, \perp, \perp, \mathbb{R} \rangle\} \to \langle \perp, \perp, \perp, \mathbb{R} \rangle \quad (Mult3)$$

Yet, $f = x \mapsto x * x * x * x$ is convex since $f''(x) = 12x^2 \geq 0$ for all x. Finally, we can see

that we also have the same problem with polynomality since

$$\frac{(Div) \quad (Var) \quad \Gamma; \Sigma \vdash x * x * x : \langle \bot, \bot, \bot, \mathbb{R} \rangle}{\Gamma; \Sigma \vdash x * x * x / x : \langle \bot, \bot, \bot, \mathbb{R} \rangle}$$

$$\Gamma \vdash / : \{\langle \bot, \bot, \bot, \mathbb{R} \rangle, \langle /, (0, 1), \nearrow, \mathbb{R} \rangle\} \to \langle \bot, \bot, \bot, \mathbb{R} \rangle \quad (Div)$$

Yet, $f = x \mapsto x * x * x / x = x^2$ is obviously quadratic.

Chapter 6

Transformations

In the following section we will establish a series of important transformations and relaxations used in mathematical programming. We begin by describing each transformation. Then, we will show that each transformation does not change the meaning of a mathematical program.

6.1 Absolute Value

We can transform an inequality containing the absolute value function into multiple inequalities that contain linear functions. Consider the case where x is scalar. The constraint $y \geq |x|$ states that y must be greater than both the positive and negative components of x. Thus, we can add each of these requirements as a separate constraint.

We generalize this concept in the following transformation

$$\begin{aligned}\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} |e_2| : \diamond) = \quad & \Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_2 : \diamond, \\ & \Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} -e_2 : \diamond\end{aligned}$$

We see these two representations are equivalent since

$$\begin{aligned}
[\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} |e_2| : \diamond]\!] &= \{x : f(x) \succeq_{\mathbb{R}_+} |g(x)|\} \\
&= \{x : f_{ij}(x) \geq |g_{ij}(x)|\} \\
&= \{x : f_{ij}(x) \geq g_{ij}(x), f_{ij}(x) \geq -g_{ij}(x)\} \\
&= \{x : f(x) \succeq_{\mathbb{R}_+} g(x), f(x) \succeq_{\mathbb{R}_+} -g(x)\} \\
&= \{x : f(x) \succeq_{\mathbb{R}_+} g(x)\} \cap \{x : f(x) \succeq_{\mathbb{R}_+} -g(x)\} \\
&= [\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_2 : \diamond, \Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} -e_2 : \diamond]\!] \\
&= [\![\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} |e_2| : \diamond)]\!]
\end{aligned}$$

where $1 \leq i \leq m$ and $1 \leq j \leq n$.

6.2 Expanding an Inequality

This transformation lies at the heart of many other transformations. Consider the case where we have a constraint of the form $z \geq x + |y|$. We would like to transform the absolute value function, but this constraint does not follow the format defined above. Recall, we defined the absolute value transformation only for constraints of the form

$$\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} |e_2| : \diamond$$

In our current example, we have a constraint of the form

$$\Gamma; \Sigma \vdash e_1 \succeq add(x, |y|) : \diamond$$

Thus, we can not immediately access the absolute value function since it is encapsulated within addition. Fortunately, we can reach it by adding an auxiliary variable. This allows us to replace the above constraint by the pair

$$z \succeq x + t$$

$$t \succeq |y|$$

Then, we can reformulate the constraint $t \succeq |y|$ using the transformation defined above.

We define our strategy according to the following rules. Consider a constraint of the form

$$h(x) \succeq_{\mathbb{R}_+} f(g_1(x), \ldots, g_m(x))$$

where $x \in \prod_{k=1}^{n} \mathcal{D}_k$ and $\mathcal{D}_k \in \{\mathbb{R}^{m \times n}, \mathcal{S}^m, \mathbb{Z}^{m \times n}, \{0,1\}^{m \times n}, \{-1,1\}^{m \times n}\}$. Define the extraction set E to be the set of all indices that correspond to functions, g_i, that are neither constant nor a projection of the original variables. In this case, there is no benefit in extracting the function. Let $\hat{g}_i$ be a new candidate set of arguments where

$$\hat{g}_i(y) = \begin{cases} g_i(\pi_x(y)) & i \notin E \\ y_{\kappa(i)} & i \in E \end{cases}$$

We define κ to be an injective function from E to the set $\{n+1,\ldots,n+|E|\}$ and $y \in \prod_{k=1}^{n} \mathcal{D}_k \times \prod_{k=n+1}^{n+|E|} \mathbb{R}^{m_k \times n_k}$ where $m_{\kappa(i)}$ and $n_{\kappa(i)}$ match the size of the codomain of g_i. We also define $\pi_x : \prod_{k=1}^{n} \mathcal{D}_k \times \prod_{k=n+1}^{n+|E|} \mathbb{R}^{m_k \times n_k} \to \prod_{k=1}^{n} \mathbb{R}^{m_k \times n_k}$ to be the projection where $(\pi_x(y))_k = y_k$ for $k = 1 \ldots n$. In other words, we project out the original domain of each function. This allows us to expand the domain of all functions to include the new auxiliary variables. Then, when the function

$$y \mapsto f(\hat{g}_1(y), \ldots, \hat{g}_m(y))$$

is monotonically increasing, we may reformulate the constraint into

$$h(\pi_x(y)) \succeq_{\mathbb{R}_+} f(\hat{g}_1(y), \ldots, \hat{g}_m(y))$$

$$y_{\kappa(i)} \succeq_{\mathbb{R}_+} g_i(\pi_x(y))$$

for $i \in E$.

Let us apply this procedure to the constraint $z \geq x + |y|$. We can rewrite this function as $x_1 \geq x_2 + |x_3|$ for $x \in \mathbb{R}^3$. Now, since the first argument to addition is a decision variable, we do not expand it. Alternatively, the second argument is neither constant nor a decision variable, so we must expand it. Thus, our candidate set of arguments is

$$\hat{g}_1(y) = [\pi_x(y)]_3 = y_3$$

$$\hat{g}_2(y) = y_4$$

where $\pi_x : \mathbb{R}^4 \to \mathbb{R}^3$ and $[\pi_x(y)]_k = y_k$ for $k = 1, 2, 3$. The function

$$y \in \mathbb{R}^4 \mapsto y_3 + y_4$$

is monotonically increasing. Thus, we may expand the above constraint into

$$y_1 \geq y_2 + y_4$$

$$y_4 \geq |y_3|$$

which is our desired reformulation.

Before we prove the correctness of this transformation, let us consider one additional example. Ruminate over the constraint $x \geq f(4, |x|, 7)$ where $f(x, y, z) = xyz$. According to the above expansion rules, we can reformulate this constraint into

$$x \geq 4(y)7$$

$$y \geq |x|$$

Although we can easily see that this set of constraints is equivalent to the original, this example is interesting since f is not monotonically increasing. That is why we are careful to distinguish when f is increasing and when the composition $x \mapsto f(\hat{g}_1(x), \ldots, \hat{g}_m(x))$ is increasing.

We must prove this strategy produces an equivalent feasible region. Let

$$x \in \left\{ x \in \prod_{k=1}^{n} \mathcal{D}_k : h(x) \succeq_{\mathbb{R}_+} f(g_1(x), \ldots, g_m(x)) \right\}$$

We must show that there exists

$$y \in \left\{ y \in \prod_{k=1}^{n} \mathcal{D}_k \times \prod_{k=n+1}^{n+|E|} \mathbb{R}^{m_k \times n_k} : h(\pi_x(y)) \succeq_{\mathbb{R}_+} f(\hat{g}_1(y), \ldots, \hat{g}_m(y)), y_{\kappa(i)} \succeq_{\mathbb{R}_+} g_i(\pi_x(y)), i \in E \right\}$$

such that $\pi_x(y) = x$. Choose y such that $y_i = x_i$ for $i = 1 \ldots n$ and $y_{\kappa(i)} = g_i(x)$ for $i \in E$. Certainly, $x = \pi_x(y)$. Therefore, we must check that y lies in the reformulated feasible region. Combining the rules of the transformational strategy and our definition of y we see that

$$\hat{g}_i(y) = g_i(\pi_x(y)) = g_i(x)$$

for $i \notin E$ and

$$\hat{g}_i(y) = y_{\kappa(i)} = g_i(x)$$

for $i \in E$. In other words, $\hat{g}_i(y) = g_i(x)$ for all i. Thus,

$$h(\pi_x(y)) = h(x) \succeq_{\mathbb{R}_+} f(g_1(x), \ldots, g_m(x)) = f(\hat{g}_1(y), \ldots, \hat{g}_m(y))$$

This shows that we satisfy the first constraint in our reformulated region. Now, we must verify that we satisfy the remaining constraints. Since $y_{\kappa(i)} = g_i(x) = g_i(\pi_x(y))$, for $i \in E$,

we see that

$$y_{\kappa(i)} = g_i(x) = g_i(\pi_x(y)) \succeq_{\mathbb{R}_+} g_i(\pi_x(y))$$

Thus, we satisfy the remaining constraints. Therefore, y is feasible and a projection of the reformulated feasible region is a subset of the original feasible region. In the reverse direction, we take

$$y \in \left\{ y \in \prod_{k=1}^{n} \mathcal{D}_k \times \prod_{k=n+1}^{n+|E|} \mathbb{R}^{m_k \times n_k} : h(\pi_x(y)) \succeq_{\mathbb{R}_+} f(\hat{g}_1(y), \ldots, \hat{g}_m(y)), y_{\kappa(i)} \succeq_{\mathbb{R}_+} g_i(\pi_x(y)), i \in E \right\}$$

We must show that there exists

$$x \in \left\{ x \in \prod_{k=1}^{n} \mathcal{D}_k : h(x) \succeq_{\mathbb{R}_+} f(g_1(x), \ldots, g_m(x)) \right\}$$

such that $x = \pi_x(y)$. Choose x such that $x = \pi_x(y)$. We must show that x lies within the original feasible region. Choose $\tilde{y}$ such that $\tilde{y}_{\kappa(i)} = g_i(\pi_x(y))$ for $i \in E$ and $\tilde{y}_i = y_i$ for $i \in \{1, \ldots, n\}$. Notice that $\tilde{y}$ possesses three properties. First, we see that $\pi_x(\tilde{y}) = x$ since $\tilde{y}_i = y_i$ for $i \in \{1, \ldots, n\}$. Second, we know that $y \succeq_{\mathbb{R}_+} \tilde{y}$ since $y_{\kappa(i)} \succeq_{\mathbb{R}_+} g_i(\pi_x(y)) = \tilde{y}_{\kappa(i)}$ for $i \in E$ and $y_i = \tilde{y}_i$ for $i \in \{1, \ldots, n\}$. Third, we have that $\hat{g}_i(\tilde{y}) = g_i(\pi_x(\tilde{y}))$ since $\hat{g}_i(\tilde{y}) = \tilde{y}_{\kappa(i)} = g_i(\pi_x(y)) = g_i(x) = g_i(\pi_x(\tilde{y}))$ for $i \in E$ and $\hat{g}_i(\tilde{y}) = g_i(\pi_x(\tilde{y}))$ for $i \notin E$. Since

the function $y \mapsto f(\hat{g}_1(y), \ldots, \hat{g}_m(y))$ is monotonically increasing, we have that

$$\begin{aligned} h(x) = h(\pi_x(y)) &\geq_{\mathbb{R}_+} f(\hat{g}_1(y), \ldots, \hat{g}_m(y)) \\ &\geq_{\mathbb{R}_+} f(\hat{g}_1(\tilde{y}), \ldots, \hat{g}_m(\tilde{y})) \\ &= f(g_1(\pi_x(\tilde{y})), \ldots, g_m(\pi_x(\tilde{y}))) \\ &= f(g_1(x), \ldots, g_m(x)) \end{aligned}$$

Therefore, x is feasible and the original feasible region is a subset of a projection of the reformulated feasible region. Hence, both feasible regions are equivalent up to a projection.

We define this transformation formally as

$$\mathcal{T}(\Gamma; \Sigma \vdash e_0 \geq_{\mathbb{R}_+} f\{e\}_i^m : \diamond) = \Gamma; \hat{\Sigma} \vdash e_0 \geq_{\mathbb{R}_+} f\{\hat{e}\}_i^m : \diamond, \{\Gamma; \hat{\Sigma} \vdash x_{\kappa(\cdot)} \geq_{\mathbb{R}_+} e : \diamond\}^{i \in E}$$

when

$$\Gamma; \hat{\Sigma} \vdash f\{\hat{e}\}_i^m : \{\langle c, p, \nearrow, o \rangle_{ij}^{st}, y\}$$

where

$$\Sigma = \{x\}_j^n$$

$$\hat{\Sigma} = \{x\}_j^{n+|E|}$$

$$\hat{e}_i = \begin{cases} x_j & e_i = x_j \\ A & e_i = A \\ x_{\kappa(i)} & otherwise \end{cases}$$

$$E = \{i : e_i \neq x_j, e_i \neq A\}$$

and κ is an injective function between E and $\{n+1, \ldots, n+|E|\}$.

In order to show that these two formulations are equivalent, we begin by noting that we have have already shown that

$$\begin{aligned}
&[\![\Gamma; \Sigma \vdash e_0 \succeq_{\mathbb{R}_+} f\{e\}_i^m : \diamond]\!] \\
=&\left\{ x \in \prod_{k=1}^{n} \mathcal{D}_k : h(x) \succeq_{\mathbb{R}_+} f(g_1(x), \ldots, g_m(x)) \right\} \\
=&\pi_x \left\{ y \in \prod_{k=1}^{n} \mathcal{D}_k \times \prod_{k=n+1}^{n+|E|} \mathbb{R}^{m_k \times n_k} : h(\pi_x(y)) \succeq_{\mathbb{R}_+} f(\hat{g}_1(y), \ldots, \hat{g}_m(y)), y_{\kappa(i)} \succeq_{\mathbb{R}_+} g_i(\pi_x(y)), i \in E \right\} \\
=&\pi_x \left[\!\!\left[\Gamma; \hat{\Sigma} \vdash e_0 \succeq_{\mathbb{R}_+} f\{\hat{e}\}_i^m : \diamond, \{\Gamma; \hat{\Sigma} \vdash x_{\kappa(\cdot)} \succeq_{\mathbb{R}_+} e : \diamond\}^{i \in E} \right]\!\!\right]
\end{aligned}$$

as long as $x \mapsto f(\hat{g}_1(y), \ldots, \hat{g}_m(y))$ is monotonically increasing. However, we assume that

$$\Gamma; \hat{\Sigma} \vdash f\{\hat{e}\}_i^m : \{\langle c, p, \nearrow, o \rangle_{ij}^{st}, y\}$$

Since our semantics are sound, this implies that

$$
\begin{aligned}
& x \mapsto f(\hat{g}_1(x), \ldots, \hat{g}_m(x)) \\
\in & \left[\!\left[\{\langle c, p, \nearrow, o\rangle_{ij}^{st}, y\} \right]\!\right] \\
= & \left\{ f \in [\mathsf{s} \to \mathbb{R}^{s \times t}] \cap [\![y]\!] : f_{ij} \in \left[\!\left[c_{ij} \right]\!\right] \cap \left[\!\left[p_{ij} \right]\!\right] \cap [\![\nearrow]\!] \cap \left\{ g \in \left[\mathsf{s} \to \left[\!\left[o_{ij} \right]\!\right] \right] \right\} \right\}_{\mathsf{s} \in \mathsf{S}}
\end{aligned}
$$

Thus, the function $x \mapsto f(\hat{g}_1(x), \ldots, \hat{g}_m(x))$ is monotonically increasing. Therefore, both formulations are equivalent up to a projection.

There are three other related transformations. First, we define that

$$\mathcal{T}(\Gamma; \Sigma \vdash e_0 \succeq_{\mathbb{R}_+} f\{e\}_i^m : \diamond) = \Gamma; \hat{\Sigma} \vdash e_0 \succeq_{\mathbb{R}_+} f\{\hat{e}\}_i^m : \diamond, \{\Gamma; \hat{\Sigma} \vdash e \succeq_{\mathbb{R}_+} x_{\kappa(\cdot)} : \diamond\}^{i \in E}$$

when

$$\Gamma; \hat{\Sigma} \vdash f\{\hat{e}\}_i^m : \{\langle c, p, \searrow, o\rangle_{ij}^{st}, y\}$$

This is equivalent to the first transformation except that the composition $x \mapsto f(\hat{g}_1(x), \ldots, \hat{g}_m(x))$ is monotonically decreasing. Second, we also define that

$$\mathcal{T}(\Gamma; \Sigma \vdash f\{e\}_i^m \succeq_{\mathbb{R}_+} e_0 : \diamond) = \Gamma; \hat{\Sigma} \vdash f\{\hat{e}\}_i^m \succeq_{\mathbb{R}_+} e_0 : \diamond, \{\Gamma; \hat{\Sigma} \vdash e \succeq_{\mathbb{R}_+} x_{\kappa(\cdot)} : \diamond\}^{i \in E}$$

when

$$\Gamma; \hat{\Sigma} \vdash f\{\hat{e}\}_i^m : \{\langle c, p, \nearrow, o\rangle_{ij}^{st}, y\}$$

This is equivalent to the first transformation except that we expand the left hand side instead of the right. Finally, we define that

$$\mathcal{T}(\Gamma; \Sigma \vdash f\{e\}_i^m \succeq_{\mathbb{R}_+} e_0 : \diamond) = \Gamma; \hat{\Sigma} \vdash f\{\hat{e}\}_i^m \succeq_{\mathbb{R}_+} e_0 : \diamond, \{\Gamma; \hat{\Sigma} \vdash x_{\kappa(\cdot)} \succeq_{\mathbb{R}_+} e : \diamond\}^{i \in E}$$

when

$$\Gamma; \hat{\Sigma} \vdash f\{\hat{e}\}_i^m : \{\langle c, p, \searrow, o \rangle_{ij}^{st}, y\}$$

We see this transformation is identical to the original except that we expand the left hand side and the composition is decreasing. The proof that each of these transformations produces an equivalent feasible region follows in a manner similar to the proof above.

6.3 Expanding an Equality

We expand an inequality constraint by adding a set of new auxiliary variables and inequality constraints. While this constitutes a valid transformation, we could add equality constraints instead. For example, we could expand the constraint $z \geq x + |y|$ into $z \geq x + t$ and $t = |x|$. In fact, we can always add an equality constraint regardless if the original function is monotonic. Unfortunately, in most instances, this makes the problem more difficult to solve. For example, the constraint $t = |x|$ is not convex while the inequality $t \geq |x|$ preserves convexity. Therefore, we only want use this transformation when the new constraint, not the original, is convex. This occurs only when the left and right hand sides of the new constraint are affine.

We formalize this strategy according to the following rules. Consider a constraint of the form

$$h(x) \text{ R } f(g_1(x), \ldots, g_m(x))$$

where $x \in \prod_{k=1}^{n} \mathbb{R}^{m_k \times n_k}$ and R denotes an arbitrary relation. Let the extraction set E be the set of all indices where the function g_i is affine. Let $\hat{g}_i$ be the new set of arguments where

$$\hat{g}_i(y) = \begin{cases} g_i(\pi_x(y)) & i \notin E \\ y_{\kappa(i)} & i \in E \end{cases}$$

where we define κ, π_x, and y analogously to our discussion above. Then, we may reformulate the above constraint into

$$h(\pi_x(y)) \text{ R } f(\hat{g}_1(y), \ldots, \hat{g}_m(y))$$

$$y_{\kappa(i)} = g_i(\pi_x(y))$$

for $i \in E$.

We must prove this transformation produces an equivalent feasible region. Let

$$x \in \left\{ x \in \prod_{k=1}^{n} \mathcal{D}_k : h(x) \text{ R } f(g_1(x), \ldots, g_m(x)) \right\}$$

We must show that there exists

$$y \in \left\{ y \in \prod_{k=1}^{n} \mathcal{D}_k \times \prod_{k=n+1}^{n+|E|} \mathbb{R}^{m_k \times n_k} : h(\pi_x(y)) \ \mathrm{R} \ f(\hat{g}_1(y), \ldots, \hat{g}_m(y)), y_{\kappa(i)} = g_i(\pi_x(y)), i \in E \right\}$$

such that $\pi_x(y) = x$. Choose y such that $y_i = x_i$ for $i = 1 \ldots n$ and $y_{\kappa(i)} = g_i(x)$ for $i \in E$. We immediately see that $\pi_x(y) = x$. Thus, we must verify that y lies within the reformulated feasible region. By combining the rules of the transformational strategy and our definition of y, we have that

$$\hat{g}_i(y) = g_i(\pi_x(y)) = g_i(x)$$

for $i \notin E$ and

$$\hat{g}_i(y) = y_{\kappa(i)} = g_i(x)$$

for $i \in E$. Thus, $\hat{g}_i(y) = g_i(x)$ for all i. Therefore,

$$h(\pi_x(y)) = h(x) \ \mathrm{R} \ f(g_1(x), \ldots, g_m(x)) = f(\hat{g}_1(y), \ldots, \hat{g}_m(y))$$

Therefore, we satisfy the first constraint. Now, we must verify that we satisfy the remaining constraints. By construction,

$$y_{\kappa(i)} = g_i(x)$$

for $i \in E$. Therefore, y lies within the reformulated feasible region and a projection of the reformulated feasible region is a subset of the original feasible region. In the reverse

direction, we take

$$y \in \left\{ y \in \prod_{k=1}^{n} \mathcal{D}_k \times \prod_{k=n+1}^{n+|E|} \mathbb{R}^{m_k \times n_k} : h(\pi_x(y)) \ \mathrm{R} \ f(\hat{g}_1(y), \ldots, \hat{g}_m(y)), y_{\kappa(i)} = g_i(\pi_x(y)), i \in E \right\}$$

We must show there exist

$$x \in \left\{ x \in \prod_{k=1}^{n} \mathcal{D}_k : h(x) \ \mathrm{R} \ f(g_1(x), \ldots, g_m(x)) \right\}$$

such that $x = \pi_x(y)$. Choose x such that $x = \pi_x(y)$. We must demonstrate that x lies within the original feasible region. Since $\hat{g}_i(y) = y_{\kappa(i)} = g_i(\pi_x(y)) = g_i(x)$ for $i \in E$ and $\hat{g}_i(y) = g_i(\pi_x(y)) = g_i(x)$ for $i \notin E$, we know that $\hat{g}_i(y) = g_i(x)$ for all i. Thus, we have that

$$\begin{aligned} h(x) = h(\pi_x(y)) \ \mathrm{R} \ & f(\hat{g}_1(y), \ldots, \hat{g}_m(y)) \\ = & f(g_1(x), \ldots, g_m(x)) \end{aligned}$$

Thus, x lies within the original feasible region and the original feasible region is a subset of a projection of the reformulated feasible region. Therefore, both feasible regions are equivalent up to a projection.

We define this expansion transformation formally as

$$\mathcal{T}(\Gamma; \Sigma \vdash e_0 \ \mathrm{R} \ f\{e\}_i^m : \diamond) = \Gamma; \hat{\Sigma} \vdash e_0 \ \mathrm{R} \ f\{\hat{e}\}_i^m : \diamond, \{\Gamma; \hat{\Sigma} \vdash x_{\kappa(\cdot)} = e : \diamond\}^{i \in E}$$

where

$$\Sigma = \{x\}_j^n$$

$$\hat{\Sigma} = \{x\}_j^{n+|E|}$$

$$\hat{e}_i = \begin{cases} x_{\kappa(i)} & \Gamma; \Sigma \vdash e_i : \{\langle /, p, m, o \rangle_{ij}^{st}, y\} \\ e_i & otherwise \end{cases}$$

$$E = \left\{ i : \Gamma; \Sigma \vdash e_i : \{\langle /, p, m, o \rangle_{ij}^{st}, y\} \right\}$$

and κ is an injective function between E and $\{n+1, \ldots, n+|E|\}$.

In order to prove that these two formulations are equivalent, we notice that

$$\begin{aligned}
&[\![\Gamma; \Sigma \vdash e_0 \text{ R } f\{e\}_i^m : \diamond]\!] \\
=&\left\{ x \in \prod_{k=1}^{n} \mathbb{R}^{m_k \times n_k} : h(x) \text{ R } f(g_1(x), \ldots, g_m(x)) \right\} \\
=&\pi_x \left\{ y \in \prod_{k=1}^{n} \mathcal{D}_k \times \prod_{k=n+1}^{n+|E|} \mathbb{R}^{m_k \times n_k} : h(\pi_x(y)) \text{ R } f(\hat{g}_1(y), \ldots, \hat{g}_m(y)), y_{\kappa(i)} = g_i(\pi_x(y)), i \in E \right\} \\
=&\pi_x \left[\!\!\left[\Gamma; \hat{\Sigma} \vdash e_0 \text{ R } f\{\hat{e}\}_i^m : \diamond, \{\Gamma; \hat{\Sigma} \vdash x_{\kappa(\cdot)} = e : \diamond\}^{i \in E} \right]\!\!\right]
\end{aligned}$$

Therefore, both formulations are equivalent up to a projection.

We have one additional transformation intimately related to the first

$$\mathcal{T}(\Gamma; \Sigma \vdash f\{e\}_i^m \text{ R } e_0 : \diamond) = \Gamma; \hat{\Sigma} \vdash f\{\hat{e}\}_i^m \text{ R } e_0 : \diamond, \{\Gamma; \hat{\Sigma} \vdash x_{\kappa(\cdot)} = e : \diamond\}^{i \in E}$$

where we define the extraction set E as above. Simply, we reverse the left and right hand sides. We can prove the correctness of this transformation using a similar argument as above.

6.4 Contracting a Single Auxiliary Variable and a Single Inequality

The transformational process requires that we expand most constraints before we can effectively transform them. As a byproduct of this process, we dramatically increase the size of the problem. Although the resulting problem is extremely sparse, we prefer that our final result be smaller. We term the process of eliminating constraints and auxiliary variables as contraction. In general, determining redundant constraints is extremely difficult. Our focus will be on eliminating the sort of constraint generated by expansion.

Consider the problem

$$\begin{aligned} \min_{(x,y,z,t)\in\mathbb{R}^4} \quad & z \\ \text{st} \quad & z \geq x + t \\ & t \geq |y| \\ & x \geq 0 \end{aligned}$$

By inspection, we can see that optimal value is 0 and the optimal solution is $(0, 0, 0, 0)$.

Noticing the exposed absolute value, we reformulate this problem into

$$\begin{aligned} \min_{(x,y,z,t)\in\mathbb{R}^4} \quad & z \\ \text{st} \quad & z \geq x + t \\ & t \geq y \\ & t \geq -y \\ & x \geq 0 \end{aligned}$$

At this point, we observe that we use the variable t in only three places: the constraints $z \geq x + t$, $t \geq y$, and $t \geq -y$. In addition, the function $(x, y, z, t) \mapsto x + t$ is monotonically increasing. Thus, we can contract the variable t and reformulate the problem into

$$\begin{aligned} \min_{(x,y,z)\in\mathbb{R}^3} \quad & z \\ \text{st} \quad & z \geq x + y \\ & z \geq x - y \\ & x \geq 0 \end{aligned}$$

We can see this problem also has an optimal value of 0 and a solution of $(0, 0, 0)$.

We codify this process in the following steps. Consider a problem of the form

$$\begin{aligned}
&\min_{y \in A \cap B} f(y) \\
&\text{where} \quad A = \prod_{k=1}^{n} \mathcal{D}_k \\
&\qquad\qquad B = \bigcap_{i=1}^{s} \{y : g_i(y) \ \mathrm{R} \ h_i(y)\} \cap \\
&\qquad\qquad\qquad \{y : g(y) \succeq_{\mathbb{R}_+} h(y)\} \cap \\
&\qquad\qquad\qquad \bigcap_{j=1}^{t} \{y : y_q \succeq_{\mathbb{R}_+} v_j(y)\}
\end{aligned}$$

for some fixed $q \in \{1, \ldots, n\}$ where R denotes an arbitrary relation and we define $\mathcal{D}_k$ as above. This problem has the same optimal value as

$$\begin{aligned}
&\min_{x \in \hat{A} \cap \hat{B}} f(\pi_y(x)) \\
&\text{where} \quad \hat{A} = \left(\prod_{k=1}^{q-1} \mathcal{D}_k \right) \times \left(\prod_{k=q+1}^{n} \mathcal{D}_k \right) \\
&\qquad\qquad \hat{B} = \bigcap_{i=1}^{s} \{x : g_i(\pi_y(x)) \ \mathrm{R} \ h_i(\pi_y(x))\} \cap \\
&\qquad\qquad\qquad \bigcap_{j=1}^{t} \{x : g(\pi_y(x)) \succeq_{\mathbb{R}_+} h(\pi_j(x))\}
\end{aligned}$$

where we define the injection $\pi_y : \hat{A} \to A$ as

$$[\pi_y(x)]_k = \begin{cases} x_k & k < q \\ \sup_j v_j(x_1, \ldots, x_{q-1}, 0, x_q, \ldots, x_{n-1}) & k = q \\ x_{k-1} & k \geq q \end{cases}$$

and the injection $\pi_j : \hat{A} \to A$ as

$$[\pi_j(x)]_k = \begin{cases} x_k & k < q \\ v_j(x_1, \ldots, x_{q-1}, 0, x_q, \ldots, x_{n-1}) & k = q \\ x_{k-1} & k \geq q \end{cases}$$

In addition, we must satisfy the following conditions. First, we require that $f(y) = f(z)$, $g_i(y) = g_i(z)$, $g(y) = g(z)$, and $v(y) = v(z)$ for all $y, z \in A$ where $y_k = z_k$ for all k save q. In other words, the only function that may depend on y_q must be h. Second, the function h must be monotonically increasing.

We must prove both formulations produce the same optimal value. Define the projection $\pi_x : A \to \hat{A}$ by $[\pi_x(y)]_k = y_k$ when $k < q$ and $[\pi_x(y)]_k = y_{k+1}$ when $k \geq q$. Our argument adheres to the following steps. First, take an optimal solution to the original problem, y^*. Then, show that $\pi_x(y^*)$ is feasible in the reformulated problem. Since the objective function does not depend on y_q, this tells us that the optimal value of the reformulated problem is less than or equal to the original. Second, take an optimal solution to the reformulated problem, x^*. Show that $\pi_y(x^*)$ is feasible in the original problem. Again, since the objective function does not depend on y_q, the optimal value of the original problem must be less than or equal to the reformulated. Therefore, both problems must possess the same optimal value.

Let y^* be optimal in the original problem and consider the point $x = \pi_x(y^*)$. We must show that x is feasible in the reformulated problem. Since g_i and h_i do not depend on y_q, we satisfy the constraint $g_i(\pi_y(x))$ R $h_i(\pi_y(x))$. Therefore, we must show that $g(\pi_y(x)) \succeq_{\mathbb{R}_+}$

$h(\pi_j(x))$. Since $y_q^* \succeq_{\mathbb{R}_+} v_j(y^*)$ for all j, we know that $y_q^* \succeq_{\mathbb{R}_+} \sup_j v_j(y^*) = [\pi_y(x)]_q$. Hence, $y^* \succeq_{\mathbb{R}_+} \pi_y(x)$. Since h is increasing, we observe that

$$g(\pi_y(x)) = g(y^*) \succeq_{\mathbb{R}_+} h(y^*) \succeq_{\mathbb{R}_+} h(\pi_y(x)) \succeq_{\mathbb{R}_+} h(\pi_j(x))$$

Thus, we have shown that x is feasible in the reformulated problem. Since the objective function does not depend on y_q, $f(y^*) = f(\pi_y(x))$. Therefore, the optimal value of the reformulated problem must be less than or equal to the original.

Let x^* be optimal in the reformulated problem and consider the point $y = \pi_y(x^*)$. We must show that y is feasible in the original problem. Since g_i and h_i do not depend on y_q, we satisfy the constraint $g_i(y)$ R $h_i(y)$. Thus, we must demonstrate that $g(y) \succeq_{\mathbb{R}_+} h(y)$ and that $y_q \succeq_{\mathbb{R}_+} v_i(y)$. First, we see that

$$g(y) = g(\pi_y(x^*)) \succeq_{\mathbb{R}_+} h(\pi_y(x^*)) = h(y)$$

We also see that

$$y_q = [\pi_y(x^*)]_q = \sup_j v_j(x_1^*, \ldots, x_{q-1}^*, 0, x_q^*, \ldots, x_{n-1}^*) = \sup_j v_j(y) \succeq_{\mathbb{R}_+} v_j(y)$$

Thus, we see that y is feasible in the original problem. Since the objective function does not depend on y_q, we see that $f(\pi_y(x^*)) = f(y)$. Hence, the optimal value of the original problem must be less than or equal to the reformulated. However, combined with the first part of our argument, this implies that both problems possess the same optimal value.

We define this transformation formally with the following

$$\mathcal{T}\left(\Gamma \vdash \min e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s, \hat{e}_1 \succeq_{\mathbb{R}_+} \hat{e}_2, \{x_q \succeq_{\mathbb{R}_+} \hat{e}_j\}_j^t\right)$$

$$= \Gamma \vdash \min e \text{ over } \{x : d\}_k^{q-1}, \{x : d\}_{k=q+1}^n \text{ st } \{e\}_i^s, \{\hat{e}_1 \succeq_{\mathbb{R}_+} \overline{e}_j\}_j^t$$

where $\overline{e}_j$ is equal to $\hat{e}_2$ where all occurrences of x_q have been replaced by $\hat{e}_j$. We require that $\Gamma; \{x : d\}_k^n \vdash \hat{e}_2 : \left\{\langle c, p, \nearrow, o\rangle_{ij}^{tu}, y\right\}$ and x_q may not occur in e, e_i, $\hat{e}_1$, and $\hat{e}_j$.

We prove that the transformation produces an equivalent formulation by noting that

$$\begin{aligned}
&\left[\!\left[\Gamma \vdash \min e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s, \hat{e}_1 \succeq_{\mathbb{R}_+} \hat{e}_2, \{x_q \succeq_{\mathbb{R}_+} \hat{e}_j\}_j^t\right]\!\right] \\
= &\left\{\begin{array}{l}
\min\limits_{y \in A \cap B} f(y) \\
\quad \text{where} \quad A = \prod\limits_{k=1}^{n} \mathcal{D}_k \\
\qquad\qquad B = \bigcap\limits_{i=1}^{s} \{y : g_i(y) \text{ R } h_i(y)\} \cap \\
\qquad\qquad\quad \{y : g(y) \succeq_{\mathbb{R}_+} h(y)\} \cap \\
\qquad\qquad\quad \bigcap\limits_{j=1}^{t} \{y : y_q \succeq_{\mathbb{R}_+} v_j(y)\}
\end{array}\right. \\
= &\left\{\begin{array}{l}
\min\limits_{x \in \hat{A} \cap \hat{B}} f(\pi_y(x)) \\
\quad \text{where} \quad \hat{A} = \left(\prod\limits_{k=1}^{q-1} \mathcal{D}_k\right) \times \left(\prod\limits_{k=q+1}^{n} \mathcal{D}_k\right) \\
\qquad\qquad \hat{B} = \bigcap\limits_{i=1}^{s} \{x : g_i(\pi_y(x)) \text{ R } h_i(\pi_y(x))\} \cap \\
\qquad\qquad\quad \bigcap\limits_{j=1}^{t} \{x : g(\pi_y(x)) \succeq_{\mathbb{R}_+} h(\pi_j(x))\}
\end{array}\right. \\
= &\left[\!\left[\Gamma \vdash \min e \text{ over } \{x : d\}_k^{q-1}, \{x : d\}_{k=q+1}^n \text{ st } \{e\}_i^s, \{\hat{e}_1 \succeq_{\mathbb{R}_+} \overline{e}_j\}_j^t\right]\!\right] \\
= &\left[\!\left[\mathcal{T}\left(\Gamma \vdash \min e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s, \hat{e}_1 \succeq_{\mathbb{R}_+} \hat{e}_2, \{x_q \succeq_{\mathbb{R}_+} \hat{e}_j\}_j^t\right)\right]\!\right]
\end{aligned}$$

as long as the following conditions are met. First, the only function that may depend on y_q must be h. We satisfy this condition since we stipulate that x_k must not occur in e, e_i, $\hat{e}_1$, and $\hat{e}_j$. Second, we require that the function h be monotonically increasing. We also satisfy this condition since our semantics are sound and we require that $\Gamma; \{x : d\}_k^n \vdash \hat{e}_2 : \left\{\langle c, p, \nearrow, o\rangle_{ij}^{tu}, y\right\}$. Therefore, our transformation is well defined.

As with expansion, we have three other cases that we must define. First, we state that

$$\mathcal{T}\left(\Gamma \vdash \mathsf{min}\ e\ \mathsf{over}\ \{x : d\}_k^n\ \mathsf{st}\ \{e\}_i^s, \hat{e}_1 \succeq_{\mathbb{R}_+} \hat{e}_2, \{\hat{e}_j \succeq_{\mathbb{R}_+} x_q]\}_j^t\right)$$
$$= \Gamma \vdash \mathsf{min}\ e\ \mathsf{over}\ \{x : d\}_k^{q-1}, \{x : d\}_{k=q+1}^n\ \mathsf{st}\ \{e\}_i^s, \{\hat{e}_1 \succeq_{\mathbb{R}_+} \overline{e}_j\}_j^t$$

when $\Gamma; \{x : d\}_k^n \vdash \hat{e}_2 : \left\{\langle c, p, \searrow, o\rangle_{ij}^{tu}, y\right\}$ and we meet the other formerly defined conditions. This is equivalent to the first transformation, but the function $\hat{e}_2$ is decreasing. Second, we define

$$\mathcal{T}\left(\Gamma \vdash \mathsf{min}\ e\ \mathsf{over}\ \{x : d\}_k^n\ \mathsf{st}\ \{e\}_i^s, \hat{e}_2 \succeq_{\mathbb{R}_+} \hat{e}_1, \{\hat{e}_j \succeq_{\mathbb{R}_+} x_q\}_j^t\right)$$
$$= \Gamma \vdash \mathsf{min}\ e\ \mathsf{over}\ \{x : d\}_k^{q-1}, \{x : d\}_{k=q+1}^n\ \mathsf{st}\ \{e\}_i^s, \{\overline{e}_j \succeq_{\mathbb{R}_+} \hat{e}_1\}_j^t$$

when $\Gamma; \{x : d\}_k^n \vdash \hat{e}_2 : \left\{\langle c, p, \nearrow, o\rangle_{ij}^{tu}, y\right\}$ and we meet the other formerly defined conditions. This is also equivalent to the first transformation, but we exchange the left and right hand

sides. Finally, we define that

$$\mathcal{T}\left(\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s, \hat{e}_2 \succeq_{\mathbb{R}_+} \hat{e}_1, \{x_q \succeq_{\mathbb{R}_+} \hat{e}_j\}_j^t\right)$$

$$= \Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^{q-1}, \{x : d\}_{k=q+1}^n \text{ st } \{e\}_i^s, \{\overline{e}_j \succeq_{\mathbb{R}_+} \hat{e}_1\}_j^t$$

when $\Gamma; \{x : d\}_k^n \vdash \hat{e}_2 : \left\{\langle c, p, \searrow, o\rangle_{ij}^{tu}, y\right\}$ and we meet the other formerly defined conditions. This is equivalent to the third transformation, but the function is decreasing. The proof that each of these transformations defines an equivalent problem follows similarly to the proof above.

6.5 Contracting a Single Auxiliary Variable and a Single Equality

We must consider one final contraction scenario. Ruminate over the problem

$$\begin{aligned} \min_{(x,y,z)\in\mathbb{R}^3} \quad & z \\ \text{st} \quad & z \geq f(x) \\ & x = g(y) \end{aligned}$$

We can transform this problem into

$$\begin{aligned} \min_{(y,z)\in\mathbb{R}^2} \quad & z \\ \text{st} \quad & z \geq f(g(y)) \end{aligned}$$

In fact, once we isolate a decision variable on one side of an equality, we can always eliminate that variable from the problem.

We summarize this process with the following rules. Consider a problem of the form

$$\begin{aligned} &\min_{y \in A \cap B} f(y) \\ &\text{where} \quad A = \prod_{k=1}^{n} \mathcal{D}_k \\ &\qquad\qquad B = \quad \{y : g_i(y) \text{ R } h_i(y)\}_i^s \cap \\ &\qquad\qquad\qquad\quad \{y : y_q = v(y)\} \end{aligned}$$

where we define q, R, and $\mathcal{D}_k$ as above. This problem has the same optimal value as

$$\begin{aligned} &\min_{x \in \hat{A} \cap \hat{B}} f(\pi_y(x)) \\ &\text{where} \quad \hat{A} = \left(\prod_{k=1}^{q-1} \mathcal{D}_k\right) \times \left(\prod_{k=q+1}^{n} \mathcal{D}_k\right) \\ &\qquad\qquad B = \quad \{y : g_i(\pi_y(x)) \text{ R } h_i(\pi_y(x))\}_i^s \end{aligned}$$

We require that $v(y) = v(z)$ for all $y, z \in A$ such that $y_k = z_k$ for all k save q. In other words, v may not depend on y_q.

We must prove this reformulation produces an equivalent problem. Since v does not depend on y_q, the constraint $y_q = v(y)$ defines the value of y_q in terms of the other decision variables. Thus, we may replace the dependency on y_q by $v(y)$. We accomplish this through the projection π_y.

We define this transformation formally with the following

$$\mathcal{T}\left(\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s, x_q = \hat{e}\right)$$

$$=\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^{q-1}, \{x : d\}_{k=q+1}^n \text{ st } \{\overline{e}\}_i^s$$

where we define $\overline{e_i}$ as e_i where we replace all occurrences of x_q by $\hat{e}$. We require that x_q does not occur in $\hat{e}$.

We prove this transformation produces an equivalent solution by observing that

$$\left[\!\left[\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s, x_q = \hat{e}\right]\!\right]$$

$$= \begin{cases} \min\limits_{y \in A \cap B} f(y) \\ \text{where} \quad A = \prod\limits_{k=1}^{n} \mathcal{D}_k \\ \qquad\quad B = \{y : g_i(y) \text{ R } h_i(y)\}_i^s \cap \\ \qquad\qquad \{y : y_q = v(y)\} \end{cases}$$

$$= \begin{cases} \min\limits_{x \in \hat{A} \cap \hat{B}} f(\pi_y(x)) \\ \text{where} \quad \hat{A} = \left(\prod\limits_{k=1}^{q-1} \mathcal{D}_k\right) \times \left(\prod\limits_{k=q+1}^{n} \mathcal{D}_k\right) \\ \qquad\quad B = \{y : g_i(\pi_y(x)) \text{ R } h_i(\pi_y(x))\}_i^s \end{cases}$$

$$= \left[\!\left[\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^{q-1}, \{x : d\}_{k=q+1}^n \text{ st } \{\overline{e}\}_i^s\right]\!\right]$$

$$= \left[\!\left[\mathcal{T}\left(\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s, x_q = \hat{e}\right)\right]\!\right]$$

as long as v does not depend on y_q. However, we satisfy this requirement since we stipulate

that y_q may not occur in $\hat{e}$. Thus, both problems produce the same optimal value.

We must define one additional, related transformation. We state that

$$\mathcal{T}\left(\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s, \hat{e} = x_q\right)$$

$$= \Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^{q-1}, \{x : d\}_{k=q+1}^n \text{ st } \{\overline{e}\}_i^s$$

This nearly identical transformation simply reverses the equality containing x_q. The correctness of this transformation follows in a manner similar to above.

6.6 Binary Maximum

We can expand an inequality containing the binary maximum function into two inequalities. The constraint $y \geq max(f(x), g(x))$ tells us that y must be bound below by both f and g. Thus, we can add each of these provisions as a separate constraint.

We define this transformation as

$$\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} max(e_2, e_3) : \diamond) = \Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_2 : \diamond,$$
$$\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_3 : \diamond$$

We prove the equivalence of both formulations by noting that

$$
\begin{aligned}
&[\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} max(e_2, e_3) : \diamond)]\!] \\
=&\{x : f(x) \succeq_{\mathbb{R}_+} max(g(x), h(x))\} \\
=&\{x : f(x) \succeq_{\mathbb{R}_+} g(x), f(x) \succeq_{\mathbb{R}_+} h(x)\} \\
=&\{x : f(x) \succeq_{\mathbb{R}_+} g(x)\} \cap \{x : f(x) \succeq_{\mathbb{R}_+} h(x)\} \\
=&[\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_2 : \diamond, \Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_3 : \diamond]\!] \\
=&[\![\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} max(e_2, e_3) : \diamond)]\!]
\end{aligned}
$$

6.7 Binary Minimum

We transform inequalities containing the binary minimum function analogously to inequalities containing the binary maximum. We characterize this transformation as

$$
\begin{aligned}
\mathcal{T}(\Gamma; \Sigma \vdash min(e_1, e_2) \succeq_{\mathbb{R}_+} e_3 : \diamond) = \quad & \Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_3 : \diamond, \\
& \Gamma; \Sigma \vdash e_2 \succeq_{\mathbb{R}_+} e_3 : \diamond
\end{aligned}
$$

We prove this produces an equivalent feasible region by noticing that

$$
\begin{aligned}
&[\![\Gamma; \Sigma \vdash min(e_1, e_2) \succeq_{\mathbb{R}_+} e_3 : \diamond)]\!] \\
=&\{x : min(f(x), g(x)) \succeq_{\mathbb{R}_+} h(x)\} \\
=&\{x : f(x) \succeq_{\mathbb{R}_+} h(x), g(x) \succeq_{\mathbb{R}_+} h(x)\} \\
=&\{x : f(x) \succeq_{\mathbb{R}_+} h(x)\} \cap \{x : g(x) \succeq_{\mathbb{R}_+} h(x)\} \\
=&[\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_3 : \diamond, \Gamma; \Sigma \vdash e_2 \succeq_{\mathbb{R}_+} e_3 : \diamond]\!] \\
=&[\![\mathcal{T}(\Gamma; \Sigma \vdash min(e_1, e_2) \succeq_{\mathbb{R}_+} e_3 : \diamond)]\!]
\end{aligned}
$$

6.8 Elementwise Maximum

We can transform an inequality containing the max function into several inequalities. In a manner similar to the binary maximum, the constraint $y \geq max(X)$ mandates that y be greater than every element of X. We add each of these requirements separately.

We characterize this transformation as

$$\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} max(e_2) : \diamond) = \{\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_{2ij} : \diamond\}_{ij}^{mn}$$

where $\Gamma; \Sigma \vdash e_2 : \left\{\langle c, p, m, o\rangle_{ij}^{mn}, y\right\}$ and e_{2ij} denotes the subindexing function $\cdot_{ij}$ applied to e_2 with arguments i and j. We prove this transformation produces an equivalent feasible

region by noticing that

$$\begin{aligned}
[\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} max(e_2) : \diamond]\!] &= \{x : f(x) \geq max(g(x))\} \\
&= \{x : f(x) \geq g_{ij}(x) \text{ for all } i, j\} \\
&= \bigcap_{ij} \{x : f(x) \geq g_{ij}(x)\} \\
&= \bigcap_{ij} [\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_{2ij} : \diamond]\!] \\
&= [\![\{\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_{2ij} : \diamond\}_{ij}^{mn}]\!] \\
&= [\![\mathcal{T}\,(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} max(e_2) : \diamond)]\!]
\end{aligned}$$

6.9 Elementwise Minimum

We transform an inequality containing the min function in a similar manner as the max function. We define that

$$\mathcal{T}\,(\Gamma; \Sigma \vdash min(e_1) \succeq_{\mathbb{R}_+} e_2 : \diamond) = \{\Gamma; \Sigma \vdash e_{1ij} \succeq_{\mathbb{R}_+} e_2 : \diamond\}_{ij}^{mn}$$

where $\Gamma; \Sigma \vdash e_2 : \left\{\langle c, p, m, o\rangle_{ij}^{mn}, y\right\}$. We prove this transformation behaves correctly by recognizing that

$$\begin{aligned}
[\![\Gamma; \Sigma \vdash min(e_1) \succeq_{\mathbb{R}_+} e_2 : \diamond]\!] &= \{x : min(f(x)) \geq g(x)\} \\
&= \{x : f_{ij}(x) \geq g(x) \text{ for all } i, j\} \\
&= \bigcap_{ij} \{x : f_{ij}(x) \geq g(x)\} \\
&= \bigcap_{ij} [\![\Gamma; \Sigma \vdash e_{1ij} \succeq_{\mathbb{R}_+} e_2 : \diamond]\!] \\
&= [\![\{\Gamma; \Sigma \vdash e_{1ij} \succeq_{\mathbb{R}_+} e_2 : \diamond\}_{ij}^{mn}]\!] \\
&= [\![\mathcal{T}\ (\Gamma; \Sigma \vdash min(e_1) \succeq_{\mathbb{R}_+} e_2 : \diamond)]\!]
\end{aligned}$$

6.10 Summation

Summations can be replaced by a series of addition and indexing operations. Sometimes this transformation is necessary to expose the individual elements of a matrix to further transformations.

Formally, we define that

$$\mathcal{T}\ (\Gamma; \Sigma \vdash sum(e) : \langle c, p, m, o\rangle) = \Gamma; \Sigma \vdash e_{11} + \cdots + e_{m1} + e_{12} + \cdots + e_{mn} : \langle c, p, m, o\rangle$$

where $\Gamma; \Sigma \vdash e : \{\langle c, p, m, o\rangle_{ij}^{mn}, y\}$. Notice, we transform a function rather than a constraint

with this transformation. We prove this produces an equivalent function by observing that

$$
\begin{aligned}
&[\![\Gamma; \Sigma \vdash sum(e) : \langle c, p, m, o\rangle]\!](x) \\
=& sum(f(x)) \\
=& \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij}(x) \\
=& f_{11}(x) + \cdots + f_{m1}(x) + f_{12}(x) + \cdots + f_{mn}(x) \\
=& [\![\Gamma; \Sigma \vdash e_{11} + \cdots + e_{m1} + e_{12} + \cdots + e_{mn} : \langle c, p, m, o\rangle]\!](x) \\
=& [\![\mathcal{T}(\Gamma; \Sigma \vdash sum(e) : \langle c, p, m, o\rangle)]\!](x)
\end{aligned}
$$

6.11 Infinity-Norm

Although some solvers can directly handle an inequality containing the infinity-norm, most solvers require that the constraint be reformulated into a system of linear inequalities. Our transformation system accomplishes this reformulation in two steps. First, we transform the constraint into a system of intermediate constraints that contain the absolute value function. Then, we may apply other transformations that linearize the result.

We define this transformation with the following rule

$$
\mathcal{T}(\Gamma; \Sigma \vdash e_0 \succeq_{\mathbb{R}_+} \|e\|_\infty : \diamond) = \{\Gamma; \Sigma \vdash e_0 \succeq_{\mathbb{R}_+} |e_{i1}| + \cdots + |e_{in}|\}_i^m
$$

We see this transformation produces the same feasible region by noting that

$$\begin{aligned}
[\![\Gamma; \Sigma \vdash e_0 \succeq_{\mathbb{R}_+} ||e||_\infty : \diamond]\!] &= \{x : f(x) \geq ||g(x)||_\infty\} \\
&= \left\{x : f(x) \geq \max_{1 \leq i \leq m} \sum_{j=1}^{n} |f_{ij}(x)|\right\} \\
&= \bigcap_{i=1}^{m} \left\{x : f(x) \geq \sum_{j=1}^{n} |f_{ij}(x)|\right\} \\
&= \bigcap_{i=1}^{m} \{x : f(x) \geq |f_{i1}(x)| + \cdots + |f_{in}(x)|\} \\
&= \left[\!\left[\{\Gamma; \Sigma \vdash e_0 \succeq_{\mathbb{R}_+} |e_{i1}| + \cdots + |e_{in}|\}_i^m\right]\!\right] \\
&= [\![\mathcal{T}\,(\Gamma; \Sigma \vdash e_0 \succeq_{\mathbb{R}_+} ||e||_\infty : \diamond)]\!]
\end{aligned}$$

6.12 One-Norm

Recall that for any $x \in \mathbb{R}^{m \times n}$, $||x||_1 = ||x^T||_\infty$. Thus, we can transform any use of the one-norm function into the infinity-norm. This allows us to utilize our above transformation.

We stipulate that

$$\mathcal{T}\,(\Gamma; \Sigma \vdash ||e||_1 : \langle c, p, m, o\rangle) = \Gamma; \Sigma \vdash ||e^T||_\infty : \langle c, p, m, o\rangle$$

We see that this produces an equivalent function since

$$\begin{aligned}
[\![\Gamma; \Sigma \vdash \|e\|_1 : \langle c, p, m, o\rangle]\!](x) &= \|f(x)\|_1 \\
&= \|f^T(x)\|_\infty \\
&= \left[\!\left[\Gamma; \Sigma \vdash \|e^T\|_\infty : \langle c, p, m, o\rangle\right]\!\right](x) \\
&= [\![\mathcal{T}(\Gamma; \Sigma \vdash \|e\|_1 : \langle c, p, m, o\rangle)]\!](x)
\end{aligned}$$

6.13 Two-Norm

Transforming a constraint containing the two-norm creates a number of difficulties. Unlike the one and infinity norms, we want to apply a different transformation when the argument is a vector rather than a matrix. In short, when the argument is a vector, we can transform the constraint into a second order cone constraint. When the argument is a matrix, we are forced to generate a semi-definite constraint. We consider these cases separately.

We define the transformation of a column vector as

$$\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \|e_2\|_2 : \diamond) = \Gamma; \Sigma \vdash \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \succeq_Q 0$$

when $\Gamma; \Sigma \vdash e_2 : \{\langle c, p, m, o\rangle_{ij}^{m1}, y\}$. We prove this transformation is well-formed by observ-

ing that

$$\begin{aligned}
[\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \|e_2\|_2 : \diamond]\!] &= \{x : f(x) \succeq_{\mathbb{R}_+} \|g(x)\|_2\} \\
&= \left\{ x : \begin{bmatrix} f(x) \\ g(x) \end{bmatrix} \succeq_Q 0 \right\} \\
&= \left[\!\!\left[\Gamma; \Sigma \vdash \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \succeq_Q 0 \right]\!\!\right] \\
&= [\![\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \|e_2\|_2 : \diamond)]\!]
\end{aligned}$$

In a similar manner, we define the transformation of row vector as

$$\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \|e_2\|_2 : \diamond) = \Gamma; \Sigma \vdash [e_1\ e_2] \succeq_Q 0$$

when $\Gamma; \Sigma \vdash e_2 : \{\langle c, p, m, o \rangle_{ij}^{1n}, y\}$. The proof that this transformation produces an equivalent feasible region is nearly identical to the proof above.

When the argument to the two-norm is a matrix, the result can be transformed into a semidefinite constraint. Before we begin, we need a related technical result. Let $X \in \mathbb{R}^{m \times n}$. We must show that the positive eigenvalues of the matrix

$$\bar{X} = \begin{bmatrix} 0 & X^T \\ X & 0 \end{bmatrix}$$

are the singular values of X. Let $v = [v_1^T, v_2^T]^T$ be an eigenvector of $\bar{X}$ with associated

eigenvalue λ. We see that

$$\begin{bmatrix} 0 & X^T \\ X & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \lambda \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\iff \begin{bmatrix} X^T v_2 \\ X v_1 \end{bmatrix} = \lambda \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\iff \begin{cases} X^T v_2 = \lambda v_1 \\ X v_1 = \lambda v_2 \end{cases}$$

$$\implies X^T X v_1 = \lambda^2 v_1$$

Thus, the absolute value of each eigenvalue of $\bar{X}$ is a singular value of X. This seems a bit puzzling since there are twice as many eigenvalues of $\bar{X}$ as there are singular values of X. However, we notice that

$$\begin{bmatrix} 0 & X^T \\ X & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ -v_2 \end{bmatrix} = \begin{bmatrix} -X^T v_2 \\ X v_1 \end{bmatrix} = \begin{bmatrix} -\lambda v_1 \\ \lambda v_2 \end{bmatrix} = -\lambda \begin{bmatrix} v_1 \\ -v_2 \end{bmatrix}$$

Thus, whenever $v = [v_1^T,\ v_2^T]^T$ is an eigenvector with associated eigenvalue λ, so is $[v_1^T,\ -v_2^T]^T$ with associated eigenvalue $-\lambda$. Therefore, we can find the singular values of X by finding the nonnegative eigenvalues of $\bar{X}$. This allows us to view a constraint on the two-norm of

X as a constraint on the largest eigenvalue of $\overline{X}$. This gives us the following transformation

$$\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \|e_2\|_2 : \diamond) = \Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \lambda_{max} \begin{bmatrix} 0 & e_2^T \\ & 0 \end{bmatrix} : \diamond$$

We see this transformation produces an equivalent feasible region by noting that

$$\begin{aligned}
[\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \|e_2\|_2 : \diamond]\!] &= \{x : f(x) \succeq_{\mathbb{R}_+} \|g(x)\|_2\} \\
&= \{x : f(x) \succeq_{\mathbb{R}_+} \sigma_{max} g(x) \\
&= \{x : f(x) \succeq_{\mathbb{R}_+} \lambda_{max} \begin{bmatrix} 0 & g^T(x) \\ g(x) & 0 \end{bmatrix} \\
&= \left[\!\!\left[\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \lambda_{max} \begin{bmatrix} 0 & e_2^T \\ & 0 \end{bmatrix} : \diamond \right]\!\!\right] \\
&= [\![\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \|e_2\|_2 : \diamond)]\!]
\end{aligned}$$

6.14 Maximum Eigenvalue

A constraint on the maximum eigenvalue of a matrix may be reformulated into a semidefinite constraint. Let t be a scalar variable and A a matrix. Then, the constraint $tI \succeq_{\mathcal{S}_+} A$ requires that t be larger than the maximum eigenvalue of A. Recall, adding a multiple of the identity to a matrix simply shifts the eigenvalues of that matrix. Thus, the matrix $tI - A$ possesses a set of eigenvalues $\{t - \lambda_{max}, \ldots, t - \lambda_{min}\}$. Since the constraint $tI - A \succeq_{\mathcal{S}_+} 0$ requires that all eigenvalues to be nonnegative, it equivalently states that $t - \lambda_i \geq 0$ or $t \geq \lambda_i$

for all i. Of course, this means that t must be at least as large as the largest eigenvalue.

We formulate this transformation as

$$\mathcal{T}\left(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \lambda_{max}(e_2) : \diamond\right) = \Gamma; \Sigma \vdash e_1 * I \succeq_{\mathcal{S}_+} e_2 : \diamond$$

where I is the identity matrix with size the same as e_2. We see that this transformation behaves correctly by noticing that

$$\begin{aligned}
[\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \lambda_{max}(e_2) : \diamond]\!] &= \{x : f(x) \geq \lambda_{max}(g(x))\} \\
&= \{x : f(x) I \succeq_{\mathcal{S}_+} g(x)\} \\
&= [\![\Gamma; \Sigma \vdash e_1 * I \succeq_{\mathcal{S}_+} e_2 : \diamond]\!] \\
&= [\![\mathcal{T}\left(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} \lambda_{max}(e_2) : \diamond\right)]\!]
\end{aligned}$$

6.15 Minimum Eigenvalue

We transform a constraint containing the minimum eigenvalue in a similar manner as the maximum eigenvalue. For a scalar variable t and a matrix A, the constraint $A \succeq_{\mathcal{S}_+} tI$ requires that t be at most the minimum eigenvalue of A. The matrix $A - tI$ has eigenvalues $\{\lambda_{max} - t, \ldots, \lambda_{min} - t\}$. Since the constraint $A - tI \succeq_{\mathcal{S}_+} 0$ requires that each of these eigenvalues be positive, it equivalently requires that t be no larger than the minimum.

We define this transformation as

$$\mathcal{T}\left(\Gamma; \Sigma \vdash \lambda_{min}(e_1) \succeq_{\mathbb{R}_+} e_2 : \diamond\right) = \Gamma; \Sigma \vdash e_1 \succeq_{\mathcal{S}_+} e_2 * I : \diamond$$

where I is the identity matrix with size compatible to e_1. We see this transformation produces an equivalent feasible region by observing that

$$\begin{aligned}
[\![\Gamma; \Sigma \vdash \lambda_{min}(e_1) \succeq_{\mathbb{R}_+} e_2 : \diamond]\!] &= \{x : \lambda_{min}(f(x)) \geq g(x)\} \\
&= \{x : f(x) \succeq_{\mathbb{R}_+} g(x) * I\} \\
&= [\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathcal{S}_+} e_2 * I : \diamond]\!] \\
&= [\![\mathcal{T}\left(\Gamma; \Sigma \vdash \lambda_{min}(e_1) \succeq_{\mathbb{R}_+} e_2 : \diamond\right)]\!]
\end{aligned}$$

6.16 Convex Quadratics

A constraint containing a convex quadratic on each side of an inequality may be transformed into a second-order cone constraint. Notice the following

$$\left\{ x : \begin{bmatrix} 1 - c^T x - \gamma \\ 2Ux \\ 1 + c^T x + \gamma \end{bmatrix} \succeq_Q 0 \right\}$$

$$= \left\{ x : 1 - c^T x - \gamma \geq \sqrt{(2Ux)^T (2Ux) + (1 + c^T x + \gamma)^2} \right\}$$

$$= \left\{ x : (1 - c^T x - \gamma)^2 \geq 4x^T U^T Ux + (1 + c^T x + \gamma)^2, 1 - c^T x - \gamma \geq 0 \right\}$$

$$= \left\{ x : 0 \geq 4x^T U^T Ux + (1 + c^T x + \gamma)^2 - (1 - c^T x - \gamma)^2, 1 - c^T x - \gamma \geq 0 \right\}$$

$$= \left\{ x : 0 \geq 4x^T U^T Ux + 4c^T x + 4\gamma, 1 - c^T x - \gamma \geq 0 \right\}$$

$$= \left\{ x : 0 \geq x^T U^T Ux + c^T x + \gamma, 1 - c^T x - \gamma \geq 0 \right\}$$

$$= \left\{ x : 0 \geq x^T U^T Ux + c^T x + \gamma \right\}$$

since $(a+b+c)^2 - (a-b-c)^2 = 4ab + 4ac$ and by observing that $1 - c^T x - \gamma \geq -x^T U^T Ux - c^T x - \gamma \geq 0$ since $U^T U \succeq 0$. Thus, when we have a constraint of the form

$$x^T Ax + a^T x + \alpha \geq x^T Bx + b^T x + \beta$$

and $U^T U = C = B - A \succeq 0$, $c = b - a$, and $\gamma = \beta - \alpha$, we can use the above transformation.

Unfortunately, our situation becomes slightly more complicated. Although we can de-

termine when a function is quadratic, we define this property in terms of all decision variables. For example, consider the function

$$X \in \mathbb{R}^{2\times 2}, y \in \mathbb{R}^{2\times 1} \mapsto (X_{11} + X_{21})^2 + 3(y_1 + y_2)^2$$

Our type system tells us that this function is quadratic with a representation

$$X \in \mathbb{R}^{2\times 2}, y \in \mathbb{R}^{2\times 1} \mapsto \begin{bmatrix} vec(X) \\ y \end{bmatrix}^T \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 & 3 & 3 \end{bmatrix} \begin{bmatrix} vec(X) \\ y \end{bmatrix}$$

where we denote the quadratic coefficient by C. Unfortunately, this representation becomes burdensome since it depends on the variables X_{12} and X_{22} even though the original representation did not. In order to resolve this problem, we define a new projection operator, $\pi_C : \mathbb{R}^m \to \mathbb{R}^n$ defined as

$$[\pi_C(x)]_k = x_{\xi(k)}$$

where m denotes the total number of rows in C and n denotes the number of nonzero rows. Then, label the nonzero rows of C, in order, from 1 to n. The function $\xi : \{1, \ldots, m\} \to \{1, \ldots, n\}$ represents an increasing, injective function that maps this labeling to the original

index in C. When we compose this projection with vectorization, we abbreviate the result as $vec_C = \pi_C \circ vec$. Using these projections, we define our quadratic as

$$X \in \mathbb{R}^{2\times 2}, y \in \mathbb{R}^{2\times 1} \mapsto \begin{bmatrix} vec_C(X) \\ vec_C(y) \end{bmatrix}^T \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 3 & 3 \end{bmatrix} \begin{bmatrix} vec_C(X) \\ vec_C(y) \end{bmatrix}$$

Where we call the smaller quadratic coefficient the *reduced* quadratic coefficient. At this point, we must factor this coefficient into $U^T U$. Unfortunately, since this matrix may be only positive semidefinite, we can not take the Choleski factorization. Instead, we find the compact singular value decomposition. the second-order coefficient into $V\Sigma V^T$ where

$$V = \begin{bmatrix} 0 & -1/\sqrt{2} \\ 0 & -1/\sqrt{2} \\ -1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 0 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 6 & 0 \\ 0 & 2 \end{bmatrix}$$

Thus, our factored, reduced quadratic coefficient is

$$U = \begin{bmatrix} 0 & 0 & -\sqrt{3} & -\sqrt{3} \\ -1 & -1 & 0 & 0 \end{bmatrix}$$

We define our factored, reduced quadratic as

$$X \in \mathbb{R}^{2\times 2}, y \in \mathbb{R}^{2\times 1} \mapsto \begin{bmatrix} X_{11} \\ X_{21} \\ y_1 \\ y_2 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & -\sqrt{3} & -\sqrt{3} \\ -1 & -1 & 0 & 0 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & -\sqrt{3} & -\sqrt{3} \\ -1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} X_{11} \\ X_{21} \\ y_1 \\ y_2 \end{bmatrix}^T$$

With this form, we can easily reformulate our quadratic into a second-order cone constraint.

We define this process formally with the following rule. Let $\Gamma; \Sigma \vdash e_1 : \langle \perp, (\alpha, a, A), \perp, \mathbb{R} \rangle$ and $\Gamma; \Sigma \vdash e_2 : \langle \perp, (\beta, b, B), \perp, \mathbb{R} \rangle$. Assume that $B - A \succeq_{\mathcal{S}_+} 0$. Then, we define that

$$\mathcal{T}\left(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_2 : \diamond\right) = \Gamma; \Sigma \vdash \begin{bmatrix} 1 - \hat{c}^T * \hat{e} - \gamma \\ 2 * U * \tilde{e} \\ 1 + \hat{c}^T * \hat{e} + \gamma \end{bmatrix} \succeq_Q 0 : \diamond$$

In addition, we define the following. Let the $\gamma = \beta - \alpha$, $c = b - a$, and $C = B - A$. Next, let $\hat{c} = vec_c(c)$ be a constant vector containing the nonzero elements of c. Associated with this vector, let $\hat{e}$ be the vertical concatenation of each variable that corresponds to a nonzero element of c. Similarly, let U be a constant matrix that corresponds to the factored, reduced

quadratic coefficient of C. Along with this matrix, let $\tilde{e}$ be the vertical concatenation of each variable that corresponds to a nonzero rows of C.

We prove this transformation behaves properly with the following. Notice that

$$
\begin{aligned}
&[\![\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_2 : \diamond]\!] \\
=&\{x : vec(x)^T A vec(x) + a^T vec(x) + \alpha \succeq_{\mathbb{R}_+} vec(x)^T B vec(x) + b^T vec(x) + \beta\} \\
=&\{x : 0 \geq vec(x)^T C vec(x) + c^T vec(x) + \gamma\} \\
=&\{x : 0 \geq vec_C(x)^T U^T U vec_C(x) + vec_c(c)^T vec_c(x) + \gamma\} \\
=&\left\{x : \begin{bmatrix} 1 - vec_c(c)^T vec_c(x) - \gamma \\ 2U vec_C(x) \\ 1 + vec_c(c)^T vec_c(x) + \gamma \end{bmatrix} \succeq_Q 0\right\} \\
=&\left\{x : \begin{bmatrix} 1 - \hat{c}^T vec_c(x) - \gamma \\ 2U vec_C(x) \\ 1 + \hat{c}^T vec_c(x) + \gamma \end{bmatrix} \succeq_Q 0\right\} \\
=&\left[\!\!\left[\Gamma; \Sigma \vdash \begin{bmatrix} 1 - \hat{c}^T * \hat{e} - \gamma \\ 2 * U * \tilde{e} \\ 1 + \hat{c}^T * \hat{e} + \gamma \end{bmatrix} \succeq_Q 0 : \diamond\right]\!\!\right] \\
=&[\![\mathcal{T}(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_2 : \diamond)]\!]
\end{aligned}
$$

6.17 Non-Convex Quadratics

We transform nonconvex quadratics in a much different fashion than convex quadratics. These constraints are difficult to handle since they can define a convoluted and possibly nonconnected feasible region. For example, the constraint $x^2 = 1$ defines an integer constraint where the variable x must be either 1 or -1. In order to reformulate these constraints, we lift each quadratic into a higher-dimensional space where the quadratic becomes linear. In other words, consider a constraint of the form

$$f(x) \text{ R } x^T Ax + a^T x + \alpha$$

Notice that we can rewrite this as

$$f(x) \text{ R } tr(x^T Ax) + a^T x + \alpha$$

since $x^T Ax$ is scalar. Using the commutative property of trace, we see that

$$f(x) \text{ R } tr(Axx^T) + a^T x + \alpha$$

At this point, we linearize the problem into

$$f(x) \text{ R } tr(AX) + a^T x + \alpha$$

where $X = xx^T$. As a consequence of this action, we add a difficult rank-1 constraint, $X = xx^T$. Later, we relax this constraint into a semidefinite program.

We define this transformation with the following. Let $\Gamma; \Sigma \vdash e_2 : \langle \perp, (\alpha, a, A), \perp, \perp, \mathbb{R} \rangle$. Then, we specify that

$$\mathcal{T}(\Gamma; \Sigma \vdash e_1 \text{ R } e_2 : \diamond) = \quad \Gamma; \Sigma, X : \mathcal{S}^m \vdash e_1 \text{ R } tr(\hat{A} * X) + \hat{a}^T * \hat{e} + \alpha : \diamond,$$
$$\Gamma; \Sigma, X : \mathcal{S}^m \vdash X = x * x^T : \diamond$$

In addition, we make the following definitions. We define $\hat{A}$ to be the reduced quadratic coefficient of size m and $\hat{a}$ to be the reduced linear coefficient. In addition, $\hat{e}$ represents the vertical concatenation of each variable that corresponds to a nonzero element of a.

Since we add an auxiliary variable, we must show that the original feasible region is a projection of the reformulated region. Let $\pi_x : (\prod_{k=1}^n \mathcal{D}_k) \times \mathcal{S}^m \to \prod_{k=1}^n \mathcal{D}_k$ be a projection defined by

$$[\pi_x(y)]_k = y_k$$

for $k = 1, \ldots, n$. Then, we see that

$$
\begin{aligned}
[\![\Gamma; \Sigma \vdash e_1 \text{ R } e_2 : \diamond]\!] =& \{x : f(x) \text{ R } vec(x)^T A vec(x) + a^T vec(x) + \alpha\} \\
=& \pi_x\{x, X : f(\pi_x(x, X)) \text{ R } vec(x)^T A vec(x) + a^T vec(x) + \alpha\} \\
=& \pi_x\{x, X : f(\pi_x(x, X)) \text{ R } vec_A(x)^T \hat{A} vec_A(x) + \hat{a}^T vec_a(x) + \alpha\} \\
=& \pi_x\{x, X : f(\pi_x(x, X)) \text{ R } tr(\hat{A}X) + \hat{a}^T vec_a(x) + \alpha, X = vec_A(x) vec_A(x)^T\} \\
=& \pi_x \left(\begin{array}{l} \{x, X : f(\pi_x(x, X)) \text{ R } tr(\hat{A}X) + \hat{a}^T vec_a(x) + \alpha\} \cap \\ \\ \{x, X : X = vec_A(x) vec_A(x)^T\} \end{array} \right) \\
=& \pi_x \left[\!\!\left[\begin{array}{l} \Gamma; \Sigma, X : \mathcal{S}^m \vdash e_1 \text{ R } tr(\hat{A} * X) + \hat{a}^T * \hat{e} + \alpha : \diamond, \\ \\ \Gamma; \Sigma, X : \mathcal{S}^m \vdash X = x * x^T : \diamond \end{array} \right]\!\!\right] \\
=& \pi_x [\![\mathcal{T}(\Gamma; \Sigma \vdash e_1 \text{ R } e_2 : \diamond)]\!]
\end{aligned}
$$

In a similar manner, we can define one additional transformation by reversing the left and right hand sides. Since equality is symmetric, this must generate an equivalent problem.

As a final note, this transformation generally makes a problem more difficult to solve. However, we can relax the rank-1 constraint into a much easier form using the next transformation.

6.18 Symmetric Rank-1 Constraints

Symmetric rank-1 constraints represent an extremely difficult class of constraints to optimize over. Fortunately, we can relax them into a semidefinite constraint. The strength

of this relaxation depends on many factors. In fact, this relaxation could be extremely weak. Nevertheless, it has proved useful in many cases.

Our strategy consists of relaxing a constraint of the form $X = xx^T$, where $X = X^T$, into $X \succeq_{\mathcal{S}_+} xx^T$. The Schur Complement theorem tells us that

$$A \succ_{\mathcal{S}_{++}} 0 \text{ and } C \succeq_{\mathcal{S}_+} BA^{-1}B^T \iff \begin{bmatrix} A & B^T \\ B & C \end{bmatrix} \succeq_{\mathcal{S}_+} 0$$

where $A \succ_{\mathcal{S}_+} 0$ denotes that A must be positive definite. Therefore, we can relax our original constraint into

$$\begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succeq_{\mathcal{S}_+} 0$$

We codify this process with the following rule

$$\mathcal{T}\left(\Gamma; \Sigma \vdash e_1 = e_2 * e_2^T : \diamond\right) = \Gamma; \Sigma \vdash \begin{bmatrix} I & e_2^T \\ & e_1 \end{bmatrix} \succeq_{\mathcal{S}_+} 0 : \diamond$$

when $\Gamma; \Sigma \vdash e_1 : \left\{\langle c, p, m, o\rangle_{ij}^{mn}, \ddagger\right\}$ and I denotes the identity matrix. Although this process does not generate an equivalent feasible region, we see that it behaves correctly by

observing that

$$\begin{aligned}
\left[\!\left[\Gamma; \Sigma \vdash e_1 = e_2 * e_2^T : \diamond\right]\!\right] &= \left\{x : f(x) = g(x)g(x)^T\right\} \\
&\subseteq \left\{x : f(x) \succeq_{\mathcal{S}_+} g(x)g(x)^T\right\} \\
&= \left\{x : \begin{bmatrix} I & g(x)^T \\ g(x) & f(x) \end{bmatrix} \succeq_{\mathcal{S}_+} 0\right\} \\
&= \left[\!\left[\Gamma; \Sigma \vdash \begin{bmatrix} I & e_2^T \\ & e_1 \end{bmatrix} \succeq_{\mathcal{S}_+} 0 : \diamond\right]\!\right] \\
&= \left[\!\left[\mathcal{T}\left(\Gamma; \Sigma \vdash e_1 = e_2 * e_2^T : \diamond\right)\right]\!\right]
\end{aligned}$$

As a final note, since equal is symmetric, we can apply this same transformation by reversing the left and right hand sides.

6.19 ± 1 Integer Variables

We find plus-minus one integer domains in a number of graph theoretic problems. For example, the max-cut problem uses this domain to divide the vertices of a graph into two sets. Instead of handling these variables directly, we can transform them into a non-convex quadratic equality constraint, $x^2 = 1$. Once we obtain this form, we can relax it into a semidefinite program.

We define this transformation with the following rule

$$\mathcal{T}\left(\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^{q-1}, x_q : \{-1,1\}^{m_q \times n_q}, \{x : d\}_{k=q+1}^{n} \text{ st } \{e\}_i^s\right)$$

$$= \Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^{q-1}, x_q : \mathbb{R}^{m_q \times n_q}, \{x : d\}_{k=q+1}^{n} \text{ st } \{e\}_i^s, \{x_{qij}^2 = 1\}_{ij}^{m_q n_q}$$

We see that this transformation does not change the solution by observing that

$$\left[\!\left[\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^{q-1}, x_q : \{-1,1\}^{m_q \times n_q}, \{x : d\}_{k=q+1}^{n} \text{ st } \{e\}_i^s\right]\!\right]$$

$$= \begin{cases} \min\limits_{x \in A \cap B} f(x) \\ \text{where} \quad A = \left\{\prod\limits_{k=1}^{q-1} \mathcal{D}_k\right\} \times \{-1,1\}^{m_q \times n_q} \times \left\{\prod\limits_{k=q+1}^{n} \mathcal{D}_k\right\} \\ \qquad\quad B = \{x \in S_i\}_i^s \end{cases}$$

$$= \begin{cases} \min\limits_{x \in A \cap B} f(x) \\ \text{where} \quad A = \left\{\prod\limits_{k=1}^{q-1} \mathcal{D}_k\right\} \times \mathbb{R}^{m_q \times n_q} \times \left\{\prod\limits_{k=q+1}^{n} \mathcal{D}_k\right\} \\ \qquad\quad B = \{x \in S_i\}_i^s \cap \\ \qquad\qquad \left\{x_{qij}^2 = 1\right\}_{ij}^{m_q n_q} \end{cases}$$

$$= \left[\!\left[\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^{q-1}, x_q : \mathbb{R}^{m_q \times n_q}, \{x : d\}_{k=q+1}^{n} \text{ st } \{e\}_i^s, \{x_{qij}^2 = 1\}_{ij}^{m_q n_q}\right]\!\right]$$

$$= \left[\!\left[\mathcal{T}\left(\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^{q-1}, x_q : \{-1,1\}^{m_q \times n_q}, \{x : d\}_{k=q+1}^{n} \text{ st } \{e\}_i^s\right)\right]\!\right]$$

where S_i denotes some arbitrary feasible region.

6.20 Quartics

Earlier, we observed the use of quartics in the chained singular function. In that example, we transformed each quartic into a quadratic. We revisit that trick with the following example. Consider the constraint,

$$y \geq x^4$$

We can rewrite the term x^4 as $(x^2)^2$. Then, we notice two facts. First, $x^2 \geq 0$ for all x. Second, the function $x \mapsto x^2$ is increasing for all positive arguments. Therefore, we can introduce a single auxiliary variable and rewrite this constraint as the pair

$$y \geq z^2$$

$$z \geq x^2$$

In order to formally show this equivalence, let us define two sets

$$A = \left\{ x \in \prod_{k=1}^{n} \mathcal{D}_k : f(x) \geq (g(x))^4 \right\}$$
$$B = \left\{ x \in \prod_{k=1}^{n} \mathcal{D}_k, y \in \mathbb{R} : f(\pi_x(x,y)) \geq y^2, y \geq (g(\pi_x(x,y)))^2 \right\}$$

where $\pi_x : \prod_{k=1}^{n} \mathcal{D}_k \times \mathbb{R} \to \prod_{k=1}^{n} \mathcal{D}_k$ and $[\pi_x(z)]_k = z_k$ for $k = 1, \ldots, n$. We must show that $A = \pi_x B$. In the forward direction, let $x \in A$. We must find a y such that $(x, y) \in B$. Let $y = (g(\pi_x(x,y)))^2$. Certainly, we see that $y = (g(\pi_x(x,y)))^2 \geq (g(\pi_x(x,y)))^2$. In addition,

since $x \in A$, we see that

$$f(\pi(x,y)) = f(x) \geq (g(x))^4 = (g(\pi(x,y)))^4 = y^2 \geq y^2$$

Therefore, $(x,y) \in B$ and $A \subseteq \pi_x B$. In the reverse direction, take $(x,y) \in B$. We must show that $x \in A$. Since $x, y \mapsto y^2$ is increasing for positive y and $(g(\pi_x(x,y)))^2 \geq 0$, we see that

$$f(x) = f(\pi_x(x,y)) \geq y^2 \geq ((g(\pi_x(x,y)))^2)^2 = (g(x))^4$$

Thus, $x \in A$ and $\pi_x B \subseteq A$. Hence, we know that $A = \pi_x B$.

We define this transformation as

$$\begin{aligned}\mathcal{T}\left(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_2^4 : \diamond\right) = \quad & \Gamma; \Sigma, y : \mathbb{R}^{1\times 1} \vdash e_1 \geq y^2 : \diamond, \\ & \Gamma; \Sigma, y : \mathbb{R}^{1\times 1} \vdash y \geq e_2^2 : \diamond\end{aligned}$$

We see this transformation behaves properly by noticing that

$$\begin{aligned}\left[\!\left[\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_2^4 : \diamond\right]\!\right] &= \left\{x \in \textstyle\prod_{k=1}^n \mathcal{D}_k : f(x) \geq (g(x))^4\right\} \\ &= \pi_x \left\{x \in \textstyle\prod_{k=1}^n \mathcal{D}_k, y \in \mathbb{R} : f(\pi_x(x,y)) \geq y^2, y \geq (g(\pi_x(x,y)))^2\right\} \\ &= \pi_x \left\{x \in \textstyle\prod_{k=1}^n \mathcal{D}_k, y \in \mathbb{R} : f(\pi_x(x,y)) \geq y^2\right\} \cap \\ &\qquad \left\{x \in \textstyle\prod_{k=1}^n \mathcal{D}_k, y \in \mathbb{R} : y \geq (g(\pi_x(x,y)))^2\right\} \\ &= \pi_x \left[\!\!\left[\begin{array}{l} \Gamma; \Sigma, y : \mathbb{R}^{1\times 1} \vdash e_1 \geq y^2 : \diamond, \\ \Gamma; \Sigma, y : \mathbb{R}^{1\times 1} \vdash y \geq e_2^2 : \diamond \end{array} \right]\!\!\right] \\ &= \pi_x \left[\!\left[\mathcal{T}\left(\Gamma; \Sigma \vdash e_1 \succeq_{\mathbb{R}_+} e_2^4 : \diamond\right)\right]\!\right]\end{aligned}$$

6.21 Linearizing the Objective Function

The vast majority of the transformations that we have defined operate on a constraint rather than on the objective function. Although many of these transformations will work in both situations, defining each case becomes cumbersome. Instead, we note that we can convert the objective function into a constraint by adding a single variable. In other words, we can convert the problem

$$\min_{x \in A} f(x)$$

into

$$\min_{x \in A, y \in \mathbb{R}} y \quad \text{st} \quad y \geq f(x)$$

We must show these problems produce an equivalent solution. Let x^* be an optimal solution to the first problem. Certainly, when $y = f(x^*)$, the point $(f(x^*), y)$ is feasible in the second problem and the objective value remains the same. Therefore, the optimal value of the first problem is greater than or equal to the second. In the reverse direction, let (x^*, y^*) be an optimal solution to the second problem. We immediately see that x^* remains feasible in the first problem. Since $y^* \geq f(x^*)$, the optimal value of the second problem is greater than or equal to the first. Thus, both problems have the same optimal value.

We define this transformation as

$$\mathcal{T}\left(\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s\right)$$

$$= \Gamma \vdash \text{min } x_{n+1} \text{ over } \{x : d\}_k^n, x_{n+1} : \mathbb{R}^{1\times 1} \text{ st } \{e\}_i^s, x_{n+1} \succeq_{\mathbb{R}_+} e$$

We see that this produces an equivalent problem since

$$
\begin{aligned}
&[\![\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s]\!] \\
&= \begin{cases} \min\limits_{x \in A \cap B} f(x) \\ \text{where} \quad A = \prod\limits_{k=1}^{n} \mathcal{D}_k \\ \qquad\quad\; B = \quad \{x \in S_i\}_i^s \end{cases} \\
&= \begin{cases} \min\limits_{y \in A \cap B} y_{k+1} \\ \text{where} \quad A = \left\{ \prod\limits_{k=1}^{n} \mathcal{D}_k \right\} \times \mathbb{R} \\ \qquad\quad\; B = \quad \{y \in S_i\}_i^s \cap \{y : y_{k+1} \geq f(\pi_x(y))\} \end{cases} \\
&= \left[\!\!\left[\Gamma \vdash \text{min } x_{n+1} \text{ over } \{x : d\}_k^n, x_{n+1} : \mathbb{R}^{1\times 1} \text{ st } \{e\}_i^s, x_{n+1} \succeq_{\mathbb{R}_+} e\right]\!\!\right] \\
&= [\![\mathcal{T}\,(\Gamma \vdash \text{min } e \text{ over } \{x : d\}_k^n \text{ st } \{e\}_i^s)]\!]
\end{aligned}
$$

where $\pi_x : \left(\prod_{k=1}^{n} \mathcal{D}_k\right) \times \mathbb{R} \to \prod_{k=1}^{n} \mathcal{D}_k$ is defined as $\pi_x(y)_k = y_k$ for $k = 1, \ldots, n$.

Chapter 7

Case Studies

In the following section, we revisit the chained singular function and the max-cut problem. We show that we can correctly model and transform these problems into more desirable formulations. In addition, we assess the type-checking capability our design by demonstrating that we can prove convexity, polynomality, and monotonicity in a wide variety of situations.

We have implemented the language as a series of quotations within OCaml using Camlp5. Essentially, a quotation allows us to generate a piece of the abstract syntax tree using our custom defined grammar. Once we have the abstract syntax tree, we can type-check and manipulate the expression. Using quotations has two benefits. First, it allows us to quickly generate and manipulate problems. Second, we avoid adding file support, bindings, and other features within our language. Simply, we leverage OCaml for these features.

7.1 Max-Cut

Recall, we formulate the max-cut problem as

$$\max_{x\in\{-1,1\}^n} \frac{1}{4}\sum_{ij}(1 - x_i x_j)w_{ij}$$

Let us define a small version of this problem in our language with the following code

```
let mc= <:mp<
    max 2.0*(1.-x*y)+1.0*(1.-y*z)+3.0*(1.-x*z)
    over x in PlusMinusOne[1,1],
         y in PlusMinusOne[1,1],
         z in PlusMinusOne[1,1] >>;;
```

Before we can transform the objective, we must linearize the problem and move the objective function into the constraints.

```
let mc=linearize mc;;
```

This generates the following problem

```
  max     _0
  over    x in PlusMinusOne[ 1,1 ], y in PlusMinusOne[ 1,1 ],
          z in PlusMinusOne[ 1,1 ], _0 in Real[ 1,1 ]
  st      [2.] * ([1.] - x * y) + [1.] * ([1.] - y * z)
              + [3.] * ([1.] - x * z) >= _0
```

The variable _0 denotes a new auxiliary variable. All auxiliary variables start with an underscore followed by a number. We use this convention since it insures that these names do not conflict with the user-defined variables. This program corresponds directly to the problem

$$\begin{aligned} \max_{(x,y,z)\in\{-1,1\}^3, t_0\in\mathbb{R}} \quad & t_0 \\ \text{st} \quad & 2(1-xy)+(1-yz)+3(1-xz) \geq t_0 \end{aligned}$$

Next, we eliminate each of the plus-minus one integer constraints.

```
let mc=napply_mp pm1 mc;;
```

The function napply_mp repeatedly applies a transformation until normalization. In other words, it applies the transformation until the transformation has no affect. This generates the problem

```
max     _0
over    x in Real[ 1,1 ], y in Real[ 1,1 ], z in Real[ 1,1 ],
        _0 in Real[ 1,1 ]
st      [2.] * ([1.] - x * y) + [1.] * ([1.] - y * z)
            + [3.] * ([1.] - x * z) >= _0
        (x[ 1 , 1 ])^([2.]) = [1.]
        (y[ 1 , 1 ])^([2.]) = [1.]
        (z[ 1 , 1 ])^([2.]) = [1.]
```

This corresponds directly to the mathematical program

$$\begin{aligned} \max_{(x,y,z,t_0)\in\mathbb{R}^4} \quad & t_0 \\ \text{st} \quad & 2(1-xy)+(1-yz)+3(1-xz) \geq t_1 \\ & x^2 = 1 \\ & y^2 = 1 \\ & z^2 = 1 \end{aligned}$$

Next, we lift each nonconvex quadratic into a linear constraint with the command

```
let mc=apply_mp (nonconvex_quad 'Left) mc;;
```

The function `apply_mp` applies a transformation to each constraint. This produces the problem

```
max     _0
over    x in Real[ 1,1 ], y in Real[ 1,1 ], z in Real[ 1,1 ],
        _0 in Real[ 1,1 ], _1 in Symmetric[ 3 ],
        _2 in Symmetric[ 1 ], _3 in Symmetric[ 1 ],
        _4 in Symmetric[ 1 ]
st      trace([0. -1. -1.5; -1. 0. -0.5; -1.5 -0.5 0.] * _1) +
            [6.] >= _0
        _1 = {x[ 1 , 1 ]; {y[ 1 , 1 ]; z[ 1 , 1 ]}} *
            ({x[ 1 , 1 ]; {y[ 1 , 1 ]; z[ 1 , 1 ]}})'
        trace([1.] * _2) = [1.]
```

```
_2 = x[ 1 , 1 ] * (x[ 1 , 1 ])'

trace([1.] * _3) = [1.]

_3 = y[ 1 , 1 ] * (y[ 1 , 1 ])'

trace([1.] * _4) = [1.]

_4 = z[ 1 , 1 ] * (z[ 1 , 1 ])'
```

This program corresponds directly to the problem

$$
\begin{aligned}
\max_{(x,y,z,t_0,t_2,t_3,t_4)\in\mathbb{R}^7,t_1\in\mathcal{S}^3} \quad & t_0 \\
\text{st} \quad & \operatorname{tr}\left(\begin{bmatrix} 0 & -1 & -1.5 \\ -1 & 0 & -.5 \\ -1.5 & -.5 & 0 \end{bmatrix} t_1\right) + 6 \geq 0 \\
& t_1 = \begin{bmatrix} x & y & z \end{bmatrix}^T \begin{bmatrix} x & y & z \end{bmatrix} \\
& \operatorname{tr}(t_2) = 1 \\
& t_2 = xx^T \\
& \operatorname{tr}(t_3) = 1 \\
& t_3 = yy^T \\
& \operatorname{tr}(t_4) = 1 \\
& t_4 = zz^T
\end{aligned}
$$

At this point we notice that we have been wasteful. We have introduced three more variables than necessary: t_2, t_3, and t_4. Each of these variables corresponds to t_{111}, t_{122}, and t_{133}

respectively. In other words, it is possible to reduce the mathematical program into

$$\begin{aligned}
\max_{(x,y,z,t_0)\in\mathbb{R}^7, t_1\in\mathcal{S}^3} \quad & t_0 \\
\text{st} \quad & \operatorname{tr}\left(\begin{bmatrix} 0 & -1 & -1.5 \\ -1 & 0 & -.5 \\ -1.5 & -.5 & 0 \end{bmatrix} t_1\right) + 6 \geq 0 \\
& t_1 = \begin{bmatrix} x & y & z \end{bmatrix}^T \begin{bmatrix} x & y & z \end{bmatrix} \\
& t_{111} = 1 \\
& t_{122} = 1 \\
& t_{133} = 1
\end{aligned}$$

This problem occurs since we apply the lifting procedure to each constraint separately. Since we do not coordinate each transformation, redundant variables may arise. Unfortunately, eliminating this problem proves difficult. It requires us to define a global nonconvex quadratic transformation that considers the entire problem. As a result, we may be forced to live with this redundancy. Despite this difficulty, we complete our reformulation by relaxing each rank-1 constraint into a semidefinite constraint with the command

```
let mc=apply_mp (lowrank 'Right) mc;;
```

This generates the following program

```
max     _0
over    x in Real[ 1,1 ], y in Real[ 1,1 ], z in Real[ 1,1 ],
        _0 in Real[ 1,1 ], _1 in Symmetric[ 3 ],
```

```
        _2 in Symmetric[ 1 ], _3 in Symmetric[ 1 ],
        _4 in Symmetric[ 1 ]
st
        trace([0. -1. -1.5; -1. 0. -0.5; -1.5 -0.5 0.] * _1) +
            [6.] >= _0
        {[1.] ({x[ 1 , 1 ]; {y[ 1 , 1 ]; z[ 1 , 1 ]}})'; _1}
            >=S [0. 0. 0. 0.; 0. 0. 0. 0.; 0. 0. 0. 0.;
                0. 0. 0. 0.]
        trace([1.] * _2) = [1.]
        {[1.] (x[ 1 , 1 ])'; _2} >=S [0. 0.; 0. 0.]
        trace([1.] * _3) = [1.]
        {[1.] (y[ 1 , 1 ])'; _3} >=S [0. 0.; 0. 0.]
        trace([1.] * _4) = [1.]
        {[1.] (z[ 1 , 1 ])'; _4} >=S [0. 0.; 0. 0.]
```

This corresponds to the problem

$$\begin{aligned} \max_{(x,y,z,t_0,t_2,t_3,t_4)\in\mathbb{R}^7,\, t_1\in\mathcal{S}^3} \quad & t_0 \\ \text{st} \quad & \operatorname{tr}\left(\begin{bmatrix} 0 & -1 & -1.5 \\ -1 & 0 & -.5 \\ -1.5 & -.5 & 0 \end{bmatrix} t_1\right) + 6 \geq 0 \\ & \begin{bmatrix} 1 & \begin{bmatrix} x & y & z \end{bmatrix} \\ \begin{bmatrix} x & y & z \end{bmatrix}^T & t_1 \end{bmatrix} \succeq_{\mathcal{S}_+} 0 \end{aligned}$$

$$\operatorname{tr}(t_2) = 1$$

$$\begin{bmatrix} 1 & x \\ x & t_2 \end{bmatrix} \succeq_{\mathcal{S}_+} 0$$

$$\operatorname{tr}(t_3) = 1$$

$$\begin{bmatrix} 1 & y \\ y & t_3 \end{bmatrix} \succeq_{\mathcal{S}_+} 0$$

$$\operatorname{tr}(t_4) = 1$$

$$\begin{bmatrix} 1 & z \\ z & t_4 \end{bmatrix} \succeq_{\mathcal{S}_+} 0$$

Thus, we have successfully generated a linear semidefinite program from our original formulation. Although we find our relaxation to be less than optimal, we have successfully derived one possible relaxation to a difficult problem.

7.2 Chained Singular Function

In the following discussion, we consider the simplest possible problem containing the chained singular function

$$\begin{aligned} \min_{(y,x_1,x_2,x_3,x_4)\in\mathbb{R}^5} \quad & y \\ \text{st} \quad & y \geq (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - 10x_4)^4 \end{aligned}$$

We represent this problem in our language with

```
let csf= <:mp<
    min y
    over x1 in Real[1,1], x2 in Real[1,1], x3 in Real[1,1],
         x4 in Real[1,1], y in Real[1,1]
    st y >= (x1+10.*x2)^2.+5.*(x3-x4)^2.+
                (x2-2.*x3)^4.+10.*(x1-10.*x4)^4. >>;;
```

This yields the program

```
  min     y
  over    x1 in Real[ 1,1 ], x2 in Real[ 1,1 ],
          x3 in Real[ 1,1 ], x4 in Real[ 1,1 ],
          y in Real[ 1,1 ]
  st      y >= ((x1 + [10.] * x2)^([2.]) + [5.] * (x3 - x4)^([2.]) +
              (x2 - [2.] * x3)^([4.]) + [10.] * (x1 - [10.] *
              x4)^([4.]))
```

We notice this problem contains two quartics embedded in the constraint. In order to reach them, we must expand the problem with the command

```
let csf=napply_mp (apply_mp (expand_ineq `Right)) csf;;
```

This yields the program

```
  min     y
  over    x1 in Real[ 1,1 ], x2 in Real[ 1,1 ], x3 in Real[ 1,1 ],
```

```
        x4 in Real[ 1,1 ], y in Real[ 1,1 ], _0 in Real[ 1,1 ],
        _1 in Real[ 1,1 ], _2 in Real[ 1,1 ], _3 in Real[ 1,1 ],
        _4 in Real[ 1,1 ], _5 in Real[ 1,1 ], _6 in Real[ 1,1 ],
        _10 in Real[ 1,1 ]
st      y >= (_0 + _1)
        _0 >= (_2 + _3)
        _2 >= (_5 + _6)
        _5 >= (x1 + [10.] * x2)^([2.])
        _6 >= [5.] * _10
        _10 >= (x3 - x4)^([2.])
        _3 >= (x2 - [2.] * x3)^([4.])
        _1 >= [10.] * _4
        _4 >= (x1 - [10.] * x4)^([4.])
```

This corresponds to the problem

$$\begin{aligned} \min_{(y,x_i,t_i)\in\mathbb{R}^{16}\in\mathbb{R}^5} \quad & y \\ \text{st} \quad & y \geq t_0 + t_1 \\ & t_0 \geq t_2 + t_3 \\ & t_2 \geq t_5 + t_6 \\ & t_5 \geq (x_1 + 10x_2)^2 \\ & t_6 \geq 5t_{10} \\ & t_{10} \geq (x_3 - x_4)^2 \\ & t_3 \geq (x_2 - 2x_3)^4 \\ & t_1 \geq 10t_4 \\ & t_4 \geq (x_1 - 10x_4)^4 \end{aligned}$$

At this point, we notice that we have exposed all the quartics. As a result, we can remove them with the command

```
let csf=apply_mp quartic csf;;
```

This produces the following program

```
  min     y
  over    x1 in Real[ 1,1 ], x2 in Real[ 1,1 ], x3 in Real[ 1,1 ],
          x4 in Real[ 1,1 ], y in Real[ 1,1 ], _0 in Real[ 1,1 ],
          _1 in Real[ 1,1 ], _2 in Real[ 1,1 ], _3 in Real[ 1,1 ],
          _4 in Real[ 1,1 ], _5 in Real[ 1,1 ], _6 in Real[ 1,1 ],
```

```
        _10 in Real[ 1,1 ], _17 in Real[ 1,1 ], _18 in Real[ 1,1 ]

st      y >= (_0 + _1)

        _0 >= (_2 + _3)

        _2 >= (_5 + _6)

        _5 >= (x1 + [10.] * x2)^([2.])

        _6 >= [5.] * _10

        _10 >= (x3 - x4)^([2.])

        _3 >= (_17)^([2.])

        _17 >= (x2 - [2.] * x3)^([2.])

        _1 >= [10.] * _4

        _4 >= (_18)^([2.])

        _18 >= (x1 - [10.] * x4)^([2.])
```

This coincides the with problem

$$\begin{aligned} \min_{(y,x_i,t_i)\in\mathbb{R}^{18}\in\mathbb{R}^5} \quad & y \\ \text{st} \quad & y \geq t_0 + t_1 \\ & t_0 \geq t_2 + t_3 \\ & t_2 \geq t_5 + t_6 \\ & t_5 \geq (x_1 + 10x_2)^2 \\ & t_6 \geq 5t_{10} \\ & t_{10} \geq (x_3 - x_4)^2 \\ & t_3 \geq t_{17}^2 \\ & t_{17} \geq (x_2 - 2x_3)^2 \\ & t_1 \geq 10t_4 \\ & t_4 \geq t_{18}^2 \\ & t_{18} \geq (x_1 - 10x_4)^2 \end{aligned}$$

At this point, we must make a decision. Ultimately, we wish to transform each convex quadratic into a second-order cone constraint. We can either employ this transformation now, or we can contract the problem then transform. For our purposes, we try both and compare the results. If we transform the convex quadratics now, we use the following command

```
let csf=apply_mp convex_quad csf;;
```

This generates the program

```
min     y
over    x1 in Real[ 1,1 ], x2 in Real[ 1,1 ], x3 in Real[ 1,1 ],
        x4 in Real[ 1,1 ], y in Real[ 1,1 ], _0 in Real[ 1,1 ],
        _1 in Real[ 1,1 ], _2 in Real[ 1,1 ], _3 in Real[ 1,1 ],
        _4 in Real[ 1,1 ], _5 in Real[ 1,1 ], _6 in Real[ 1,1 ],
        _10 in Real[ 1,1 ], _17 in Real[ 1,1 ], _18 in Real[ 1,1 ]
st       y >= (_0 + _1)
        _0 >= (_2 + _3)
        _2 >= (_5 + _6)
        {[1.] - ([-1.])' * _5[ 1 , 1 ]; {[2.] * [-1. -10.] *
            {x1[ 1 , 1 ] ; x2[ 1 , 1 ]}; [1.] + ([-1.])' *
            _5[ 1 , 1 ]}} >=Q [0.; 0.; 0.]
        _6 >= [5.] * _10
        {[1.] - ([-1.])' * _10[ 1 , 1 ]; {[2.] * [-1. 1.] *
            {x3[ 1 , 1 ] ; x4[ 1 , 1 ]}; [1.] + ([-1.])' *
            _10[ 1 , 1 ]}} >=Q [0.; 0.; 0.]
        {[1.] - ([-1.])' * _3[ 1 , 1 ]; {[2.] * [1.] *
            _17[ 1 , 1 ]; [1.] + ([-1.])' * _3[ 1 , 1 ]}}
            >=Q [0.; 0.; 0.]
        {[1.] - ([-1.])' * _17[ 1 , 1 ]; {[2.] * [-1. 2.] *
            {x2[ 1 , 1 ] ; x3[ 1 , 1 ]}; [1.] + ([-1.])' *
```

```
    _17[ 1 , 1 ]}} >=Q [0.; 0.; 0.]

_1 >= [10.] * _4

{[1.] - ([-1.])' * _4[ 1 , 1 ]; {[2.] * [1.] *

    _18[ 1 , 1 ]; [1.] + ([-1.])' * _4[ 1 , 1 ]}}

    >=Q [0.; 0.; 0.]

{[1.] - ([-1.])' * _18[ 1 , 1 ]; {[2.] * [-1. 10.] *

    {x1[ 1 , 1 ] ; x4[ 1 , 1 ]}; [1.] + ([-1.])' *

    _18[ 1 , 1 ]}} >=Q [0.; 0.; 0.]
```

This program matches the problem

$$\begin{array}{lll} \min\limits_{(y,x_i,t_i)\in\mathbb{R}^{18}\in\mathbb{R}^5} & y & \\ \text{st} & y \geq t_0 + t_1 & t_0 \geq t_2 + t_3 \\ & t_2 \geq t_5 + t_6 & t_6 \geq 5t_{10} \\ & t_1 \geq 10t_4 & \\ & \begin{bmatrix} 1 + t_5 \\ 2(-x_1 - 10x_2) \\ 1 - t_5 \end{bmatrix} \succeq_Q 0 & \begin{bmatrix} 1 + t_{10} \\ 2(-x_3 + x_4) \\ 1 - t_{10} \end{bmatrix} \succeq_Q 0 \\ & \begin{bmatrix} 1 + t_3 \\ 2t_{17} \\ 1 - t_3 \end{bmatrix} \succeq_Q 0 & \begin{bmatrix} 1 + t_{17} \\ 2(-x_2 + 2x_3) \\ 1 - t_{17} \end{bmatrix} \succeq_Q 0 \\ & \begin{bmatrix} 1 + t_4 \\ 2t_{18} \\ 1 - t_4 \end{bmatrix} \succeq_Q 0 & \begin{bmatrix} 1 + t_{18} \\ 2(-x_1 + 10x_4) \\ 1 - t_{18} \end{bmatrix} \succeq_Q 0 \end{array}$$

Thus, we have generated a nice extremely-sparse second-order cone program. Unfortunately, we have added 13 new decision variables. Thus, this process has been wasteful. Alternatively, before we employ this transformation, we can contract the system with the command

```
let csf=napply_mp (contract_ineq ‘Left) csf;;
```

This generates the program

```
min     y
over    x1 in Real[ 1,1 ], x2 in Real[ 1,1 ], x3 in Real[ 1,1 ],
        x4 in Real[ 1,1 ], y in Real[ 1,1 ], _4 in Real[ 1,1 ],
        _5 in Real[ 1,1 ], _6 in Real[ 1,1 ], _17 in Real[ 1,1 ],
        _18 in Real[ 1,1 ]
st      _5 >= (x1 + [10.] * x2)^([2.])
        _17 >= (x2 - [2.] * x3)^([2.])
        _4 >= (_18)^([2.])
        _18 >= (x1 - [10.] * x4)^([2.])
        y >= (_5 + _6 + (_17)^([2.]) + [10.] * _4)
        _6 >= [5.] * (x3 - x4)^([2.])
```

This representation matches the problem

$$\begin{aligned} \min_{(y,x_i,t_i)\in\mathbb{R}^{10}\in\mathbb{R}^5} \quad & y \\ \text{st} \quad & t_5 \geq (x_1 + 10x_2)^2 \\ & t_{17} \geq (x_2 - 2x_3)^2 \\ & t_4 \geq t_{18}^2 \\ & t_{18} \geq (x_1 - 10x_4)^2 \\ & y \geq t_5 + t_6 + t_{17}^2 + 10t_4 \\ & t_6 \geq 5(x_3 - x_4)^2 \end{aligned}$$

Notice that we have eliminated eight decision variables. However, also note that we could potentially eliminate t_4, t_5, and t_6 within the constraint

$$y \geq t_5 + t_6 + t_{17}^2 + 10t_4$$

but did not. This illustrates a weakness within our typing and contraction rules. The function $y, x, t \mapsto t_5 + t_6 + t_{17}^2 + 10t_4$ is not monotonic in *all* variables since it contains the term t_{17}^2. However, it is monotonic in t_4, t_5, and t_6. Thus, if we want to correctly eliminate these variables, we must make an addition to our type system. Instead of monitoring whether a function is monotonic in all variables, we must keep track of the monotonicity in each variable. Despite this difficulty, we can still transform this problem into a second-order cone program with the command

```
let csf=apply_mp convex_quad csf;;
```

This produces the following program

```
  min     y
  over    x1 in Real[ 1,1 ], x2 in Real[ 1,1 ], x3 in Real[ 1,1 ],
          x4 in Real[ 1,1 ], y in Real[ 1,1 ], _4 in Real[ 1,1 ],
          _5 in Real[ 1,1 ], _6 in Real[ 1,1 ], _17 in Real[ 1,1 ],
          _18 in Real[ 1,1 ]
  st      {[1.] - ([-1.])' * _5[ 1 , 1 ]; {[2.] * [-1. -10.] *
              {x1[ 1 , 1 ] ; x2[ 1 , 1 ]}; [1.] + ([-1.])' *
```

```
    _5[ 1 , 1 ]}} >=Q [0.; 0.; 0.]
{[1.] - ([-1.])' * _17[ 1 , 1 ]; {[2.] * [-1. 2.] *
    {x2[ 1 , 1 ] ; x3[ 1 , 1 ]}; [1.] + ([-1.])' *
    _17[ 1 , 1 ]}} >=Q [0.; 0.; 0.]
{[1.] - ([-1.])' * _4[ 1 , 1 ]; {[2.] * [1.] *
    _18[ 1 , 1 ]; [1.] + ([-1.])' * _4[ 1 , 1 ]}}
    >=Q [0.; 0.; 0.]
{[1.] - ([-1.])' * _18[ 1 , 1 ]; {[2.] * [-1. 10.] *
    {x1[ 1 , 1 ] ; x4[ 1 , 1 ]}; [1.] + ([-1.])' *
    _18[ 1 , 1 ]}} >=Q [0.; 0.; 0.]
{[1.] - ([-1.; 10.; 1.; 1.])' * {y[ 1 , 1 ]; {_4[ 1 , 1 ];
    {_5[ 1 , 1 ]; _6[ 1 , 1 ]}}}; {[2.] * [1.] *
    _17[ 1 , 1 ]; [1.] + ([-1.; 10.; 1.; 1.])' *
    {y[ 1 , 1 ]; {_4[ 1 , 1 ]; {_5[ 1 , 1 ];
    _6[ 1 , 1 ]}}}}} >=Q [0.; 0.; 0.]
{[1.] - ([-1.])' * _6[ 1 , 1 ]; {[2.] * [-2.2360679775
    2.2360679775] * {x3[ 1 , 1 ]; x4[ 1 , 1 ]}; [1.] +
    ([-1.])' * _6[ 1 , 1 ]}} >=Q [0.; 0.; 0.]
```

This corresponds to the problem

$$
\begin{aligned}
\min_{(y,x_i,t_i)\in\mathbb{R}^{10}\in\mathbb{R}^5} \quad & y \\
\text{st} \quad & \begin{bmatrix} 1+t_5 \\ 2(-x_1-10x_2) \\ 1-t_5 \end{bmatrix} \succeq_Q 0 \qquad \begin{bmatrix} 1+t_{17} \\ 2(-x_2+2x_3) \\ 1-t_{17} \end{bmatrix} \succeq_Q 0 \\
& \begin{bmatrix} 1+t_4 \\ 2t_{18} \\ 1-t_4 \end{bmatrix} \succeq_Q 0 \qquad \begin{bmatrix} 1+t_{18} \\ 2(-x_1+10x_4) \\ 1-t_{18} \end{bmatrix} \succeq_Q 0 \\
& \begin{bmatrix} 1+y-10t_4-t_5-t_6 \\ 2t_{17} \\ 1-y+10t_4+t_5+t_6 \end{bmatrix} \succeq_Q 0 \quad \begin{bmatrix} 1+t_6 \\ 2(-\sqrt{5}x_3+\sqrt{5}x_4) \\ 1-t_6 \end{bmatrix} \succeq_Q 0
\end{aligned}
$$

Hence, we have successfully generated a linear second-order cone program from the chained-singular function. As a final note, this problem highlights one possible challenge when employing transformations during the modeling process. Depending on what order we expand, transform, and contract a constraint we arrive at different transformations. These formulations may possess different properties. However, by automating these transformations, we can quickly check and determine which formulation is best for a particular application.

7.3 Type Checking

In the following discussion, we give a two short examples of the type-checker and explore the accuracy of its analysis.

We begin with a simple function $x, y \mapsto (x^2 + y^2)^{.5}$. Of course, we recognize this as the two-norm of the vertical concatenation of x and y. Thus, the result must be convex . We represent this function in our language with

```
let sigma=[ <:var< x in Real[1,1] >> ; <:var< y in Real[1,1] >> ];;
let twonorm= <:exp< (x^2.+y^2.)^2. >> ;;
```

Then, we type-check the expression with the command

```
let _=type_expr sigma twonorm;;
```

This yields the result

```
# - : Type_mathprog.texpr =
TModel ([[(‘Convex, ‘Bottom, ‘Bottom, ‘Real)]], 1, 1, ‘Symmetric)
```

In other words, we assert that we have specified a convex function with an unknown polynomality and monotonicity.

This example highlights one of the strengths of our analysis. As an alternative to our approach, we could simply catalog the convexity and monotonicity of each function such as addition, exponentiation, etc. Then, we know that the function $x \mapsto f(g(x))$ is convex when f is convex, increasing and g is convex. However, notice that the function the function

$x \mapsto x^{.5}$ is not convex. In fact, this function is concave for $x \geq 0$. Thus, this alternative scheme can not correctly assert the convexity of this function.

In a similar manner, we can express the Frobenius norm of a matrix with the following command

```
let sigma=[ <:var< x in Real[5,3] >> ];;
let frobnorm= <:exp< (trace(trans(x)*x))^0.5 >>;;
```

After we type check the result, we see that

```
- : Type_mathprog.texpr =
TModel ([[(‘Convex, ‘Bottom, ‘Bottom, ‘Real)]], 1, 1, ‘Symmetric)
```

Thus, we can assert the convexity of this function. This example demonstrates that we correctly type difficult expressions such as `trans(x)*x`. Recall, we must represent the type of this expression with a matrix of properties. Then, even after taking the trace and square root of the result, we prove convexity.

Chapter 8

Conclusions

During our discussion, we have accomplished the following. We have developed a grammar suitable for representing a broad class of mathematical programs. Next, we have designed a type system which allows us to constructively prove properties about our problem. Based on this structure, we have developed a system of semantics which give an accurate mathematical depiction of our language. This foundation has allowed us to specify a series of transformations that manipulate our problem. Finally, we have concluded our discussion by considering three different examples.

The grammar specifies the internal structure of our language. We have found that a reduced version of lambda calculus that does not contain abstractions, but does include constants, allows us to be surprisingly expressive. This simplicity has been somewhat offset by the complexity of our types. Nonetheless we have found this complexity necessary to accurately depict the structure of a program.

The type system allows us to automatically characterize the mathematical properties of our problem. We have found that the most interesting typing rules occur during function application. This application allows us to combine variables and constants into more com-

plicated functions. It is at this point where we determine the mathematical properties of the resulting composition. In practice, this reduces to cataloging the different scenarios that can occur.

The semantics give the mathematical characterization of our language. We have found the key to this process has been not to view operations such as addition as a function that maps two concrete numbers to another. Rather, we view each function as a composition. In other words, we view addition as a function that accepts two functions as its arguments, then produces another function where we add the two arguments together. This allows us to establish a direct connection between our types, which represent mathematical properties, and our expressions. Once we define our semantics, we prove soundness of our language. This gives us confidence that our type system accurately depicts the mathematical properties of our problem.

We have used transformations as our primary tool for taking advantage of the underlying structural properties of our problem. In order to simplify this process, we have established a system of transformations that expand and contract the problem. These transformations allow us to expose many of the problem's hidden structural features. Once we expose these features, we can reformulate the problem into a more computable form.

Our three examples have been chosen to demonstrate the following. The max-cut problem demonstrates the necessity of transformations from a modeling perspective. Simply, the semidefinite relaxation appears nothing like the original formulation. Thus, asking a user to manually manipulate the problem runs the risk of introducing errors. Second, the

chained singular function demonstrates the necessity of program analysis during program manipulation. Although this problem contains many convex, quadratic functions, they are not presented in a nice canonical form. Thus, we must ascertain their presence during our program analysis. Finally, the two and Frobenius norms demonstrate the power of our type system. Systems such as CVX and YALMIP use a much simpler scheme which can not prove the convexity of these functions. While our system is more complicated, it provides us with more flexibility.

8.1 Future Work

Our ultimate goal lies in extending these techniques to general purpose codes. In other words, we believe that we can prove whether a function written in C or Fortran is convex, polynomial, monotonic, or possesses some other mathematical property. Certainly, our analysis becomes more complicated. For example, consider the routine

```
float polynomial(float x,int n){
    float y=1;
    for(int i=0;i<n;i++)
        y*=x;
    return y;
}
```

Depending on the value of `n`, the problem may be linear, quadratic, or neither. Regardless, we know this function represents some sort of polynomial. Thus, we don't expect that

we can prove a certain property in all cases. Rather, we believe that we can prove these properties in many different, common situations.

Once we can prove properties about a general routine, we may transform it. This becomes especially interesting when generalizing the expansion and contraction transformations. Essentially, this amounts to decomposing a routine into several pieces which expose structure. This work is closely related to compiler optimizations. However, instead of transforming a routine so that it runs faster, we design transformations that alter the mathematical properties.

Each of these developments forces us to enrich the type system. As routines become more complicated, we require additional information to correctly classify their properties. For example, if we could guarantee that the variable $x \geq 0$, we could assert that the function $x \mapsto 1/x$ is convex. Which properties we analyze depends on the structure of the problem. Nonetheless, we have established a flexible typing scheme which extends to other properties.

Within combinatorial optimization, we have established a series of transformations that allow us to automatically relax an integer program into a semidefinite program. For example, we can apply the same techniques used to transform the max-cut problem to the graph partitioning problem. While these relaxations frequently give poor bounds, their overall utility remains unknown. Simply, generating these relaxations has been so prohibitive in the past, that researchers have been unable to comprehensively explore the accuracy of these relaxations. It may be that these relaxations are well suited toward branch and bound

algorithms. However, we must make generating these problems far easier before we can adequately explore this possibility.

Within global optimization, these techniques show promising applicability. One common technique to global optimization involves approximating each routine by a piecewise linear function. This results in an integer program that approximates the true problem. We can use our transformational techniques to automatically derive this formulation.

This same idea applies to problems within robust optimization. One possible robust formulation of a linear program results in a second-order cone program. Similarly, a robust formulation of a second-order cone program results in a semidefinite program. We can use our methods to automatically derive each of these formulations.

Finally, we should not restrict ourselves to only considering mathematical programs. Discovering and exploiting hidden structure within a problem has direct implications to differential equations, dynamical systems, and other mathematical models. In the end, our goal lies in developing techniques that allow us to analyze and transform each of these problems.

Bibliography

[1] E. D. Andersen and K. D. Andersen. The MOSEK Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm. In H. Frenk, K. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, volume 33 of *Applied Optimization*, pages 197–232. Kluwer Academic Publishers, 1999.

[2] M. Aponte, A. Laville, M. Mauny, A. Suarez, and P. Weis. The CAML reference manual. Technical Report 121, INRIA, Domaine de Volceau, Rocquencourt, 1990.

[3] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. Report on the algorithmic language ALGOL 60. *Commun. ACM*, 3(5):299–314, 1960.

[4] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. Revised report on the algorithm language ALGOL 60. *Commun. ACM*, 6(1):1–17, 1963.

[5] John Backus. The history of Fortran I, II, and III. pages 25–74, 1981.

[6] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An ap-

plication of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988.

[7] E. M. L. Beale. Matrix generators and output analyzers. In *Proceedings of the Princeton Symposium on Mathematical Programming*, pages 25–36. Princeton University Press, 1970.

[8] Steven J. Benson and Yinyu Ye. DSDP4-a software package implementing the dual-scaling algorithm for semidefinite programming, 2002. Technical Report ANL/MCS-TM-255.

[9] H. Bhargava, R. Krishnan, and P. Piela. Formalizing the semantics of ASCEND. *Proceedings of the 27th Hawaii International Conference on the System Sciences.*, pages 505–516, 1994.

[10] Hemant K. Bhargava, Ramayya Krishnan, and Peter Piela. On formal semantics and analysis of typed modeling languages: An analysis of ascend. *INFORMS J. on Computing*, 10(2):189–208, 1998.

[11] Johannes Bisschop and Alexander Meeraus. On the development of a general algebraic modeling system in a strategic planning environment. In *Mathematical Programming Study 20*, pages 1–29. North-Holland Publishing Company, 1982.

[12] Robert E. Bixby. ILOG CPLEX. http://www.ilog.com/products/cplex/.

[13] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. In *Acta Numerica*, pages 1–51. Cambridge University Press, 1995.

[14] J. M. P. Booler. A method for solving crew scheduling problems. *Operational Research Quarterly (1970-1977)*, 26(1):55–62, 1975.

[15] Brian Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software 11*, pages 613–623, 1999.

[16] Michel Bréal. *Semantics: Studies in the Science of Meaning*. Henry Holt and Company, 1900.

[17] Michael R. Bussieck and Alex Meeraus. *General Algebraic Modeling System (GAMS)*, pages 137–157. Springer, 2003.

[18] Alonzo Church. A set of postulates for the foundation of logic. *The Annals of Mathematics*, 33(2):346–366, 1932.

[19] Alonzo Church. *The Journal of Symbolic Logic*, 5(2):56–68, 1940.

[20] Andrew R. Conn, Nicholas I. M. Gould, and Philippse L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50(182):399–430, 1988.

[21] Robert L. Constable, Stuart F. Allen, H. M. Bromley, Walter Rance Cleaveland, J. F. Cremer, Robert W. Harper, Douglas J. Howe, Todd B. Knoblock, Nax P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall:NJ, 1986.

[22] G. Cousineau, P. L. Curien, and M. Mauny. The categorical abstract machine. *Lecture Notes in Computer Science*, 201:50–64, 1985.

[23] H. B. Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences of the United States of America*, 20(11):584–590, 1934.

[24] Haskell B. Curry. The combinatory foundations of mathematical logic. *The Journal of Symbolic Logic*, 7(2):49–64, 1942.

[25] G. B. Dantzig. Programming in a linear structure. USAF, Washington D.C., 1948.

[26] G. B. Dantzig. Linear programming. *Operations Research*, 50(1):42–47, 2002.

[27] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Paulin, and B. Werner. The coq proof assistant user's guide, version 5.6. Technical Report 134, INRIA, December 1991.

[28] R. J. Duffin. Cost minimization problems treated by geometric means. *Operations Research*, 10(5):668–675, 1962.

[29] R. J. Duffin. Dual programs and minimum cost. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):119–123, 1962.

[30] R. J. Duffin, E. L. Peterson, and C. Zener. *Geometric Programming– Theory and Application*. John Wiley & Sons, Inc., 1967.

[31] Anthony V. Fiacco and Garth P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley & Sons, Inc., 1968.

[32] Robert Fourer. Modeling languages versus matrix generators for linear programming. *ACM Transactions on Mathematical Software*, 9(2):143–183, 1983.

[33] Robert Fourer, David M. Gay, and Brian W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36:519–554, 1990.

[34] Robert Fourer, David M. Gay, and Brian W. Kernighan. *Design Principles and New Developments in the AMPL Modeling Language*, pages 105–135. Springer, 2003.

[35] Gottlob Frege. On sense and nominatum. In *Readings in Philosophical Analysis*, pages 85–102. 1949. Translated by Herbert Feigl from, "Ueber Sinn und Bedeutung," Zeitschr. f. Philos. und Philos. Kritik; 100, 1892.

[36] K. Fujisawa and M. Kojima. SDPA(semidefinite programming algorithm) : User's manual, 1995.

[37] P. Gahinet, A. Nemirovskii, A.J. Laub, and M. Chilali. The LMI control toolbox. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, volume 3, pages 2038–2041, 1994.

[38] U. M. Garcia-Palomares and O. L. Mangasarian. Superlinearly convergent quasi-newton algorithms for nonlinearly constrained optimization problems. *Mathematical Programming*, 11(1):1–13, 1976.

[39] Philip E. Gill, Walter Murray, Michael A. Saunders, J. A. Tomlin, and Margaret H. Wright. On projected newton barrier methods for linear programming and an equiv-

alence to karmarkar's projective method. *Mathematical Programming: Series A and B*, 36(2):183–209, 1986.

[40] Michael J. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. *Edinburgh LCF A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[41] Michael Grant. *Disciplined Convex Programming*. PhD thesis, Stanford University, 2005.

[42] Michael Grant, Stephen Boyd, and Yinyu Ye. Disciplined convex programming. *Global Optimization: From Theory to Implementation*, pages 155–200, 2006.

[43] Magnus R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969.

[44] Sa Neung Hong and Michael V. Mannino. Formal semantics of the unified modeling language $\mathcal{L}_u$. *Decision Support Systems*, (13):263–293, 1995.

[45] W. A. Howard. Formulae-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.

[46] N. K. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[47] W. Karush. Minima of functions of several variables with inequalities as side constraints. Master's thesis, University of Chicago, 1939.

[48] L.G. Khachian. A polynomial algorithm in linear programming. *Soviet Math. Dokl.*, 20:191–194, 1979. Translated from Russian.

[49] S. C. Kleene and J. B. Rosser. The inconsistency of certain formal logics. *The Annals of Mathematics*, 36(3):630–636, 1935.

[50] Donald E. Knuth. The remaining trouble spots in ALGOL 60. *Commun. ACM*, 10(10):611–618, 1967.

[51] Kazuhiro Kobayashi, Sunyoung Kim, and Masakzu Kojima. Sparse second order cone programming formulations for convex optimization problems. Technical report, Department of Mathematical and Computing Sciences Tokyo Institute of Technology, 2007.

[52] H. W. Kuhn and A. W. Tucker. Nonlinear programming. *Proceedings of Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, 1951.

[53] P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, January 1964.

[54] P. J. Landin. A correspondence between ALGOL 60 and church's lambda-notation: Part i. *Communications of the ACM*, 8(2):89–101, February 1965.

[55] P. J. Landin. A correspondence between ALGOL 60 and church's lambda-notation: Part ii. *Communications of the ACM*, 8(3):158–165, March 1965.

[56] J. Löfberg. YALMIP : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[57] P. Lucas and K. Walk. On the formal description of PL/I. *Annual Review in Automatic Programming*, 6:105–182, 1969.

[58] Michel Mauny and Ascánder Suárez. Implementing functional languages in the categorical abstract machine. In *LFP '86: Proceedings of the 1986 ACM conference on LISP and functional programming*, pages 266–278, New York, NY, USA, 1986. ACM.

[59] A. Nemirovskii and P. Gahinet. The projective method for solving linear matrix inequalities. *American Control Conference, 1994*, 1:840–844 vol.1, 29 June-1 July 1994.

[60] Y. Nesterov and A. Nemirovski. A general approach to polynomial-time algorithms design for convex programming. Technical report, Central Economic and Mathematical Institute, USSR Academy of Sciences, Moscow, 1998.

[61] Laurel Neustadter. Formalization of expression semantics for an executable modeling language. In *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*, pages 492–504, 1994.

[62] P. Piela, T. Epperly, K. Westerberg, and A. Westerberg. ASCEND: An object-oriented computer environment for modeling and analysis: The modeling language. *Computers and Chemical Engineering*, 15(1):53–72, 1991.

[63] Didier Rémy and Jérôme Vouillon. Objective ML: A simple object-oriented extension of ml. In *Proceedings of the 24th ACM Conference on Principles of Programming Languages*, pages 40–53, Paris, France, January 1997.

[64] Bertrand Russell. *The Principles of Mathematics*. George Allen & Unwin Ltd, 1903.

[65] Bertrand Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30(3), 1908.

[66] Dana Scott. Mathematical concepts in programming language semantics. In *1972 Spring Joint Computer Conference*, pages 225–234. AFIPS Press, 1972.

[67] Dana Scott and Christopher Strachey. Toward a mathematical semantics for computer languages. In *Proceedings of the Symposium on Computers and Automata*, pages 19–46, 1971.

[68] Christopher Strachey. The varieties of programming languages. In *Proceedings of the International Computing Symposium 1972*, pages 222–233, 1972.

[69] J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. Special issue on Interior Point Methods (CD supplement with software).

[70] Alfred Tarski. The semantic conception of truth and the foundation of semantics. In *Readings in Philosophical Analysis*, pages 52–84. 1949. Reprinted from, ”Symposium on Meaning and Truth”, Philosophy and Phenomenological Research, Vol. IV, 1944.

[71] K. C. Toh, M. J. Todd, and R. Tutuncu. SDPT3–a matlab software package for semidefinite programming. *Optimization Methods and Software*, 11:545–581, 1999.

[72] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[73] Fernando Vicuna. *Semantic Formalization in Mathematical Modeling Languages*. PhD thesis, University of California Los Angeles, 1990.

[74] R. B. Wilson. *A Simplicial Algorithm for Concave Programming*. PhD thesis, Harvard University, 1963.

[75] Margaret H. Wright. The interior-point revolution in optimization: History, recent developments, and lasting consequences. *American Mathematical Society*, 42(1):39–56, 2004.